Safe Model Based Policy Search

Kyriakos Polymenakos

Kellogg college

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2020

Acknowledgements

First I would like to thank my supervisors, Stephen Roberts and Alessandro Abate for their guidance and support throughout the years of my research. They showed me remarkable trust and understanding, allowing me to explore different ideas, and provided me with inputs and answers when most needed. They worked alongside me through numerous tight deadlines, correcting my manuscripts and helping me hone my ideas time and time again.

I want to thank Wendy for being an incredible course administration, helping me keep track of deadlines and navigate university processes, which I never would have managed on my own. I am also grateful to all my lab mates (in both labs!) for the stimulating discussions, the inevitable rants and for overall making me want to come in every day. I am thankful to my collaborators Nikitas, Andrea and Luca for making our work together fruitful and enjoyable.

During these years I was lucky enough to make great new friends, that made life beyond the university exciting. Thanks to Leo, Paula, Nikitas, Ada, Bernardo, Gabriele, Henry, Tom, Ivan and Ahsan. I am also grateful to my earlier friends, who stuck around for many years, despite the long absences.

Finally, I am forever thankful to my family, mother, father and sister, and to Thaleia, without whom I wouldn't be able to get here.

Abstract

In this dissertation we focus on safe model based policy search, a subfield of reinforcement learning with two main objectives: data efficiency and safety. To achieve data efficient learning, we use Gaussian process regression to model the dynamics of unknown non-linear systems. The flexibility and probabilistic nature of GPs, along with their useful mathematical properties that often allow for closed form calculations, facilitate building accurate models efficiently, and using these models to optimise control policies for the underlying systems. Furthermore, our safety objective, also probabilistic in nature, is formalised as predefined state space constraints. The model's predictions are used to certify the safety of a candidate policy before deploying it on the system, and we thus manage to avoid constraint violations while training. We present an open source, openly available software tool implementing our proposed algorithm for safe and data efficient policy search. Furthermore, we propose a novel method for planning over multiple time steps with Gaussian processes, and provide formal guarantees bounding the predictive uncertainty. We consider safety and data efficiency critical challenges for the wider adoption of reinforcement learning algorithms, and we hope that our contributions will be useful in this effort.

Contents

List of Figures

List of Tables

1

Introduction

Reinforcement learning has seen a huge surge in research interest in recent years. Independent improvements in hardware, software and infrastructure, along with the introduction of novel algorithms, have made tackling increasingly complex problems possible. Two important challenges that persist are data efficiency and safety. Data efficiency allows algorithms to reach their learning objectives with limited data, a consideration that has always been significant but becomes imperative in specific domains, such as physical systems, financial applications and user-facing systems, which are all critical for the wide adoption of reinforcement learning. Safety, when referring to a learning system, can be a hard to define, ambiguous and multifaceted property. It can be interpreted as a reduction in the variance of the performance of a system, as operational constraint specifications, or as a formally defined property in a temporal logic, among others options. Nevertheless, some concept of safety, or its opposite, risk, is present in most domains of application, often with crucial importance. In this thesis we focus on safe model based policy search to

address both of these concerns.

In the following, we assume that we need to control a system without having a priori access to a model of its dynamics. Instead, we learn a probabilistic model of the system dynamics, based on the observed data. The model allows us to predict the system's behaviour under different control policies. This serves a dual purpose: it allows us to guide the policy search to promising, high-performing policies and therefore increases data efficiency; moreover, it enables the assessment of a candidate policy's safety (which for us takes the form of state space constraints) before it is deployed on the unknown system. Finally, as reliable predictions are a prerequisite of ensuring safety via the model, we introduce formal guarantees for multi-step ahead predictions with Gaussian processes, the class of models used throughout this thesis.

Our main contributions are three:

- We present a new software tool, based on the PILCO framework, implemented in Python and taking advantage of popular and efficient machine learning libraries. The new tool is modular, easy to use and expand, and publicly available.

- We develop a new algorithm expanding the PILCO framework, that incorporates state space constraints and promotes the synthesis of safe controllers that respect the introduced constrains.

- We provide a new method for propagating uncertainty in multi step ahead planning with Gaussian processes, and provide formal bounds for its accuracy.

With the introduction of the SafePILCO tool we aim to facilitate the use of the PILCO framework both by the research community and in industrial and

educational applications. Modern RL research results are often hard to reproduce, due to the dependence of performance on seemingly insignificant implementation decisions, the non-trivial software engineering effort required for implementation and the high computational demands, that in practice often require expensive hardware infrastructure, or equivalently costly access to computational resources on the cloud. PILCO has comparatively more modest requirements in compute, and by reworking the implementation, we aim to make experimentation with model based RL more accessible, thus encouraging future research in this direction.

A challenging aspect of applying RL in practical problems is related to safety, and this is the focus of our second and third contributions. We first incorporate state space constraints in the PILCO framework and we develop a method to estimate constraint violation risk for a specific policy, before it is implemented. Two fundamental objectives are at tension here: the main learning task of synthesizing a high performing policy by policy exploration and data collection, and keeping the system safe by minimizing or ideally eliminating constraint violations. Our method accepts a natural definition of an acceptable risk threshold in probabilistic terms, which, under certain assumptions, is respected throughout training and by the resulting policy.

Finally our third contribution re-examines a commonly used approximation technique used in for multi-step prediction with GPs, including PILCO, moment matching. While useful, and reasonably accurate in the average case, moment matching can introduce unaccounted for errors in the uncertainty quantification of the prediction. Instead, we provide formal probabilistic guarantees, at the expense of additional computational demands. In safety critical applications this can be a significant advantage.

1.1 Assumptions

Here we outline the main assumptions of our work, examine them, and discuss them critically along with alternatives. Throughout, we rely on the learnt GP models both for their predictions and for the uncertainty quantification they provide. We do not try to estimate or bound the modelling error itself; instead we depend on the uncertainty expressed as the variance of the GP. Thus, we can focus on the model, and make sure we take full advantage of the information it captures. To estimate the modelling error directly, one needs to pose assumptions on the properties of the system the GP is modelling [78]. Such work can be complementary to ours, and developing methods that use both is a promising direction for future research.

Furthermore, throughout the thesis we opt to use very limited prior information in general, and especially so towards modelling and policy design. Our models are learnt from scratch, despite the fact that for many systems some approximate model could capture part of their dynamics. An alternative approach would be to use the approximate model in combination with a trainable model that learns to predict the difference between the real system and the approximate model. We chose to focus on the GP model itself and develop a methods that are agnostic to the real system and independent of prior models to keep the applicability of our method as wide as possible, and to focus our efforts on the key issues that arise with the use of learnt models. We leave the hybrid model approach for future work, despite considering it potentially significant in numerous practical applications.

Beyond model construction, prior information can be available regarding the policy itself. There can be available data of an expert (human or not) operating the system, without access to a parameterised policy (as is often the

case in imitation learning), or even a previous policy that needs to fine tuned. We assume that we start fresh with no such prior information. However, as our models are trained from an initial dataset, in contrast with the methodology we explore in this work, that is constructing this dataset from random actions, any dataset that is available before training can be used. Additionally, the policy could also be trained, in a supervised manner, to match the actions chosen by the expert system, and the resulting trained policy be used as the initialisation point for the model based policy search approach we pursue.

Finally, it is worth highlighting here that we assume that the real states of the system are observable. Dealing with systems with partial observability is very important, but in combination with our previous assumptions of very limited prior information about the system, and the safety requirements we aim to satisfy, create a very challenging problem. Tackling all out once would either restrict the scale of the environments the method can practically be applicable for drastically, leaving in its scope only small toy problems of educational or illustrative value, or would otherwise require particularly strong assumptions on their properties. For these reasons, such systems are left outside of the scope of this thesis.

1.2 Structure

In Chapter 2, we introduce some of the key notions used throughout the dissertation with an emphasis on the mathematical tools that facilitate our analyses. We also review the PILCO framework, which we further investigate and extend in later chapters. In Chapter 3, we review the relevant literature, focusing first on Reinforcement learning in general and safe model based policy search in particular, before reviewing different definitions of safety and the

associated methods. In Chapter 4, we present our software tool, giving more insight into the workings of PILCO. We evaluate the tool in a number of widely used benchmark environments. In chapter 5 we extend the PILCO algorithm, adding constraints and the algorithmic changes needed to enforce them. In Chapter 6 we introduce our new algorithm for iterative prediction with Gaussian processes, and show how bounds can be calculated for the predictive uncertainty and its evolution over multiple time steps. Finally, in Chapter 7 we summarise our work and conclude the thesis.

1.3 Publications

Parts of the work that comprises this thesis, and mainly Chapters 4, 5 and 6 have been been presented in peer reviewed conference proceedings, and, in certain cases, preliminary versions were presented in conference workshops. In chronological order:

- A workshop paper was presented in the workshop on Transparent and Interpretable Machine Learning in Safety Critical Environments, at NIPS 2017 (no proceedings published), as: Kyriakos Polymenakos, Alessandro Abate, Stephen Roberts. "Safe Policy Search with Gaussian Process Models"[1].

- A conference paper was presented in AAMAS on our work on Safe PILCO (Chapter 5): Kyriakos Polymenakos, Alessandro Abate, and Stephen Roberts. "Safe Policy Search Using Gaussian Process Models". In: *Proceedings of the 18th International Conference on Autonomous Agents and Multi Agent Systems*. IFAAMS. 2019, pp. 1565–1573.

[1] `https://sites.google.com/view/timl-nips2017/submissions`

- An extended version of the previous AAMAS paper is available on Arxiv: Kyriakos Polymenakos, Alessandro Abate, and Stephen Roberts. *Safe Policy Search with Gaussian Process Models*. 2019. arXiv: 1712.05556.

- Preliminary results on our work on safe iterative planning with Gaussian processes were presented in the workshop on Safety and Robustness in Decision Making, at NeurIPS 2019 (no proceedings published), as: Kyriakos Polymenakos, Luca Laurenti, Andrea Patane, Jan-Peter Calliess, Luca Cardelli, Marta Kwiatkowska, Alessandro Abate, Stephen Roberts. "Safety Guarantees for Planning Based on Iterative Gaussian Process"[2].

- The software tool, described in Chapter 4 and used for the experiments of Chapter 5, was presented in QEST 2020 as a software tool demonstration: Kyriakos Polymenakos, Nikitas Rontsis, Alessandro Abate, and Stephen Roberts. "SafePILCO: a software tool for safe and data-efficient policy synthesis". In: *International Conference on Quantitative Evaluation of Systems*. Springer. 2020.

- A conference paper, corresponding to the material of Chapter 6, has been accepted for publication at CDC 2020: Kyriakos Polymenakos, Luca Laurenti, Andrea Patane, Luca Cardelli, Marta Kwiatkowska, Alessandro Abate, and Stephen Roberts. "Safety guarantees for iterative predictions with Gaussian Processes". In: *Proceedings of the IEEE Conference on Decision and Control*. IEEE Xplore. 2020.

[2] https://sites.google.com/view/neurips19-safe-robust-workshop

1.4 Notation

An important note on notation, we use a slightly unusual convention to discern between scalar, vectors and matrix valued variables and random variables. Scalars are denoted with lowercase italic characters x, vectors with lowercase bold italics $\boldsymbol{x}$ and matrices bold capital italics $\boldsymbol{X}$. For the corresponding instances of random variables we use x, $\mathbf{x}$ and $\mathbf{X}$.

2

Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is a computational paradigm that allows an agent, acting in an environment, to improve its performance over time based on feedback provided through rewards. In a common taxonomy of machine learning approaches, reinforcement learning represents the third machine learning branch, alongside supervised and unsupervised learning. In the supervised learning paradigm, the dataset is comprised of inputs and the (possibly noisy) outputs; the model needs to fit those outputs and generalise to unseen data coming from the same data generating distribution. The model's evaluation usually involves some comparison with the provided outputs, despite the differences between the different metrics employed. In the typical unsupervised learning setting there are no outputs provided, and the machine learning model commonly seeks to identify some structure underlying the data, such as data clusters, principal components, or the presence of anomalies. Evaluating

unsupervised learning models tends to be more subtle, due to the lack of outputs to compare against. In reinforcement learning, the reward provides a metric for the agent to maximise, but it doesn't provide exemplary outputs. The reward constitutes an external metric for the reinforcement learning model, which can straightforwardly be used for model evaluation and comparison. Still, in contrast with supervised learning, the reward does not directly provide the RL model with the actions or behaviour to follow. In that sense reinforcement learning takes an intermediate spot on a spectrum between supervised and unsupervised learning: the rewards collected provide some supervision to the RL agent, but no direct outputs for a given input.

2.1.1 The setup and problem formulation

To ground RL theoretically, we introduce the most common mathematical framework used for RL, the Markov Decision Process (MDP). A MDP is a tuple $< \mathcal{X}, \mathcal{U}, \mathcal{T}, \mathcal{R} >$, which includes the state space $\mathcal{X}$, the control input or action space $\mathcal{U}$, the stochastic transition function $\mathcal{T}$ and the reward function $\mathcal{R}$.

The state space is the set of all possible states of the environment. Throughout this thesis we work in continuous spaces, so we assume $\mathcal{X} \subseteq \mathbb{R}^{N_s}$, where N_s is the number of dimension of the state space. Similarly the action (or input) space is the space of all possible actions, $\mathcal{U} \subseteq \mathbb{R}^{N_a}$, with N_a the dimensionality of the action space. The transition function of the MDP, for a given pair of state and action gives a stochastic transition to the state of the environment at the next time interval, $T : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$. Instead of a stochastic transition function, the MDP can also be defined using the transition probabilities $\mathcal{P}_a$, that give a probability distribution over the next state, given the current state and input $\mathcal{P}_a : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to [0, 1]$, $\mathcal{P}_a(x, u, x') = p(x'|x, u)$. The reward function $\mathcal{R}$,

$\mathcal{R} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, assigns a numerical value to the state of the environment and the control input at each time step, providing valuable feedback to the agent.

The MDP framework is abstract and adaptable [140] and many variations exist in the literature. These are the core components that one way or another are shared between the wide majority of variations. Some other components that are often present include the initial distribution ρ_0, that is a probability distribution over the state space, which the first state follows, $\mathrm{x}_0 \sim p(x_0) = \rho_0, \rho_0 : \mathcal{X} \to [0, 1]$; some tasks are characterised as *episodic*, where the agent's interaction with the environment can be broken down into independent sequences of transitions, known as episodes; the number of transitions in an episode (and sometimes the time duration of an episode) is called the horizon, T; a set of terminal states $\mathcal{G} \subset \mathcal{X}$ upon which, if reached, the episode terminates; and finally the *return*, which refers to the sum of rewards during an episode.

It's worth noting that there are two equally popular types of notation used in the RL community, the one we outlined above and an alternative, using $\mathcal{S}$ and $\mathcal{A}$ for state and input (action) spaces. Similarly, with respect to terminology, the agent's policy (and sometimes the agent itself) is referred to as the controller and the environment as the the controlled system. In general we stick with the mathematical notation presented in the MDP definition, but we use both sets of terms interchangeably.

2.2 Gaussian Processes

Gaussian Processes (GPs) are stochastic processes widely used in Machine Learning [118, 18], from Bayesian optimisation and time-series forecasting to reinforcement learning [108, 135, 119, 34]. In this work we use GPs extensively to model the dynamics (the transition probabilities) of unknown systems. For-

mally:

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

Let $x \in \mathbb{R}$ and the Gaussian process map $x \mapsto \mathrm{f}(x) \in \mathbb{R}$. The mean function $m(x)$ and the covariance function $k(x, x')$ are enough to fully specify a Gaussian process [118]. For a real process $\mathrm{f}(x)$ they can be defined as

$$m(x) = \mathbb{E}[\mathrm{f}(x)], \tag{2.1}$$

$$k(x, x') = \mathbb{E}[(\mathrm{f}(x) - m(x))(\mathrm{f}(x') - m(x'))]. \tag{2.2}$$

We may then write the GP as

$$\mathrm{f}(x) \sim \mathcal{GP}(m(x), k(x, x')). \tag{2.3}$$

The mean function $m(x)$ is often assumed to be zero for all x, with no loss of generality. That allows us to focus on the covariance function $k(x, x')$ as the sole element that specifies a GP. With the covariance function we can calculate the covariance between any two of the random variables of Definition 1 that form the GP. Note that this definition does not restrict the number of random variables that can be used; indeed, in most interesting cases that number is infinite. After all, a finite number of random variables that are jointly Gaussian distributed defines a multivariate Gaussian distribution. As an example, let us assume that there is an unknown function, $h : [a, b] \subset \mathbb{R} \to \mathbb{R}$, that we want to model with a GP $\mathrm{f}(x)$ [1]. The jointly Gaussian random variables of $\mathrm{f}(x)$ correspond to the values that h takes over the interval $[a, b]$. These random

[1] It would make more sense to model a stochastic process with a GP rather than a well defined function, but we use a function here for simplicity.

variables are infinite, because they correspond to the continuous domain of h. The covariance function, for any $x_1, x_2 \in [a, b]$ defines the covariance between the values that the random variables take (and, for the model to be accurate, reflects a similar relationship between the values $h(x_1), h(x_2)$).

There are many types of covariance functions that can be used with GPs (see [118]), often referred to as GP kernels. We mostly work with the *squared exponential* covariance function, which in its simplest form can be written as

$$\text{cov}[\text{f}(x), \text{f}(x')] = k(x, x') = \exp\left(-\frac{1}{2\lambda}|x - x'|^2\right), \tag{2.4}$$

where λ is the lengthscale hyperparameter of the kernel. The choice of covariance function encodes some information about the type of function the GP can model. The squared exponential can in principle model any infinitely differentiable function and is usually a good first choice of kernel to try. The lengthscale hyperparameter, roughly speaking, corresponds to how *slow* the value of the function changes. A large lengthscale means that the function values change slowly (since the value of the covariance $k(x, x')$ decreases slowly as the distance between x and x' increases), while a relatively smaller lengthscale means the function values change relatively rapidly. Other types of structure, like periodicity for example, are more effectively captured with other kernels (a periodic kernel in this case) [118, 36].

2.2.1 Gaussian Process Regression

Regression is a standard, widely studied supervised learning task and represents a natural application domain for Gaussian processes.

Let us consider a dataset $\mathcal{D} := \{x_1, x_2, \ldots, x_n, y_1, , y_2, \ldots, y_n\}$ generated by $y_i = h(x_i) + \epsilon_i$, with $h : \mathbb{R} \to \mathbb{R}$, and additive, independent, identically dis-

tributed Gaussian noise $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2)$. We also define sets $\mathcal{X} = \{x_1, \ldots, x_n\}$ and $\mathcal{Y} = \{y_1, \ldots, y_n\}$, and the corresponding vectors $\boldsymbol{x} = [x_1, \ldots, x_n]^T, \boldsymbol{y} = [y_1, \ldots, y_n]^T$. The standard regression goal is using the finite dataset $\mathcal{D}$ to recover the values of the function h over its domain.

Using a Gaussian process for this purpose, and in particular a zero mean GP with a squared exponential kernel, we start by considering a set of random variables, for the values of $\mathrm{f}(x_i)$ at each point in $\boldsymbol{x}$. The covariance between any two of these is given by $k(x_i, x_j)$ and they can be summarised in a matrix $\boldsymbol{K}$, with elements $\boldsymbol{K}_{i,j} = k(x_i, x_j)$. Consider now a new random variable, corresponding to the value of f at an arbitrary point $x_\star$ in the domain of h. The covariance between $\mathrm{f}(x_\star)$ and $\mathrm{f}(x_i), i = 1, \ldots, n$, is given by $k(x_i, x_\star)$. For a finite set $\mathcal{X}_\star := \{x_1, \ldots, x_k\}$, we form the vector $\boldsymbol{x}_\star = [x_1, \ldots, x_k]^T$ and we calculate the covariance matrix $\boldsymbol{K}_{\star\star}$ where $(\boldsymbol{K}_{\star\star})_{i,j} = k(x_i, x_j)$, for the covariances between the values of f at these new points. Similarly, the covariance matrix $\boldsymbol{K}_\star$ with $(\boldsymbol{K}_\star)_{i,j} = k(x_i, x_j)$ summarises the covariances between the values of f at points in the dataset $x_i \in \mathcal{X}$ and the new points $x_j \in \mathcal{X}_\star$. Now we can consider the joint Gaussian distribution of the values of f for the points in $\boldsymbol{x}$ and $\boldsymbol{x}_\star$, which in matrix form can be written as: [2]

$$\begin{bmatrix} \mathrm{f}(\mathcal{X}) \\ \mathrm{f}(\mathcal{X}_\star) \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_\star \\ \boldsymbol{K}_\star & \boldsymbol{K}_{\star\star} \end{bmatrix}\right) \tag{2.5}$$

Note that so far we have not used the values y_i in any computation; what we have is a *prior*, and specifically a prior Gaussian *distribution*, obtained by the prior Gaussian *process* by selecting a specific set of points. The only information it contains is the choice of kernel along with its hyperparameter values.

[2] Matrices $\boldsymbol{K}, \boldsymbol{K}_\star, \boldsymbol{K}_{\star\star}$ are all symmetric and positive definite, as they are covariance matrices.

Since this is a standard Gaussian distribution we can straightforwardly sample from it. However, these samples, coming solely from the prior, do not take advantage of the data at all.

To perform the regression task, we of course need to incorporate the dataset $\mathcal{D}$ in our predictions. In probabilistic terms, we have noisy observations for a subset of the random variables that form the joint Gaussian distribution in Equation (2.7). Thus what we need to do is *condition* the random variables that correspond to the data x_i on their observed values y_i and then calculate the posterior Gaussian distribution of f at the points $x_\star \in \mathcal{X}_\star$. Denoting $f(\boldsymbol{x}_\star) = [f(x_1), \ldots, f(x_k)]^T$ for $x_i, \ldots, x_k \in \mathcal{X}_\star$, the distribution we are interested in is $p(f(\boldsymbol{x}_\star)|\boldsymbol{y})$ and from Bayes' rule we have

$$p(f(\boldsymbol{x}_\star)|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|f(\boldsymbol{x}_\star)p(f(\boldsymbol{x}_\star))}{p(\boldsymbol{y})} \tag{2.6}$$

Adjusting the prior to acknowledge the observation noise can be done by adding the noise variance on the diagonal elements of K giving the prior

$$\begin{bmatrix} \mathrm{y} \\ \mathrm{f}(\boldsymbol{x}_\star) \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I} & \boldsymbol{K}_\star \\ \boldsymbol{K}_\star & \boldsymbol{K}_{\star\star} \end{bmatrix}\right) \tag{2.7}$$

The posterior distribution of the unobserved variables can be obtained using standard properties of Gaussian distributions as

$$p(f(\boldsymbol{x}_\star)|\boldsymbol{y}) = \mathcal{N}\left(m(\mathrm{f}_\star), \mathrm{cov}\left(\mathrm{f}_\star\right)\right) \tag{2.8a}$$

$$m(\mathrm{f}_\star) = \mathbb{E}_f[f(\boldsymbol{x}_\star)] = \boldsymbol{K}_\star[\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}]^{-1}\boldsymbol{y}, \tag{2.8b}$$

$$\mathrm{cov}\left(\mathrm{f}_\star\right) = \mathbb{E}_\mathrm{f}[(\mathrm{f}(\boldsymbol{x}_\star) - m(\mathrm{f}_\star))^2] = \boldsymbol{K}_{\star\star} - \boldsymbol{K}_\star[\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}]^{-1}\boldsymbol{K}_\star. \tag{2.8c}$$

The function view of Gaussian processes

The discussion above focused on specific points, either in the observed $\boldsymbol{x}$ or some selected unobserved points forming $\boldsymbol{x}_\star$, but the real power of GPs is that these points (especially the ones in $\boldsymbol{x}_\star$) are arbitrary, and the same procedure would be valid for *any* number of points. So, instead of looking at the GP as providing a prior on the values of specific random variables, that correspond to specific function values, we can see it as providing a prior over *functions themselves* (for more details see [118]). So we can write the prior over the modelled function h:

$$p(h) = \mathcal{GP}(m(x), k(x, x')) \tag{2.9}$$

and the posterior:

$$p(h|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|h)p(h)}{p(\boldsymbol{y})} \tag{2.10}$$

For any new chosen set of points $\boldsymbol{x}_\star$, the posterior Equations (2.8) are still valid.

2.2.2 Model selection and hyperparameter adaptation for Gaussian processes

So far, we have assumed the kernel along with its hyperparameters to be prespecified. In most practical scenarios, these hyperparameters need to be adjusted to the data. Firstly, let us revisit Equation (2.4), where we gave the simplest form of a squared exponential kernel, that has a single hyperparameter, the lengthscale. A more flexible version, which can also been seen as combining an exponential kernel with a distinct white noise kernel, is given by:

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2\lambda}|x - x'|^2\right) + \sigma_\epsilon^2 \delta_{x,x'}, \tag{2.11}$$

This definition allows for two more hyperparameters, the signal variance σ_f^2 that simply scales how much the GP varies, and the noise variance σ_ϵ^2, that accounts for observational noise. Notice the $\delta_{x,x'}$, that is the Kronecker delta, that is 1 only if x and x' are the same point, and 0 otherwise. This extra noise on the diagonal actually serves a secondary purpose in addition to modelling additive measurement noise: it improves numerical conditioning of necessary algebraic operations, such as matrix inversions. We define $\boldsymbol{\theta}_m := \{\lambda, \sigma_f^2, \sigma_\epsilon^2\}$ to refer to all the hyperparameters of the model at once.

Often the term Model Selection is used to refer both to the choice of the covariance function *type* (e.g. squared exponential, linear, Matérn etc.) and the hyperparameter values [118]. We will not consider choosing between different types of covariance functions (since we are building on specific properties of the exponential kernel), but instead consider only the hyperparameter values. We start by rewriting Equation (2.10) so as to make the dependence on $\boldsymbol{\theta}_m$ explicit:

$$p(h|\boldsymbol{y}, \boldsymbol{\theta}_m) = \frac{p(\boldsymbol{y}|h, \boldsymbol{\theta}_m)p(h|\boldsymbol{\theta}_m)}{p(\boldsymbol{y}|\boldsymbol{\theta}_m)} \tag{2.12}$$

Next we focus on the term on the denominator, $p(\boldsymbol{y}|\boldsymbol{\theta}_m)$, known as the *marginal likelihood* or the *evidence*. To obtain the marginal likelihood we need to start from the likelihood of the observations $p(\boldsymbol{y}|h, \boldsymbol{\theta}_m)$ given the GP prior h, and subsequently marginalise out h:

$$p(\boldsymbol{y}|h, \boldsymbol{\theta}_m) = \prod_{i=1}^{n} \mathcal{N}(h(x_i), \sigma_\epsilon^2), \tag{2.13}$$

$$p(\boldsymbol{y}|\boldsymbol{\theta}_m) = \int p(\boldsymbol{y}, h|\boldsymbol{\theta}_m)dh = \int p(\boldsymbol{y}|h, \boldsymbol{\theta}_m)p(h|\boldsymbol{\theta}_m)dh \tag{2.14}$$

Furthermore, in a fully Bayesian approach, we would put a prior over the hyperparameters, $p(\boldsymbol{\theta}_m)$ and keep track of the posterior distribution of the hyper-

parameters

$$p(\boldsymbol{\theta}_m|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{\theta}_m)p(\boldsymbol{\theta}_m)}{p(\boldsymbol{y})} \tag{2.15}$$

This is often called level-2 inference, that is inference over the hyperparameters, after performing inference at the level of the model itself (level-1 inference) in Equation (2.14). Notice how the marginal likelihood at level 1, appears as the likelihood in level 2. The fully Bayesian approach is unfortunately not tractable in most cases [30, 118]. Instead, we obtain a point estimate $\hat{\boldsymbol{\theta}}_m$ for the hyperparameters $\boldsymbol{\theta}_m$, by maximising the marginal likelihood $p(\boldsymbol{y}|\boldsymbol{\theta}_m)$

$$\hat{\boldsymbol{\theta}}_m \in \text{argmax}_{\boldsymbol{\theta}_m} p(\boldsymbol{y}|\boldsymbol{\theta}_m) \tag{2.16}$$

This procedure is referred to as evidence maximisation or type-II maximum likelihood [89, 18]. In certain cases we do take into account an informative prior over the hyperparameters and we instead maximise $p(\boldsymbol{y}|\boldsymbol{\theta}_m)p(\boldsymbol{\theta}_m)$, a procedure which for similar reasons is called type-II maximum a posteriori probability estimation (type-II MAP). The optimisation objective is then given by:

$$\hat{\boldsymbol{\theta}}_m \in \text{argmax}_{\boldsymbol{\theta}_m} (p(\boldsymbol{y}|\boldsymbol{\theta}_m)p(\boldsymbol{\theta}_m)) \tag{2.17}$$

Both of these optimisation problems are reasonably hard, as they are non-linear and non-convex [30]. Additionally, they are naturally constrained, since the hyperparameters (lengthscales and variances) have to be positive. For practical reasons, chiefly numerical stability, we actually constrain them to be greater than some small value, usually 10^{-6} for the variances and 10^{-3} for the lengthscales. We solve this problem *locally* using an off-the-self implementation of the Limited memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS-B, [87] as implemented in the Python library Scipy [150]). L-BFGS-

B is a popular [104] optimisation algorithm in the quasi-Newton family. As such, it requires the gradients of the optimisation objective with respect to the variables, but uses an estimate of the (inverse) Hessian. From this point we will refer to this process of optimising the GP hyperparameters by maximising the evidence as *training* the GP model. Despite this problem being potentially hard, in the practical applications we are considering it is not the computational bottleneck of the algorithms we present.

2.2.3 Gaussian process regression on noisy inputs

In the previous, we showed how we can obtain the posterior distribution of the GP at any arbitrary point $x_\star$, or a number of arbitrary points we denoted $x_\star$. In certain scenarios, as we will see later, we do not have perfect knowledge of the point where the prediction needs to be made; instead the point itself is a random variable, which we assume it follows a Gaussian distribution of its own $\mathrm{x}_\star \sim \mathcal{N}(\mu_x, \sigma_x^2)$. This can be obtained by using the posterior of $\mathrm{f}(x)_\star$ and integrating out the random variable $\mathrm{x}_\star$

$$p(f(x_\star)|\boldsymbol{y}, \boldsymbol{\theta}_m) = \int p(f(x_\star)|x_\star, \boldsymbol{y}, \boldsymbol{\theta}_m) p(x_\star) dx_\star \tag{2.18}$$

Even though the term $p(f(\mathrm{x}_\star)|x_\star, \boldsymbol{y}, \boldsymbol{\theta}_m)$ is given by Equation (2.8), this integration is intractable analytically.

There are multiple ways of dealing with this issue: different approximation techniques [22, 52], estimating the posterior through Markov Chain Monte Carlo (MCMC), or using other numerical integration techniques [148]. In the next section we summarise *moment matching*, arguably the most widely used of these methods, which we extensively use in the rest of this thesis before exploring an alternate approach in Chapter 6.

Moment Matching

With moment matching we approximate the integral in Equation (2.18) with a Gaussian distribution: the integral itself is intractable, but calculating its first two moments has been shown to be [22] analytically tractable for GPs with squared exponential (or linear) kernels. By calculating the mean and variance of $p(f(\mathbf{x}_\star)|\boldsymbol{y}, \boldsymbol{\theta}_m)$ and ignoring all higher moments, we end up with a Gaussian posterior distribution for $\mathrm{f}(\mathbf{x}_\star)$. We reiterate all the components of the GP here, and we slightly expand them to account for GPs with multiple dimensions in their input. We assume:

- a dataset $\mathcal{D} = \{\boldsymbol{x}, y\}_{i=1}^n$, with every $\boldsymbol{x}_i \in \mathbb{R}^M$ and every $y_i \in \mathbb{R}$
- a SE kernel of the form $k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp(-2(\boldsymbol{x}-\boldsymbol{x}')^T \boldsymbol{\Lambda}^{-1}(\boldsymbol{x}-\boldsymbol{x}')) + \sigma_\epsilon^2 \delta_{\boldsymbol{x},\boldsymbol{x}'}$, where $\boldsymbol{\Lambda}$ is an $M \times M$ diagonal matrix, with each entry in its diagonal a lengthscale λ_i corresponding to an input dimension
- a zero mean Gaussian process using the above covariance function, where $\boldsymbol{K}$ is the covariance matrix of f at the data points $\boldsymbol{x}_i$, with $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$,
- a distribution over a point $\mathbf{x}_\star \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$.

Then the mean of the predictive posterior distribution over the noisy input $\mathbf{x}_\star$ is given by

$$\mu_{f(\boldsymbol{x}_\star)} = \mathbb{E}_{\boldsymbol{x}_\star, f}[f(\boldsymbol{x}_\star)] = \boldsymbol{\beta}^T \boldsymbol{q}, \tag{2.19}$$

with $\boldsymbol{\beta}$ an $M \times 1$ vector

$$\boldsymbol{\beta} = (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})\boldsymbol{y}, \tag{2.20}$$

and $\boldsymbol{q}$ an $N \times 1$ vector whose elements are

$$q_i = \frac{\sigma_f^2}{\sqrt{|\boldsymbol{\Sigma}_x \boldsymbol{\Lambda}^{-1} + \boldsymbol{I}|^{-0.5}}} \exp\left(-\frac{1}{2}\boldsymbol{v}_i^T (\boldsymbol{\Sigma}_x + \boldsymbol{\Lambda})^{-1} \boldsymbol{v}_i\right), \tag{2.21}$$

where

$$\boldsymbol{v}_i = (\boldsymbol{x}_i - \boldsymbol{\mu}_x). \tag{2.22}$$

Turning our attention to the variance of the predictive distribution we start from

$$\sigma^2_{f(\boldsymbol{x}_\star)} = \text{Var}_{\boldsymbol{x}_\star, f}[f(\boldsymbol{x}_\star)] = \mathbb{E}_{\boldsymbol{x}_\star, f}\left[(f(\boldsymbol{x}_\star) - \mu_{f(\boldsymbol{x}_\star)})^2\right] \tag{2.23}$$

From the law of total variance we have

$$\sigma^2_{f(\boldsymbol{x}_\star)} = \mathbb{E}_{\boldsymbol{x}_\star}\left[\text{Var}_{f|\boldsymbol{x}_\star}[f(\boldsymbol{x}_\star)]\right] + \text{Var}_{\boldsymbol{x}_\star}[\mathbb{E}_{f|\boldsymbol{x}_\star}[f(\boldsymbol{x}_\star)]], \tag{2.24}$$

where $\text{Var}_{f|\boldsymbol{x}_\star}[f(\boldsymbol{x}_\star)]$ and $\mathbb{E}_{f|\boldsymbol{x}_\star}[f(\boldsymbol{x}_\star)]$ are given by Equation (2.8) as $\text{cov}(f_\star)$ and $m(f_\star)$ respectively. After substitution of these expressions and some linear algebra (see [30] or [117] for more details) we obtain:

$$\sigma^2_{f(\boldsymbol{x}_\star)} = \sigma^2_f - \text{tr}\left(K + \sigma^2_\epsilon \boldsymbol{I})^{-1}\boldsymbol{Q}\right) + \boldsymbol{\beta}^T\boldsymbol{Q}\boldsymbol{\beta} - m^2_\star \tag{2.25}$$

The elements of the $n \times n$ matrix $\boldsymbol{Q}$ are

$$Q_{i,j} = \frac{k(\boldsymbol{x}_i, \boldsymbol{\mu}_x)k(\boldsymbol{x}_j, \boldsymbol{\mu}_x)}{|2\boldsymbol{\Sigma}_x\boldsymbol{\Lambda}^{-1} + \boldsymbol{I}|^{0.5}} \exp\left((\boldsymbol{s}_{i,j} - \boldsymbol{\mu}_x)^T(\boldsymbol{\Sigma}_x + \frac{1}{2}\boldsymbol{\Lambda})^{-1}\boldsymbol{\Sigma}_x\boldsymbol{\Lambda}^{-1}(\boldsymbol{s}_{i,j} - \boldsymbol{\mu}_x)\right) \tag{2.26}$$

with $\boldsymbol{s}_{i,j} = \frac{1}{2}(\boldsymbol{x}_i + \boldsymbol{x}_j)$.

2.3 Probabilistic Inference and Learning in Control - the PILCO framework

PILCO takes advantage of the modelling power and attractive analytical properties of GPs, in a reinforcement learning setting, in order to train successful

policies with minimal interaction between the RL agent and its environment. In this section we review how this is achieved, while in the next chapter we focus on real applications and our implementation of the PILCO framework.

2.3.1 Problem formulation

The overall goal is to design a controller for an unknown, non-linear dynamical system that achieves good performance, as indicated by a reward function over a time horizon T. We assume:

- a state space $\mathcal{X} \subset \mathbb{R}^D$,
- an input space $\mathcal{U} \subset \mathbb{R}^E$ as the set of all legal inputs,
- a dynamical system with an unknown transition function $\mathbf{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{v}_N$, where $\boldsymbol{v}_N$ is assumed to be i.i.d. Gaussian noise,
- a reward function $r : \mathcal{X} \to \mathbb{R}$,
- a normally distributed initial state distribution such that $\mathbf{x}_0 \sim p(\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0),\ \boldsymbol{x}_0 \in \mathcal{X}$.

Our task is to design a policy, $\pi^{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{U}$, with parameters $\boldsymbol{\theta}$, that maximises the expected total reward over time T. An episode is a sequence of T transitions of the dynamical system, with the initial state drawn from $p(\boldsymbol{x}_0)$. The sequence of states the system passes through during an episode, is called a *trajectory*, is $\mathbf{X_{tr}} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$.

Modelling the dynamical system

We use the GP regression framework to obtain a model of the dynamical system. We will often refer to the dynamical system as the real system, in contrast

to the model that we ourselves build.

We assume that there are available, or at least that we can obtain, a number of observed trajectories of the dynamical system, along with the inputs that produced them. In other words, for each time step, there is a starting state $\boldsymbol{x} \in \mathcal{X}$, a selected input $\boldsymbol{u} \in \mathcal{U}$, and a new state $\boldsymbol{x}' \in \mathcal{X}$. From those datapoints, we construct a dataset $\mathcal{D}$ in the appropriate form for regression as $\mathcal{D} = \{\bar{\boldsymbol{x}}_i = (\boldsymbol{x}, \boldsymbol{u}), \boldsymbol{y}_i = \boldsymbol{x}' - \boldsymbol{x}\}_{i=1}^{n}$. Thus, the inputs to the regression are pairs of state and input $\boldsymbol{x}, \boldsymbol{u}$ which we denote $\bar{\boldsymbol{x}}$, and the targets of the regression are the differences between the new state $\boldsymbol{x}'$ and the starting state $\boldsymbol{x}$. For convenience we define $\Delta \boldsymbol{x}_t = \boldsymbol{x}_t - \boldsymbol{x}_{t-1}$, for two consecutive states $\boldsymbol{x}_{t-1}, \boldsymbol{x}_t$, where subscripts $t, t-1$ denote time. For each *target* dimension (D in total) we use an independent Gaussian process, and each of these GPs takes $\boldsymbol{x}, \boldsymbol{u}$ pairs as inputs, so their dimensionality is $\mathbb{R}^{D+E} \to \mathbb{R}$. The dimensionality of the overall model, that combines the predictions of the individual GPs is $\mathbb{R}^{D+E} \to \mathbb{R}^{D}$.

We use a squared exponential kernel for each GP model, and they are all trained with evidence maximisation as outlined in Section 2.2.

2.3.2 Controllers

PILCO uses two main controller types, a linear controller and a controller based on a set of Radial Basis Functions (RBF). To incorporate constraints on the control signal, when necessary, the inputs are transformed via a sinusoidal function and are scaled appropriately. The final controllers take the form:

$$\boldsymbol{u} = \pi^{\boldsymbol{\theta}}(\boldsymbol{x}) = e \sin(\boldsymbol{W_c}\boldsymbol{x} + \boldsymbol{b_c}) \text{ and} \tag{2.27}$$

$$\boldsymbol{u} = \pi^{\boldsymbol{\theta}}(\boldsymbol{x}) = e \sin(\sum_{i=1}^{n_{\text{bf}}} (\boldsymbol{x} - \boldsymbol{g}_i)^T \boldsymbol{W}_{\boldsymbol{c},i}^{-1} (\boldsymbol{x} - \boldsymbol{g}_i)) \tag{2.28}$$

where n_{bf} is the number of basis functions for the RBF controller and e a scaling coefficient, assuming constraints of the form $-e \leq u \leq e$. For the linear controller, $\boldsymbol{\theta}$ will denote $\{\boldsymbol{W_c}, \boldsymbol{b_c}\}$, while for the RBF $\{\boldsymbol{g}_i, \boldsymbol{W}_{\boldsymbol{c},i}\}_{i=1}^{n_{\text{bf}}}$.

Notice that in both cases the analytical form of the controller allows us to a) calculate the mean and variance of the distribution of $\boldsymbol{u}$, if $\boldsymbol{x}$ is Gaussian distributed, b) calculate the gradient of $\boldsymbol{u}$ with respect to $\boldsymbol{x}$ and $\boldsymbol{\theta}$.

2.3.3 Trajectory predictions

In this section we outline the GP model can be used to obtain predictions for a whole trajectory of the system.

Starting with an estimate of $\mathbf{x}_{t-1}$, as a Gaussian distributed random variable with mean $\boldsymbol{\mu}_{t-1}$ and variance $\boldsymbol{\Sigma}_{t-1}$ our goal is to estimate the state $\mathbf{x}_t$. Firstly, using the policy $\pi^{\boldsymbol{\theta}}$ we calculate the input $\mathbf{u}_{t-1}$, as $\boldsymbol{\mu}_u$ and $\boldsymbol{\Sigma}_u$. Then $\bar{\mathbf{x}}_{t-1} = (\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ is Gaussian distributed too, $\bar{\mathbf{x}}_{t-1} \sim \mathcal{N}(\boldsymbol{\mu}_{\bar{\mathbf{x}}_{t-1}}, \boldsymbol{\Sigma}_{\bar{\mathbf{x}}_{t-1}})$.

We recognise here that the prediction we have to make requires the GP posterior predictive distribution at a normally distributed (noisy) input $\bar{\mathbf{x}}_{t-1}$. We obtain these using Equations (2.19), (2.25).

Now we have a prediction for $\Delta\mathbf{x}_t$, as $\boldsymbol{\mu}_{\Delta\mathbf{x}}$ and $\Sigma_{\Delta\mathbf{x}}$. A prediction for $p(\boldsymbol{x}_t)$ can be obtained via:

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \boldsymbol{\mu}_{\Delta\mathbf{x}} \tag{2.29}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} + \boldsymbol{\Sigma}_{\Delta\mathbf{x}} + \text{cov}[\mathbf{x}_{t-1}, \Delta\mathbf{x}_t] + \text{cov}[\Delta\mathbf{x}_t, \mathbf{x}_{t-1}] \tag{2.30}$$

$$\text{cov}[\mathbf{x}_{t-1}, \Delta\mathbf{x}_t] = \text{cov}[\mathbf{x}_{t-1}, \boldsymbol{u}_{t-1}]\boldsymbol{\Sigma}_u^{-1}\text{cov}[\boldsymbol{u}_{t-1}, \Delta\mathbf{x}_t] \tag{2.31}$$

This process can be repeated for all time steps in a trajectory, or in other words for all $t = 1, \ldots, T$.

2.3.4 Reward function

We assume a reward function of the form:

$$r(\boldsymbol{x}) = \exp\left(-(\boldsymbol{x} - \boldsymbol{x}_{\text{target}})^T \boldsymbol{\Sigma}_r^{-1} (\boldsymbol{x} - \boldsymbol{x}_{\text{target}})\right). \tag{2.32}$$

It is worth considering this assumption in more detail. Firstly, the reward function is known *a priori* which is not usually the case in Reinforcement Learning. Secondly, the reward function has the specific form outlined in Equation (2.32), which is smooth and unimodal, making many manipulations significantly easier (as discussed later). This assumption can be seen as limiting the applicability of PILCO; however, we argue that it is not necessarily as restrictive as it might appear at first glance. While it is true that in the typical RL setting, as with the MDP framework we described in Section 2.1, the reward function is unknown and can be fairly arbitrary, these requirements are often relaxed in different ways.

In many practical scenarios, a researcher may wish to train the agent to perform a task that does not come with an independent numerical reward function. The specifications of the desired agent behaviour are often loosely described in natural language, and specifying an appropriate reward function is part of the research question. Taking into account domain expertise, the nature of the agent and the algorithm the agent is trained with are all part of this process, often called *reward shaping* [102]. Furthermore, surrogate reward functions are often used, usually to encourage certain aspects of the agent's behaviour, for example, encouraging exploration throughout training [142, 14] or accelerating learning [91, 92]. In all these cases the reward function does not fit the canonical description of an arbitrary, unknown, externally defined function. The assumption that PILCO makes, while it can be restrictive, poses

the challenge of capturing the essential aspects of the task at hand in the pre-specified reward function form. In cases where the task comes with a reward function that is not given in closed form, but a qualitative understanding of what behaviour this reward function encourages is available, it is often possible to approximate the original reward function, with a function of the form specified in Equation (2.32). We present examples of this in the experimental sections of Chapters 4 and 5.

2.3.5 Policy Evaluation and Policy Improvement

Predicting the trajectory of the system for a given controller is a crucial step, but it is not our end goal: the goal is obtaining a controller that maximises the expected return. So far, for any parameter value $\boldsymbol{\theta}$, we can produce a sequence of mean and variance predictions for the states the system is going to be in the next T time steps. We use this prediction to estimate the reward that would be accumulated by implementing the policy. We hence only evaluate promising policies, and we drastically increase the data efficiency of the method.

The sequence of predictions we get from the model (Gaussian posteriors over states) form a *probabilistic trajectory*, which can be parametrised using the posterior means and covariances as:

$$\text{tr}^{\boldsymbol{\theta}} = \{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2, ..., \boldsymbol{\mu}_T, \boldsymbol{\Sigma}_T\}. \tag{2.33}$$

The expected value of the sum of rewards accumulated over an episode, known as the expected *return*, is the sum of the expected rewards received at each time step. Since every predicted $p(\boldsymbol{x}_t)$ is approximated by a Gaussian

distribution we can write the total expected reward as:

$$R^{\pi}(\theta) = \mathbb{E}_{\mathbf{X_{tr}}}\left[\sum_{t=1}^{T} r(\boldsymbol{x}_t)\right] = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}_t}[r(\boldsymbol{x}_t)], \tag{2.34}$$

where:

$$\mathbb{E}_{\mathbf{x}_t}[r(\boldsymbol{x}_t)] = \int r(\boldsymbol{x}_t)\mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)d\boldsymbol{x}_t. \tag{2.35}$$

Given the form of the reward function we use, calculation of the integral in equation (2.35) is easy to obtain in closed form [30]. This step is known as *policy evaluation* and many different RL approaches implement a version of it.

After evaluating a candidate policy, PILCO proposes a modified, improved policy for evaluation. A gradient-based optimisation algorithm is used to optimise the controller's parameters so that the expected return is maximised. In fact, the same optimiser used for training the GP model with evidence maximisation, L-BFGS-B is used here too. The gradient of the return R with respect to the parameters θ is calculated analytically using the model. In the policy gradient literature (for example [156, 141, 130]), the gradients are usually estimated stochastically. However, we do not have to resort to stochastic estimation, which is a major advantage of using (differentiable) models in general and GP models in particular.

In order to calculate the gradients of the return over the parameters, we sum the gradients of the reward at each time step over all time steps, as:

$$\frac{dR^{\pi}(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \sum_{t=1}^{T} \frac{d\mathbb{E}_{\mathbf{x}_t}[r(\boldsymbol{x}_t)]}{d\boldsymbol{\theta}}. \tag{2.36}$$

Now we consider a single time step t. We can write

$$\frac{d\mathbb{E}_{\mathbf{x}_t}[r(\boldsymbol{x}_t)]}{d\boldsymbol{\theta}} = \frac{\partial\mathbb{E}[r(\mathbf{x}_t)]}{\partial\boldsymbol{\mu}_t}\frac{d\boldsymbol{\mu}_t}{d\boldsymbol{\theta}} + \frac{\partial\mathbb{E}[r(\mathbf{x}_t)]}{\partial\boldsymbol{\Sigma}_t}\frac{d\boldsymbol{\Sigma}_t}{d\boldsymbol{\theta}} \tag{2.37}$$

The partial derivatives can be calculated from Equation (2.35) and basic identities for Gaussian distributions (the fact that the reward belongs in the exponential family of functions makes this possible). The terms $\frac{d\boldsymbol{\mu}_t}{d\boldsymbol{\theta}}$ and $\frac{d\boldsymbol{\Sigma}_t}{d\boldsymbol{\theta}}$ require us to consider how the parameters $\boldsymbol{\theta}$ influence the mean and variance of the system's state at time t. To address that we will have to use recursion back to the starting state. We here focus on the mean $\boldsymbol{\mu}$ but the analysis for the variance is similar [30]. For $\boldsymbol{\mu}_t$ we have from Equation (2.29) and (2.19):

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \boldsymbol{\mu}_{\Delta \mathbf{x}_t} = \boldsymbol{\mu}_{t-1} + \boldsymbol{\mu}_{f(\bar{\mathbf{x}}_{t-1})}, \tag{2.38}$$

and thus:

$$\frac{d\boldsymbol{\mu}_t}{d\boldsymbol{\theta}} = \frac{d\boldsymbol{\mu}_{t-1}}{d\boldsymbol{\theta}} + \frac{d\boldsymbol{\mu}_{f(\bar{\mathbf{x}}_{t-1})}}{d\boldsymbol{\theta}} \tag{2.39}$$

where $\boldsymbol{\mu}_{f(\bar{\mathbf{x}}_{t-1})}$ is obtained by applying Equation (2.19) D times, each time obtaining one (scalar) $\mu_{f(\bar{\mathbf{x}}_{t-1}),d}$, for each dimension $d = 1, \ldots, D$.

For $\frac{d\boldsymbol{\mu}_{t-1}}{d\boldsymbol{\theta}}$ we repeat the same process, but on the previous time step, thus completing the recursion.

Let us focus our attention on the dth component of the vector $\boldsymbol{\mu}_{f(\bar{\mathbf{x}}_{t-1})}$, which we denote $\mu_{f(\bar{\mathbf{x}}_{t-1}),d}$ and its derivative with respect to $\boldsymbol{\theta}$, $\frac{d\mu_{f(\bar{\mathbf{x}}_{t-1}),d}}{d\boldsymbol{\theta}}$. Equation (2.19) allows us to calculate $\mu_{f(\bar{\mathbf{x}}_{t-1}),d}$ as the mean of the Gaussian predictive posterior distribution at the noisy input $\bar{\mathbf{x}}_{t-1} \sim \mathcal{N}(\boldsymbol{\mu}_{\bar{\mathbf{x}}_{t-1}}, \boldsymbol{\Sigma}_{\bar{\mathbf{x}}_{t-1}})$. Recall that $\bar{\mathbf{x}}_{t-1} = (\mathbf{x}_{t-1}, \boldsymbol{u}_{t-1})$, with $\mathbf{x}_{t-1} \sim \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$ and $\boldsymbol{u}_{t-1} = \pi^{\boldsymbol{\theta}}(\mathbf{x}_{t-1})$. Hence, after making the appropriate substitutions, we can express $\boldsymbol{\mu}_{f(\bar{\mathbf{x}}_{t-1})}$, as a function of three variables, $\boldsymbol{\mu}_{t-1}$, $\boldsymbol{\Sigma}_{t-1}$ and $\boldsymbol{\theta}$. We thus write:

$$\frac{\mu_{f(\bar{\mathbf{x}}_{t-1}),d}}{d\boldsymbol{\theta}} = \frac{\partial \mu_{f(\bar{\mathbf{x}}_{t-1}),d}}{\partial \boldsymbol{\mu}_{t-1}} \frac{d\boldsymbol{\mu}_{t-1}}{d\boldsymbol{\theta}} + \frac{\partial \mu_{f(\bar{\mathbf{x}}_{t-1}),d}}{\partial \boldsymbol{\Sigma}_{t-1}} \frac{d\boldsymbol{\Sigma}_{t-1}}{d\boldsymbol{\theta}} + \frac{\partial \mu_{f(\bar{\mathbf{x}}_{t-1}),d}}{\partial \boldsymbol{\theta}}. \tag{2.40}$$

The partial derivatives can be directly calculated by the expression that gives $\mu_{f(\bar{\mathbf{x}}_{t-1}),d}$, in part thanks to the specific form of the controllers used. The terms $\frac{d\boldsymbol{\mu}_{t-1}}{d\boldsymbol{\theta}}$ and $\frac{d\boldsymbol{\Sigma}_{t-1}}{d\boldsymbol{\theta}}$ again lead to a recursion, as they correspond to the same terms we are calculating here, but at the previous time step. For additional details we refer the reader to [30].

This step, obtaining a new candidate policy based on some criterion and previous data, is known as *policy improvement* and is also common in numerous RL algorithms. The policy improvement step is, in computational terms, the most expensive operation of the algorithm, since it involves multiple passes through the equations for GP predictions with noisy inputs, one for each time step.

Algorithm 1 Main PILCO algorithm

```
Initialize model and policy with θ_m and θ
Interact with the system, collect data
Train GP model (updating θ_m) with evidence maximisation
repeat
    repeat
        Evaluate policy with R^π(θ)
        Update policy (updating θ) using gradient of R^π(θ)
    until Convergence or a time limit is reached
    Interact with the system, collect data
    Retrain GP model on the new data set
until task learned (or run out of time, interactions budget etc.)
```

2.3.6 Discussion

Algorithm 1 offers an overview of the high level steps PILCO follows. After initialisation, the model training proceeds. Subsequently, the model is used in the policy evaluation - policy improvement loop, both to evaluate the expected return, which is the objective function used to optimise the policy parameters, and to provide the gradients of the objective with respect to said parameters.

Once a new candidate policy is available, the controller is implemented on the original dynamical system and new data are collected. This process is repeated until either the task is successfully learnt (based on some task-specific criterion) or until a limit in the number of interactions is reached.

PILCO's main advantage is its unprecedented data-efficiency. The combination of a flexible probabilistic model, that allows for closed-form, probabilistic predictions of entire trajectories, with closed-form gradient calculations proved very successful for training controllers with very few data points. PILCO's limitations are not entirely unrelated to its strengths: while data-efficient, the policy optimisation can be very costly computationally; the GP model uses a smoothness assumption to learn fast and generalise, but that can make dealing with discontinuities or stiffness rather hard; the gradient computations (hard-coded in PILCO's implementation) can make changing any component of the algorithm time consuming.

In the Chapter 4, we introduce our own software tool that implements and expands the PILCO framework. We detail how each component of the algorithm performs separately and various test cases of the whole algorithm in action.

3

Review of relevant literature

This dissertation is focused on model based policy search methods for safe reinforcement learning. To ground our work in the wider research field, we first discuss model based policy search as a subset of RL methods, and then focus on notions of safety, as employed within RL as well as in other research communities, along with the various methods used to achieve these safety objectives.

3.1 Model-based policy search methods as part of the reinforcement learning literature

Reinforcement learning is a wide field with a rich and complex history. It unifies independently developed lines of research and it tackles a wide range of applications, each with its own priorities that call for distinct approaches despite the overarching similarities. Sutton and Barto [140] identify research on optimal control, and the related introduction of value functions and dynamic

programming as the one thread leading to modern RL research, with another being research on learning through trial-and-error, inspired by learning in animals. This taxonomy is interesting from a historical perspective, but as these threads have been successfully brought together long since [140], this alone cannot make sense of the multitude of current RL algorithms.

While there is no definitive classification for RL methods, it is helpful to consider some high-level distinctions. One such distinction is that between value function based methods and policy search methods (often the distinction is formulated as between value function based and policy gradient methods, but policy gradient methods are just the most popular class of policy search algorithms [32]). Another distinction is between model-based and model free methods, and a third between methods that assume access to perfect state information and those that take into account partial information. Another class of RL algorithms are those that combine RL techniques with deep learning methods [53]. Deep learning has achieved momentous success in numerous domains, including reinforcement learning, computer vision [79], natural language processing [147, 75] and speech recognition [8].

3.1.1 Value function and policy search methods

Policy search methods generally work with policies that are *parametrised*, and try to find values for these parameters that maximise long term return [140, 32]. Optimising the policy for maximum expected return is thus formulated as a search task in the space of the policy parameters. Value function based methods focus on learning a mapping from states (or state-action pairs) to long term return (the value function). The value function is subsequently used either for action selection directly, or to update the current policy. Still, as

the crucial task is learning an accurate value function, the core components of these algorithms work in the space of states, or the combined space of state and actions [32]. Bridging the gap between these two categories of RL methods are *actor-critic* approaches, that employ both a parametrised policy and a value function [140, 10].

Policy search methods are generally considered a better fit for applications involving physical systems [32]. They can handle stochastic policies and incorporate prior knowledge (e.g. in the form of structured policies) more naturally than their value function based counterparts [140, 32]. As an example, robotic systems typically involve continuous (and possibly high dimensional) state and action spaces which tend to be more challenging for value function based methods [32].

The policy search domain can be further broken down in different segments, depending on the type algorithm used to perform the search: gradient free methods, such as evolution strategies [62, 123] have been explored, along with optimisation schemes inspired by expectation-maximisation [77] and information theory [110]. The most common approach however is that of using the gradients of the expected return with respect to the policy parameters, leading to *policy gradient* methods [141, 73, 111]. PILCO [34] belongs to the policy gradient family, but is not a typical example: most policy gradient algorithms rely on a stochastic estimator of the gradients, while PILCO uses the learnt model of the system dynamics to analytically calculate the gradients, based on the model's predictions.

3.1.2 Model-based and model-free RL

Model-Free Reinforcement Learning (MFRL) approaches rely on sampled trajectories for learning without using an explicit model, whether they construct a value function or optimize the policy directly. In contrast, Model-Based Reinforcement Learning (MBRL) approaches use a model of the system dynamics to predict trajectories and/or future rewards, instead of (or sometimes in combination with) sampling directly from the real system. The term Model-Based Reinforcement Learning (MBRL) is somewhat ambiguous. On one hand it refers to algorithms that assume a model of the environment is given (a priori), for example as a simulator, or as a closed form description of the transition probabilities. Simultaneously, the term can refer to algorithms that *build* a model of the environment as an intermediate step in the effort of training a policy that achieves maximal return.

The difference between these three approaches can be illustrated by considering three major recent papers that attracted a lot of attention both from within and without the research community. Firstly, Deep Q-Network (DQN) [97], is a model free algorithm, combining the original Q-learning algorithm of Watkins and Dayan [154] with deep neural networks, introducing several novelties that improve and stabilise training. It showed human-level results in a large set of video games based on the Atari 2600 engine, taking the raw pixels as inputs, and thus having an extremely high dimensional state space [97]. The algorithm does not try to learn the system dynamics (a very hard task given the dimensionality of the state space), and does not have access to a simulator. AlphaGo [133], a hybrid algorithm using both reinforcement learning and Monte Carlo Tree Search (MCTS) to play the game of Go, is a model-based algorithm, in the sense that it relies on a provided simulator for

the game's transitions. Finally, MuZero [129] is an algorithm applicable to both Atari games and Go, but also Shogi and Chess, matching or surpassing the state of the art in each of these domains, without access to a simulator. Instead, MuZero, also a model-based algorithm, learns a model of each domain that allows for long term prediction. As we focus on learnt models, we use the term MBRL to associate with the latter meaning exclusively from this point forwards.

While the line between MBRL and other methods can be blurry, Dyna [139] is often considered the first seminal work representing the start of the field. As a general architecture Dyna allows learning a model, and choosing whether to sample trajectories from the model, or interacting with the environment. The model is learning the dynamics of the system, on a timestep-by-timestep basis and the potential use of various classes of models is outlined. Two variants are presented in Sutton [139], one closely related to value iteration and one based on Q-learning [154]. Both methods demonstrated how planning even with an incomplete learnt model is beneficial and can accelerate learning drastically. The notion of non-stationary environments, where the dynamics of the system are changing over time is also described, as an additional motivation for the use of a flexible model. Another early work in MBRL is Moore [99], where a very simple model keeps track of all pairs of initial and successor states encountered, and updates value estimates not uniformly but by prioritizing important and relevant updates.

MBRL has attractive qualities [153], with *data-efficiency* and *cross-task generalisation* being the most prominent. The core idea behind the data-efficiency potential of MBRL is that an agent that builds a model of the environment can use that internal model to reduce the amount of interactions with the environment necessary to achieve a certain level of performance. For cross-task

generalisation, it is assumed that when the task changes in a limited, structured way, a model-based RL algorithm might be able to leverage the model it has constructed to speed up learning, whereas a model-free agent, having only trained a task-specific policy, will have to restart learning virtually from scratch. A canonical example involves a robotic agent, which has learned to follow a specific goal trajectory as a first task. A change in the goal trajectory can be reflected as a change of the reward function in a MDP. Since the transition probabilities remain unchanged, a model-based agent can in principle leverage the model built during the first task to adapt to the second task, while there is no such straightforward way for a model-free agent to make use of the structure that is shared across the two tasks.

For Deep Reinforcement Learning (DRL) in particular, we note that early successes came mostly in the model free domain [97, 96], but model based approaches have gained significant interest more recently [155, 109, 57, 40, 72]. This push also includes approaches combining DRL with online control [157, 27] and high level planning performed with neural networks [81].

An argument against MBRL is that building an accurate, global model of the environment's dynamics can often be a lot harder than the actual RL objective of finding a good policy (one that achieves high reward) [56, 28]. This claim is of course informal, since it is unclear what 'hard' means in this context and actually formalising it is surprisingly challenging. Nonetheless, it seems to hold true in some scenarios; for example, for the widely used RL benchmarks based on Atari video games, building a model that is accurate enough to predict screen outputs, for hundreds of time steps, with any degree of accuracy, seems difficult no matter which definition of hardness we choose. However this is not necessarily the case for all settings, and in particular settings where the transition probabilities have more benign properties (e.g. smoothness). In-

terestingly, physical systems, have both a need for data-efficiency, as data collection can be costly, and their dynamics, having to follow physical laws, tend to have a degree of structure rather than being completely arbitrary, which can be exploited in building a model.

A major limiting factor in the usage of model-based methods is *model bias*: an inaccurate model favours policies that are suboptimal. When that inaccuracy is due to limited data (variance), then as more data are collected, inaccuracy reduces. If the inaccuracy is due to the model being not flexible enough however, it persists even in the limit of infinite data. Since a very flexible, but complex, model can be very hard to train, some model bias is to be expected. In fact the trade off between greater model flexibility, and thus less model bias, and the cost of training and using a complex model is an important topic on its own. More expressive models typically require more data, and are more expensive to use, on a per-prediction basis, which can be crucial in real-time settings. An interesting approach to dealing with the issue of model bias, and the one we follow, is to quantify the uncertainty of the model's predictions. This can encourage further exploration for uncertain predictions, or in general be used for a principled trade off between exploration and exploitation. To quantify the model's uncertainty one may choose to use a probabilistic model, such as Gaussian processes, which is the approach we and others [34, 25, 148] adopt. GPs also indirectly address the issue of model flexibility, as they are not based on a fixed number of parameters, but use the available data for predictions [118]. Other methods [27, 151] estimate the predictive uncertainty using ensembles of neural networks models. Another research direction is to combine model based and model free methods in a hybrid approach that ideally retains the advantages of both methods [105, 101].

3.1.3 Perfect and partial state information

So far we have assumed that the state of the system can be measured directly, perhaps in the presence of measurement noise. However, it is often the case that certain state variables cannot be measured at all, and they need to be inferred indirectly through a model that takes into account both the transitions of the unobserved states, and the generating process of the observations. In general the model takes the form:

$$x_t = f(x_{t-1}; A) + \mathrm{n}_{x,t} \tag{3.1}$$

$$y_t = g(x_t; B) + \mathrm{v}_{y,t}, \tag{3.2}$$

where x is the unobserved state, y are the observations, n, v represent noise terms and A, B represent parameters of the functions f and g. The Partially Observable Markov Decision Process (POMDP) [11, 70] is a standard framework, suitable for this setting. Surveying the whole corpus of the POMDP literature is beyond our scope, but surveys of the field are available [131, 121], and we point out that in the special case of a linear system (both f, g assumed to be linear) and Gaussian noise, an excellent review was provided by Roweis and Ghahramani [122]. It is also worth noting that dual control theory [41] follows an equivalent formulation, but early work in the two communities was largely independent [140]. Our work assumes observable states, but some of the methods developed in partially observed environments is closely related to ours, and the similarities open up promising directions for future research.

3.1.4 Reinforcement learning with Gaussian Process based models and full observability

After briefly reviewing the characteristics of the various reinforcement learning approaches, we now focus on methods close to our approach. In this section we start with PILCO [34] and discuss relevant model based RL methods that mostly use Gaussian processes, in a fully observable setting.

PILCO [34] assumes no latent states and uses a single GP model to capture the system dynamics. As previously discussed, the GP model is used with moment matching to predict the outcome of a given policy (policy evaluation). The gradients of the error along the policy parameters can then be calculated allowing the optimal policy problem to be solved as a standard gradient-based non-convex optimisation problem, using L-BFGS or a similar method (policy improvement). Once the policy is designed, it is implemented on the actual system (typically not the GP model, but potentially a simulation environment), the GP model is refined and the process is repeated until the task at hand is successfully solved.

An extension to PILCO has been proposed [33] that allows a robotic manipulator to avoid obstacles in the state space while training. The method can successfully deal with many practical challenges of the application (low cost robot manipulator, simple, off-the-self camera sensor and so on) but the most interesting element is the introduction of *obstacles in the state space*. In order for PILCO to avoid the obstacles, Gaussian-shaped penalties are incorporated (by simple addition) in the reward function. Comparing this approach with the original algorithm, we note that it takes slightly longer for the successful policy to be learned (due to the more complex reward structure), but the number of collisions while training is drastically reduced.

The PILCO framework offers a rich and carefully implemented base for modelling dynamical systems with GPs, incorporating inputs and a procedure to evaluate, improve and test policies. Some of its limitations are related to intrinsic problems with GPs, such as scalability, since, with a naive approach, the computational complexity of fitting the GP model is cubic to the number of data points. This restriction in the amount of data that can be handled favours low-dimensional state spaces over high dimensional and complex ones. Other limitations arise from the moment matching assumption made by PILCO in order to perform longer term predictions in a tractable way (we discuss this further in Chapter 6).

In another work [46] extending PILCO, some of the aforementioned limitations of the framework are addressed by replacing the GP model with a neural network. To incorporate uncertainty in this neural network approach dropout is used as an approximation of the output uncertainty [45] and particle filtering to capture the input uncertainty. The proposed method can leverage GPU architectures for training and, combined with much better asymptotic complexity (constant in the number of trials and linear to the number of input and output dimensions), shows promising results on the benchmark tests, especially for higher dimensional applications with larger datasets.

PILCO updates its model and policy offline, between episodes, based on the predicted trajectory. An alternative approach is to update the model, the policy, or both online during the episode, using the extra observations as they become available. Several works have explored this direction [74, 9, 146] combining GP-based models with Model Predictive Control (MPC). MPC is a widely used methodology originating in control theory, with the key concept of using a (usually linearised) model to plan for a number timesteps, taking a number of actions according to that plan and then updating it, based on the new state

observations. The online setting clearly increases the need for formulating optimisation problems that can be solved quickly, so typically these methods use additional approximations. In Kamthe and Deisenroth [74] for example a reformulation of the stochastic problem of optimising a control policy given a probabilistic model is employed, resulting in an equivalent deterministic problem. In the deterministic setting, MPC can be straightforwardly applied. Other works [9, 146] use sparse GP approximations [134, 83] to facilitate faster updates, obviating the naive cubic complexity associated with GP training.

3.1.5 GP state space models with partial observability

We focus next on a number of methods for the POMDP setting that are closer to our work, i.e. combining GP-based models with other techniques to improve scalability or enable online planning.

Gaussian Process Dynamical Models (GPDM) [152] use GPs for modelling dynamical systems with latent variables. GPDM uses a more elaborate scheme than PILCO to model the system dynamics, but without designing a controller, as no control input is present in this formulation. It is assumed functions f and g, from equations (3.1) and (3.2), can be expressed in terms of linear combinations of basis functions as:

$$f(x; A) = \sum_{i=1}^{N} a_i \phi_i \tag{3.3}$$

$$g(x; B) = \sum_{j=1}^{M} b_j \psi_j, \tag{3.4}$$

where ϕ and ψ are the basis functions, a, b are trainable weights, and N, M the number of basis functions used in each case.

Isotropic priors are placed on the weights $a_1..a_N$ and $b_1..b_M$ and the func-

tions f and g are modelled by a GP each, where a squared exponential (SE) kernel is used for g and a linear + SE kernel used for f. The algorithm is tested in open loop prediction and shows improved performance in comparison with baselines (e.g. [82]). Hybrid Monte Carlo (HMC) sampling is also used, and the authors suggest that initialising HMC using the mean-prediction trajectories offers a speed up on HMC convergence. In another experiment, the GPDM approach outperforms a linear dynamical system approximation, as expected.

An extension to GPDM has been proposed [76] that attempts to verify safety properties of the GPDM model. While the two GP approach of GPDM is followed, there are no latent variables in this model. The safety property is expressed in linear temporal logic (LTL) (similar to other works [58, 69]) and the problem is formulated as a probabilistic reachability objective: a requirement of reaching some "unsafe" states T with probability less than some given λ. Equivalently, this can be posed as a time invariance property, demanding that the system does not leave a safe subset of the state space with probability greater than some λ. To answer whether the system admits the safety property Hybrid Monte Carlo is used, an approach that is in essence statistical: the probability of the property being verified is estimated by the fraction of experiments where it holds. At the same time, this is part of an iterative algorithm that uses the HMC samples to refine the GP model, and samples from the new model. The assumption behind this iterative process is that convergence (of the GP model) implies that the GP is an unbiased estimator and that the HMC samples are fair.

Another approach was presented by Frigola et al. [44], where Gaussian processes are used in combination with variational Bayes and sequential Monte Carlo to learn a state-space model. The goal of learning is to provide an approximation of the posterior distribution over the possible dynamical systems,

given a very flexible prior and the available data. The use of sparse GPs allows the user to trade off model capacity with computational burden at will. Interestingly, the system is shown to be capable of learning latent models of higher dimensionality than the observation space available. Furthermore, the variational approximation allows keeping the computational complexity of making predictions independent of the length of the time series trained on. Eleftheriadis et al. [37] adopt a similar variational Bayes approach, but combine it with a recognition model using a bi-directional recurrent neural network, instead of a particle based method. Their approach can use effectively different kernel functions (smooth and non-smooth), which benefits learning, allowing the model to capture both sharp, instantaneous changes, and more consistent behaviour away from the training data. This is achievable due to the combination of shorter and longer length scales used. Experimentally the system is shown capable to correctly capture the latent dynamics, with few data points of partial observations.

Berkenkamp and Schoellig [17] use a GP to model a dynamical system, and design a controller that guarantees stability *while learning*. The dynamics are given by:

$$x_{k+1} = f(x_k, u_k) + g(x_k, u_k) \tag{3.5}$$

$$y_k = Cx_k + \omega_k \tag{3.6}$$

where function f represents an a priori known model and function g the unknown part of the system's dynamics, while ω is white noise. Function g is modelled with a GP. Assuming an operating point, the GP, *as well as its analytically calculated derivative*, which is also a GP, are used to linearise the system. Then, robust control theory is employed to create a stable controller for the linear system in question. As more data become available, the GP model is

updated. To ensure online stability during the learning procedure good priors and initial values for the hyperparameters are needed.

3.1.6 Imitation Learning

Imitation learning is a framework centered around the key idea of learning from observation, either with or without direct access to the states and actions of the demonstrator. It is often connected with RL, despite being a distinct methodology with a rich history of its own [126, 106]. Roughly speaking, imitation learning can often provide the crucial initial information that an RL algorithm needs to bootstrap learning in complex tasks where uninformed exploration would be prohibitively data intensive and time consuming [126]. It has achieved wide usage and significant practical success in robotics. DAGGER [121], standing for dataset aggregation, is a popular imitation learning algorithm, incorporating prior knowledge about the task at hand from an expert while also providing strong theoretical guarantees in a no-regret framework. Briefly, an agent collects experience with a policy and an expert annotates optimal actions for the states encountered by the agent. Then, the dataset of past state-action pairs is aggregated with the expert's suggestions and the policy is optimised on the aggregated dataset. In Innes and Ramamoorthy [67] imitation learning is combined with added specification information, expressed in linear temporal logic. This way the safety considerations are explicitly embedded in the learning process. Ideas from robust control and curriculum learning are also used to enable robotic agents to data-efficiently learn policies that respect even complex specifications.

3.2 Safety

Broadly speaking, there are diverse considerations under the term of safety, coming from different research communities. We examine some of these definitions, along with the relevant methods devised to solve this wide array of problems.

Ideas relevant to safety have been in the scope of control theory for decades. Usually the focus is stability, which in coarse terms corresponds to the ability of the controller to keep the system operating close to a set point, under certain disturbances. Robust control is more relevant to our problem, since it deals with model uncertainty (certain parameters are not exactly known, but take values within bounds) and optimising performance simultaneously. The dual estimation problem refers to the problem of estimating a good model of the system, while inferring its state at the same time (it is indeed directly related to section 3.1.5). Furthermore, H_∞ (H-infinity) theory allows optimising performance, while guaranteeing stability. Model predictive control (MPC) also optimises performance, uses a model of the system, deals naturally with constraints and is widely used in practical applications [88]. Chance constraint optimisation deals with optimising a cost function under uncertainty and constraints. Constraints are probabilistic, in the sense that they have to be respected with some predefined probability. For more on chance constraint optimisation, see Li et al. [85]. A survey of the safe reinforcement learning literature was presented by García and Fernández [48].

In the following we try to identify and group together various notions of safety, in other words the different considerations that are encapsulated in the term safety, and then describe the various strategies and techniques employed to achieve them.

3.2.1 Formalising safety

Lower bounds on the return

A central idea in RL, almost synonymous with RL itself, is learning by 'trial and error': the algorithm has to explore new policies and/or areas of the state space, in order to learn how to make optimal decisions, based on the feedback from the reward function. In the long run a successful RL algorithm will lead to increased expected return[1]. However, two important issues can be raised [48]:

- Even a policy that is optimal in terms of *expected* return can have significant variance, so that in a fraction of episodes it collects very low return (e.g. due to the stochasticity of the environment).
- During training, policies with very low performance might be explored by the agent.

In different *risk sensitive* contexts, one, or both of the above can be unacceptable. In finance, for example, the agent can lose a significant part, or the totality, of the portfolio it manages. In robotic applications the agent could be causing costly damage to the robot itself or its surroundings. A straightforward way of formulating a safety property for a reinforcement learning algorithm is thus providing a, perhaps probabilistic, lower bound on the return the algorithm achieves, either exclusively for the final policy it converges at, or also valid during training. To address such concerns several *risk aware* reinforcement learning algorithms have been proposed [137, 16, 54, 124], some of which we review in more detail in Section 3.2.2.

[1]Recall that return is defined as the sum of rewards in an episode

State space constraints

In many RL tasks the agent operates in a state space with subsets known a priori to be undesirable. These can include parts of the physical two dimensional or three dimensional space, where (stationary) obstacles exist, or operating conditions, such as temperatures and pressures that are dangerous or costly. A typical RL approach would be to assign negative reward to these states, capturing the task designer's preference for avoiding them. In practice however, this is much harder than one might initially think, both because the negative rewards need to be carefully weighted versus the possible increase in positive reward the agent would get by passing through these states to reach its eventual goals, and because, during exploration, the agent would either need to experience these negative rewards for a number of times or the negative rewards have to be given on a superset of the dangerous states, introducing extra hyperparameters and design choices. An alternative approach, mostly applicable to model based methods, is to encode the fact that certain states should not be visited in a set of constraints. Then, by using the model, the agent can evaluate whether a set of actions (or a policy) will lead to constraint violations, in a deterministic or probabilistic fashion, depending on the nature of the model. Several methods [9, 146, 33, 74, 4] adopt such an approach to encode a notion of safety, and this is the approach we follow in Chapters 5 and 6.

Safety as ergodicity

Another way to define safety is through the notion of ergodicity [145, 98]; the agent can explore states in the state space only if it is possible to reverse its actions and revert to previous states. At first glance, the connection between

ergodicity and other notions of safety can be unclear. With the appropriate formulation though, we see that by ensuring ergodicity we can guarantee other safety notions too. A robotic agent cannot destroy its physical components, assuming that it lacks the capacity to repair itself, as this change is irreversible. Similarly, in a financial context, a complete loss of an agent's portfolio is also irreversible, as in a typical scenario there is no action to recover from such a situation. In general, an ergodic policy prohibits the agent from visiting "sink states"; parts of the state space the agent cannot escape from once it enters. On the other hand, in practice it can be hard to ensure safe operation in this fashion, as the MDP framework rarely models the full spectrum of possible consequences of the agent's actions, e.g. the integrity of the physical components of a system. Still, this approach is especially attractive for non-episodic settings, where the agent's experience cannot be broken down in limited episodes, after which the agent and the environment resets, but is treated like a continuous stream.

Safety as a verifiable property

An alternative approach to similar safety notions comes from the formal verification community. The tools of formal verification can provide a robust basis for a rigorous treatment of systems' properties [13]. In fact we already mentioned such an approach [67] discussing imitation learning. Junges et al. [69] assume a known model, unknown costs/rewards and safety constraints. The safety constraints here take the form of a probabilistic reachability objective. The problem of designing a policy that minimises the expected cost and respects the safety constraint is solved iteratively. First a set of safe policies is constructed in the form of a permissive scheduler, using an SMT solver. Then, reinforcement learning is used, along with the safe permissive scheduler, to

minimise the expected cost given this scheduler (this is equivalent to picking the best policy from the set). In the next step the permissive scheduler is enriched through the SMT solver, the policy is optimised again and so on.

In another work [58] in this direction it is assumed that a set of Linear Time Invariant (LTI) systems with no process noise, parametrised by some θ, can effectively capture the system dynamics. Then probabilistic reachability properties are expressed in Linear Temporal Logic (LTL). A satisfaction function is constructed, as a mapping from the parameter space to the interval [0,1] (the probability of the system satisfying the property). *A priori* knowledge is encoded by placing priors on the parameters θ. As noisy observations are obtained, the prior is replaced by the posterior distribution in a Bayesian approach. The probability of satisfying the safety property is calculated as an expectation over the posterior of θ. An important theoretical result presented is that if the property is admitted in a polytope in the state space, the feasible values of θ form a polytope in the parameter space Θ. In conclusion, the paper outlines an algorithm that allows, given a prior and noisy observations, to estimate whether a property is admitted, along with an estimate of the uncertainty of this answer.

Another line of research also specifies the safety property using LTL, but in this case the LTL formulas are translated into Limit Buchi Deterministic Automata [59, 60]. The automata are combined with the MDP of the task, to produce a reward function that by design ensures safety, with no need for further prior knowledge, or hyperparameter tuning. The intersection of formal verification methods with RL is a very active research field exploring new directions [29] and combining formal verification with deep reinforcement learning [61, 65].

3.2.2 Promoting safety and applications

Safety with accurate uncertainty estimation for planning

An important challenge when planning with Gaussian processes is tracking uncertainty accurately, as predicting over uncertain inputs is analytically intractable, as we show in Section 2.2.3. Underestimating predictive uncertainty can lead to critical safety risks, such as policies that violate imposed constraints. PILCO uses moment matching to approximate this computation.

An interesting alternative approach is presented in [149, 148], but it is focused on stability, not safety. Two distinct cases are drawn early on: system dynamics described by the mean of a GP and system dynamics described by a full GP distribution. The authors define two variants of stability, one equivalent to Lyapunov's, for systems with deterministic dynamics (mean of a GP) and the stochastic equivalent for systems with process noise, dubbed finite time stochastic stability. The novelty of this work does not lie in the introduction of these definitions though: it lies in the way uncertainty is propagated through the system's dynamics. In PILCO, the non-Gaussian distributions that result from iterative predictions are approximated via moment matching. That makes the uncertainty bounds that are propagated along time to be dependent on the precision of this approximation. For a multimodal distribution for example, the uncertainty bounds tend to be inaccurate. In these methods [149, 148] numerical quadrature is used to approximate the analytically intractable integral that one needs to calculate to propagate the distribution over states through the dynamics for multiple time steps; the distribution is approximated by a mixture of Gaussians and the uncertainty takes explicitly into account the error in this computation through quadrature error analysis.

In related work Koller et al. [78] focus on bounding the modelling error,

that is the difference between the underlying system dynamics and the learnt GP model, with the same goal objective of planning safely with GP models. In order to compute error bounds they assume that the underlying function, describing the system dynamics, that is approximated by the GP has a bounded RKHS norm and use existing results for this setting [136].

In Chapter 6 we present a novel method that formally bounds the propagated uncertainty on the model's predictions, and explain in more detail the differences between the aforementioned methods and ours.

Safe Bayesian optimisation of policy parameters

As Bayesian optimisation provides principled uncertainty estimates, it is a natural fit for risk aware applications. Schneider [128] combines ideas from the control literature with Bayesian model approximation, using the uncertainty estimates provided to keep exploration of new policies safe. The focus of this work is on industrial applications, where data are available only in a small region of the parameter space, and while local adjustments and improvements are desirable, there are significant costs associated with aggressive exploration. The models employed provide probabilistic estimates of the performance for different values of the controller's parameters and are subsequently used to guide exploration in a conservative manner, penalising high variance. The method is tested on the classic cart-pole system, with the objective of moving the system from one set point to the next while balancing the pole, which starts from the upright position every time.

Another line of work [137, 16] incorporates constraints into the Bayesian optimisation framework, rendering parameter optimisation safer. According to the problem statement, there is an unknown reward function, $f : D \to \mathbb{R}$, where D is a finite set of decisions, for example values for some parameters

under tuning. In each round a decision a_i is made and a noisy observation $y_i = f(a_i) + n_i$ is obtained. Both studies take constraints into account, albeit formulated differently. In the first case [137] a lower bound is imposed on the algorithm's performance, i.e. $f(a_i)$ has to be greater than some h. In the second case [16] a number q of constraints are imposed, where each one is an inequality of the form $g(a_i) > 0$. Every g function is similar in essence with the reward function f: unknown, but for every episode a new noisy observation is obtained.

This formulation blurs the line between the interpretation of safety as guaranteeing performance and the one that respects state space constraints, as every g function can in principle capture the minimum distance between the system's trajectory and violating a state space constraint. On the other hand, as the method is not considering the state space at all, it is arguably closer in essence with other methods that bound the worst case reward or return. Again, cautious exploration is encouraged, without use of simulator capable of predicting trajectories. The probability of satisfying the safety property is calculated as an expectation over the posterior. The reward function, as well as the constraint function in [16] are assumed to be smooth, and using a Lipschitz continuity assumption, probabilistic guarantees are proven. Instead of assuming or determining the Lipschitz constants, in their practical implementations, both algorithms rely on GPs with a squared-exponential kernel for capturing the assumed smoothness.

Monotonic improvement for safety

Trust region policy optimisation (TRPO) [130] is a practical policy gradient algorithm, suitable for large nonlinear policies, that can solve diverse complex tasks, continuous, such as simulated robotic swimming and walking, and dis-

crete, like Atari games with raw pixels as inputs. An interesting trait of the algorithm, is that it is based on a procedure that is proven analytically to give *monotonic* improvements with every new policy suggested. It uses a total variation divergence penalty on every step, keeping the proposed policy similar to the previous one. To get the practical algorithm to be scalable though, the total variation divergence penalty is replaced with a KL-divergence bound for computational reasons. Still, TRPO is empirically shown not only to work well, but also to be able to produce monotonic improvements (after some hyperparameter tuning) in practical settings. Monotonic improvement guarantees can be connected to notions of safety; for example, as long as every unsafe policy achieves a lower reward than safe policies, starting from a safe policy, the algorithm is guaranteed to not deploy any unsafe policies.

Constrained Policy Optimisation (CPO) [4] is also a policy search algorithm closely related to TRPO. CPO guarantees not only monotonic improvement in expected return but also constraint satisfaction, throughout training. A theoretical result is presented first, expanding the results of TRPO, which bounds the difference of the expected return of two policies, given their average divergence. Assuming there are available numerical costs that quantify how close a policy is to violating each of the constraints, the algorithm guarantees, in a similar fashion with monotonic improvement, the satisfaction of the constraints. Based on these results, a practical, approximate algorithm is designed, in a scalable and computationally efficient way. Again the practical algorithm is shown empirically to respect the constraints without impeding learning in complex environments such as video games, observing only the raw pixel information.

Switching policies to ensure safety

A common practical strategy to achieve safe operation of a system is to switch to a safe policy when needed [50, 42, 86, 69]. This approach is simple conceptually, and can achieve strong empirical performance. The requirement that a safe policy exists can seem restrictive, but in practice this function can often be performed by low-level stabilising controllers, or even simple rule-based policies (e.g. "slowly back away from obstacle"). Another issue is the timely identification of the need to switch to the safe policy, and the fact that switching between policies can hamper exploration [26].

The field of robotics is a natural domain of application for safety-aware learning methods. Here the common assumption is that a reasonably accurate model of the robotic system is available, but uncertainty can still be present, either in the motion of the system [138, 68], or in the map, or model of the environment the robotic agent operates in [12]. Of particular interest to us is the work of Akametalu et al. [6] which also uses Gaussian process regression for safe learning. In their case, safety is achieved by the system switching, when necessary, to a safe control law, while in our case a single controller is combining both objectives. We also avoid solving the expensive Hamilton-Jacobi-Isaacs partial differential equation [51] that is used to initially compute the safe control law.

Safety as collision avoidance

In Calliess et al. [21] collision avoidance is considered, in a multi-agent setting, with the planning performed centrally. The trajectories of the agents, not being available explicitly, have to be predicted. The two first moments are used for this prediction, but in a very different manner to PILCO [34]: instead of

assuming that the distribution is Gaussian with the known moments, it is required that the collision is avoided for *any* distribution with these moments; thus the approach is conservative. Another interesting point comes from the way the space is partitioned to ensure no collision has too high a chance of occurring. A Lipschitz continuous function predicting collisions is constructed, such that when it takes a 0 value, the probability of a collision occurring is equal to some marginally acceptable value λ, whereas positive values are safe and negative unsafe. This allows the authors to refine the grid at each area as much as needed for the Lipschitz factor to guarantee no negative values, or for a negative value to be found (a collision is likely to occur). It also worth noting the ways collisions are resolved when identified. In the first case, agents wait at their starting position before following the designed trajectory in order to avoid colliding. Alternatively, agents are allowed to add different set-points in their plan, and conflicts are resolved through a fixed priority or an auction protocol.

Safety for transfer learning

An issue that arises when modelling physical systems is that the parameters of the system can change, due to multiple reasons such wear and tear, sensor drift etc. An instance of this problem is addressed in Held et al. [63], where a robotic system changes in time and has to be controlled with a learnt policy. The policy thus has to be updated periodically to adapt to the changing system. This situation can lead to unsuccessful and possibly catastrophic behaviour during the time it takes for the policy to adapt to the new characteristics of the system. This notion is formalised with a damage measuring function, which is used to define a safety constraint, by demanding its expected value to be lower than a threshold. The proposed algorithm assumes that when there is high

uncertainty about the system's parameters, the torques applied by the robot's motors are limited and so the expected damage can be reduced. Assuming a safe starting torque limit, the algorithm allows the system to continue training safely, increasing the limit as more data become available. The uncertainty is modelled with Gaussian distributions, and the KL divergence between policies in each iteration is used to estimate, and bound, how much the policies can change in a single step, and subsequently how much the torque limit can be increased. In that sense this work is closely related to TRPO and CPO.

Safety by human supervision

Abel et al. [3] develop a general framework called a *protocol program* that contains different ways for a human to intervene in an agent's learning without major assumptions on the agent's structure or inner workings. Curriculum learning, reward shaping and action pruning are incorporated as special cases of human-in-the-loop reinforcement learning. Subsequently, they develop some specific protocols, again as special cases of the proposed framework, in one of which the agent has to deal with a safety constraint: certain state action pairs are deemed catastrophic and the agent should never realise them. In this publication the actions are simply pruned (forbidden to the agent). In [125], the approach is extended and the agent learns to play Atari games without violating some predefined constraints, using human intervention in the starting phases of training and learning to imitate it successfully.

Table 3.1 contains a classification of the works mentioned above, along with some of the described axes. We describe the state and action spaces as continuous or discrete (C and D). The observability is described as full (F), referring to access to all state variables or partial (P). Approaches that allow for noise in the observations, but where each observations still corresponds to one state

variable, are classified as fully observable. In other words, only works that explicitly accommodate hidden/latent states are denoted with P. The model column takes the values of Free, meaning model free, Full, assuming access to real dynamics, NN, referring to modelling approaches based on neural networks (ranging from simple to more complex architectures, including variational autoencoders, recurrent neural networks etc.), GP for Gaussian processes and Other. A single paper is classified with GP, NN because it combines both of these. Safety refers to any of the safety notions described earlier (denoted only with a Yes or No), and Online refers to *online planning* specifically, and not online learning or related notions.

	States	Actions	Observability	Model	Safety	Online
Deisenroth and Rasmussen [34]	C	C	F	GP	No	No
Mnih et al. [97]	D	D	F	Free	No	No
Silver et al. [133]	D	D	F	Full	No	Yes
Schrittwieser et al. [129]	D	D	F	NN	No	Yes
Weber et al. [155]	D	D	F	NN	No	Yes
Pascanu et al. [109]	D	D	F	NN	No	Yes
Ha and Schmidhuber [57]	D	D	F	NN	No	No
Farquhar et al. [40]	D	D	F	NN	No	Yes
Kaiser et al. [72]	D	D	F	NN	No	No
Zhang et al. [157]	C	C	F	Approx.	No	Yes
Chua et al. [27]	C	C	F	NN	No	Yes
Gu et al. [56]	C	C	F	LL	No	No
Clavera et al. [28]	C	C	F	NN	No	No
Chatzilygeroudis et al. [25]	C	C	F	GP	No	No
Vinogradska et al. [148]	C	C	F	GP	No	No
Vuong and Tran [151]	C	C	F	NN	No	No
Gal et al. [46]	C	C	F	NN	No	No
McAllister and Rasmussen [94]	C	C	P	GP	No	No
Kamthe and Deisenroth [74]	C	C	F	GP	No	Yes
Andersson et al. [9]	C	C	F	GP	Yes	Yes
Van Niekerk et al. [146]	C	C	F	GP	Yes	Yes
Wang et al. [152]	C	-	P	GP	-	-
Lawrence [82]	C	-	P	GP	-	-
Frigola et al. [44]	C	-	P	GP	-	-
Eleftheriadis et al. [37]	C	-	P	GP, NN	-	-
Junges et al. [69]	D	D	F	Full	Yes	No
Sui et al. [137]	C	D	F	GP	Yes	No
Berkenkamp et al. [16]	C	C	F	GP	Yes	No
Gosavi [54]	D	D	F	NN	Yes	No
Achiam et al. [4]	C	C	F	NN	Yes	No
Turchetta et al. [145]	D	D	F	GP	Yes	Yes
Moldovan and Abbeel [98]	D	D	F	Other	Yes	Yes
Hasanbeig et al. [59]	D	D	F	Free	Yes	No
Hasanbeig et al. [60]	D	D	F	Other	Yes	No
Hasanbeig et al. [61]	C	C	F	Other	Yes	No
Hunt et al. [65]	C	C	P	Other	Yes	No
Vinogradska et al. [149]	C	C	F	GP	Yes	No
Schulman et al. [130]	C	C	F	Free	No	No
Gillula and Tomlin [50]	C	C	F	Other	Yes	Yes
Fisac et al. [42]	C	C	F	GP	Yes	No
Li et al. [86]	C	C	F	Full	Yes	No
Akametalu et al. [6]	C	C	F	GP	Yes	No
Calliess et al. [21]	C	C	F	Other	Yes	No
Saunders et al. [125]	D	D	F	Free	Yes	No

Table 3.1: Characterisation of important works in the literature, based on the properties discussed in the rest of the chapter.

4

A new software tool for PILCO

4.1 Chapter overview

Reinforcement learning (RL) is widely used to find suitable policies for a broad range of tasks. Model free methods are especially popular, but they often require a large number of interactions with the system, usually a simulator, in order to converge to an effective policy. In applications without a reliable and accurate simulator, each policy has to be evaluated on a physical system. Physical systems limit the possible evaluations, since there is a non-negligible cost associated with each evaluation, in multiple resources: time, since every trial usually lasts several seconds, normal wear-and-tear inflicted to the system etc.

This scarcity of evaluations motivates us to be more data-efficient. Model-based methods are known to often converge to a solution faster than model-free alternatives. However, models are also known to inhibit learning either by lack of flexibility or by introducing model bias, in both cases favouring solutions that are suboptimal, often persistently enough to stop the learning al-

gorithm from finding better solutions. We address the lack of flexibility by using a non-parametric model which can in principle be as flexible as needed. Model bias is also addressed, by using a model that explicitly accounts for uncertainty in its outputs. That way, even when the model's predictions are wrong, the model should provide them along with a suitably high uncertainty estimation, indicating that there is more to be learnt in the area and motivating further evaluations by the learning algorithm. Gaussian processes (GPs), our Bayesian non-parametric model of choice, fulfil both these criteria and are a suitable match for our purposes. PILCO came originally as a MATLAB open-source implementation. Our tool, termed SafePILCO, extends the original algorithm with safety constraints embedded in the training procedure, and comes as a concise and efficient Python implementation, which can benefit a wide community of researchers and practitioners that is largely moving towards this language. In this chapter we focus on highlighting the shared components of PILCO and SafePILCO, throwing light in the inner workings of the algorithm in practical scenarios, while showcasing the capabilities of our new implementation, and examples of usage [1]. In Chapter 5 we expand on this framework, incorporating safety constraints.

Ongoing discussion regarding reproducibility of results in RL [64] pinpoints a set of factors, including computational cost of large-scale experiments, algorithm sensitivity to design choices and brittleness to environmental variation, and high variance observed in performance across random seeds. As a possible mitigation, the community favours the development of open-source implementations evaluated on common benchmarks that are easy to obtain and be re-used [35, 153]. With this in mind we made our implementation freely available and as easy to use as possible.

[1] The software tool was presented as a tool demonstration paper in QEST 2020 [116]

4.1.1 Goals and design philosophy

SafePILCO is underpinned by an object-oriented architecture, enabling code re-use by keeping the implementation short and modular, with the capability to flexibly replace individual components. It takes advantage of available open source libraries, both as building blocks of the core algorithm, and as scenarios for evaluation in the case studies. It uses standard libraries to implement specific sub-tasks and facilitates extensions: libraries like GPflow give access to an array of models with a consistent interface, making new studies in this direction easier; Tensorflow provides automatic differentiation, allowing gradient-based optimisation to be employed with minimal effort; OpenAI gym provides a suite of easy to use reinforcement learning tasks, making prototyping and testing easy and consistent across algorithms. SafePILCO takes advantage of these capabilities, enabling users to employ it as a benchmark to easily compare their own methods against.

4.2 Description of the Software Tool

Our implementation of SafePILCO comes as an open source Python package, available for download [2]. To make reproduction of the experiments presented here easier we provide additional functionalities (such as logging, post processing results and creating the plots presented) in a separate repository [3]. In standard object-oriented fashion, the main components of the algorithm are organised as objects, following a hierarchy of classes. The main components of this implementation are:

[2]Main package repository: `https://github.com/nrontsis/PILCO`

[3]Experiments and figures reproduction repository: `https://github.com/kyr-pol/SafePILCO_Tool-Reproducibility`

- the environment, or scenarios, which capture the environment dynamics the agent interacts with;
- the Gaussian process model of the system dynamics providing short-term and long-term predictions;
- the policy or controller, which selects an action based on the state observation at every time step;
- the parametric reward function, which captures the performance of the algorithm.

In the following we give an overview of how these components come together to form a reinforcement learning problem.

4.2.1 Environments

Firstly, the environment the agent interacts with needs to be specified. Remembering the MDP definition in 2.1.1, the environment specifies the dimensionality of the state space $\mathcal{X} \subseteq \mathbb{R}^D$ and action space $\mathcal{A} \subseteq \mathbb{R}^E$, the transition probabilities $\mathcal{P}_a$, and, in most cases, the length of the episode T. The combination of environment and reward function defines a *task*.

The RL community has recently gravitated towards using a set of available, common environments, for several reasons. The task of defining environments that are reliable and have the right degree of difficulty can be time consuming work, which is not directly related with the research goal of designing better algorithm and/or gaining new insight. Having high quality environments available to all researchers significantly reduces that burden. Further, having common environments used widely, facilitates comparisons across algorithms and paradigms, addressing the reproducibility concerns of the community.

Several such efforts have been made, resulting in a multitude of available frameworks [20, 15, 107, 143, 144]. The OpenAI gym [20] is a popular Python library providing such a suite of reinforcement learning tasks. It provides a large number of tasks, including discrete and continuous problems, from simple toy-example tasks to complex high-dimensional ones. It includes both the popular set of Atari games that are widely used as the standard large-scale, discrete space test cases with very high dimensional observations, and the suite of continuous RL tasks that are based on the Mujoco [144] physics simulator, where the agent controls simulated robots. Figure 4.1 shows snapshots from some gym environments, using the visualisation capabilities that come with the library. In addition to the provided tasks, gym offers an easy to use interface, so new tasks and algorithms can easily be connected even if they have been developed independently. SafePILCO is designed to seamlessly interface with any environment following the OpenAI gym interface. Therefore, gym environments can be directly invoked, as well as user-defined environments equipped with the necessary functionalities.

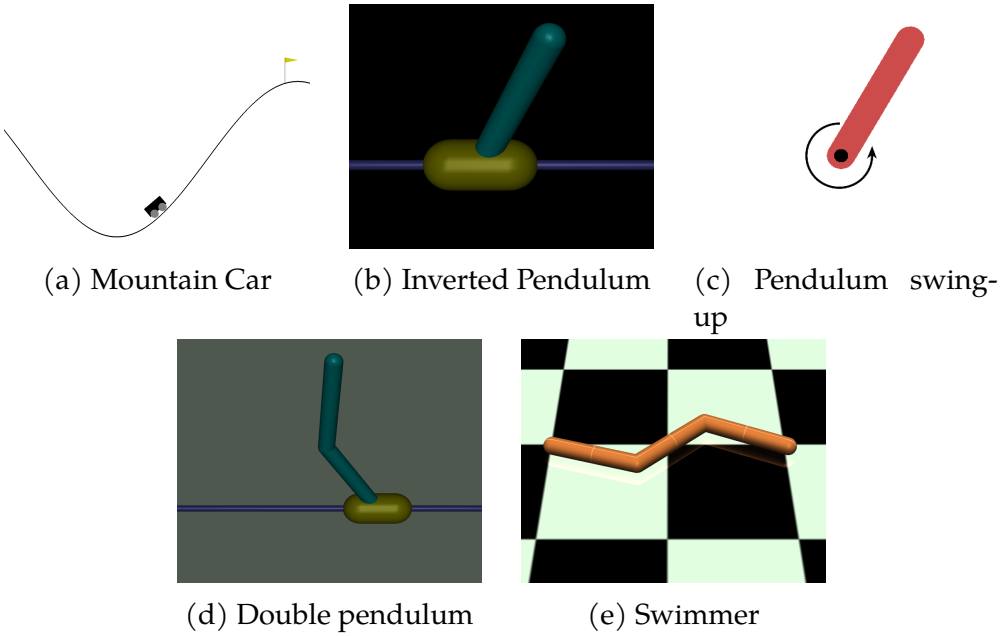
Let us introduce an example here, for illustration purposes, in parallel with the discussion of the algorithm's components. The inverted pendulum gym task (Figure 4.1b), is a variant of the classic cart-pole stabilisation task, where a pendulum is attached to a cart on a rail, and the controller applies a force to the cart. The pendulum starts close the upright position and the controller's task is to stabilise it by moving the cart to the left or to the right on the rail. The episode terminates either after a set number of timesteps (40) or, when the angle of the pendulum (with respect to the upright position), gets over a certain threshold (0.2rad). The state space has four dimensions, $\mathcal{X} \subseteq \mathbb{R}^4$, that correspond to the position and velocity of the cart, and the angle and angular velocity of the pendulum. The action space has one dimension, $\mathcal{A} \subseteq \mathbb{R}$ that is

the force applied to the cart by the controller. For every episode, an initial state is randomly sampled from the initial state distribution, which for this scenario is uniform with a range of 0.02 around the starting upright position for all four dimensions. Parenthetically, one might have noticed that throughout we have relied on Gaussian distributions for modelling our beliefs over the states and not uniform distributions. Part of the challenge of using externally defined environments is that they do not always match the assumptions of one's preferred algorithm completely, and various approximations need to be made, sometimes simple (like approximating this uniform initial distribution with a Gaussian), sometimes more involved, and of course sometimes the differences are too large to be reconciled and the algorithm is not applicable for the task. In our view this is a healthy sanity check, implicitly verifying the algorithm's versatility. As we are mostly using prespecified tasks for our experiments, we will encounter similar situations a number of times.

Moving back to the environment's interaction with the algorithm, once the initial state is sampled, the agent receives an observation and is called to choose an action. The action is provided to the environment and the environment transitions to a new state. If the conditions for terminating the episode are satisfied, the episode finishes and the sequence of states and actions from the initial state to the current state form a complete trajectory. Otherwise, the agent chooses a new action, given the state, and this process is repeated until the episode terminates.

4.2.2 Model

The model is possibly the most crucial component of the framework as a whole. It contains all the knowledge accumulated from past data, allowing us to pre-



(a) Mountain Car (b) Inverted Pendulum (c) Pendulum swing-up

(d) Double pendulum (e) Swimmer

Figure 4.1: Snapshots of various OpenAI gym environments, as visualised in OpenAI gym.

dict the system's trajectories, and thus evaluate and improve the proposed policies. As mentioned earlier, the model takes as inputs state-action pairs $\bar{\mathbf{x}} = (\mathbf{x}, \mathbf{u}) \in \mathbb{R}^{D+E}$, and its outputs are probabilistic predictions over the next state, and specifically over the difference between the current state and the next state $\Delta\mathbf{x} = \mathbf{x}' - \mathbf{x} \in \mathbb{R}^{D}$, and is thus a multi-input, multi-output model. The multi-input multi-output model is constructed by a combination of distinct multi-input, single output GP models, one for each output dimension, wrapped up in one object. The user only needs to interact with the wrapper object and not the underlying models.

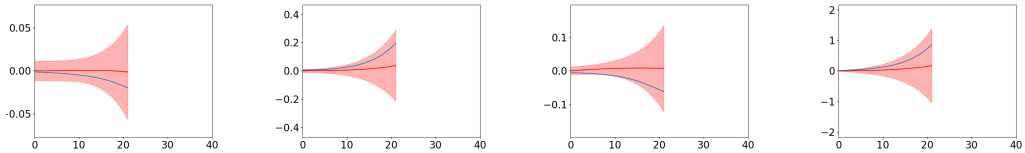
The main functionalities we require from the model is the ability to be trained, so that it fits the collected data, and to predict the next state, as a Gaussian distribution, given a Gaussian distribution over the current state and the control input.

Turning back to the inverted pendulum example, the state space is 4 dimen-

sional and the control input has 1 dimension. The overall model then takes a 5 dimensional input, and produces 4 dimensional outputs, using four distinct GP models, each one with 5 inputs and a single output. Every GP model has a squared exponential kernel which as we mentioned has 3 types of hyperparameters, the lengthscales, the signal variance and the signal noise. The lengthscales of each model form a 5×5 diagonal matrix. The overall model has thus 28 hyperparameters, that are trained with evidence maximisation using the L-BFGS-B algorithm.

In Figure 4.2 we see the model predictions for the system trajectory for two controllers. We plot the the four state variables of the inverted pendulum system, for a randomly initialised controller (a to d) and for an optimised controller (e-h). The shaded area signifies two standard deviations. The stabilising effect of the optimised controller is appropriately identified by the model, with the uncertainty bounds staying relatively close to the reference point in comparison with the random controller. Note that the episode where the random controller is used is cut short, due to the pendulum deviating from the reference point leading to the environment terminating the episode. The random controller is suitable for this comparison, to demonstrate the model's capability to discern between unstable and stable controllers. This illustration is not informative however on the quality of the trained controller used.

The model is of critical importance for the performance of the algorithm as the process of evaluating and improving candidate controllers is entirely dependent on the trajectories predicted by the model. To evaluate the model's performance, standard metrics can be used, such as the mean squared error (MSE) or the root mean squared error (RMSE). In any case, we are simultaneously facing the fundamental reinforcement learning challenge of the strong coupling between the observed data and the policies we are using or trying

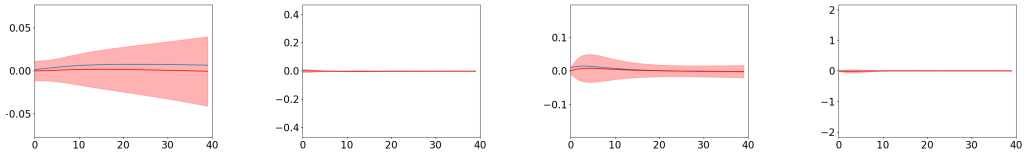


(a) Random controller, cart position

(b) Random controller, pendulum angle

(c) Random controller, cart velocity

(d) Random controller, pendulum ang. velocity



(e) Optimised controller, cart position

(f) Optimised controller, pendulum angle

(g) Optimised controller, cart velocity

(h) Optimised controller, pendulum ang. velocity

Figure 4.2: Model predictions for the system's trajectories. Top row shows predictions for a randomly initialised controller, while the bottom row for a controller that has been optimised. Each column corresponds to a different state variable for the cart-pole system.

to evaluate: even a model that is fairly accurate when evaluated on data generated from previously applied controllers, can be significantly less accurate under a new policy [140]. This issue is mitigated to an extent by the probabilistic nature of our model as the uncertainty of the predictions made is captured accurately, and by the fact that the policies proposed, evaluated and used are updated incrementally: every update to the policy is made such that the estimated performance of the policy is improving, and policies that are radically different than all policies tried so far would generate highly uncertain predicted trajectories, unlikely to achieve high return.

4.2.3 Controllers

Next, we turn our attention to the controller, or policy, the agent follows. The controller defines the manner in which the agent selects appropriate actions at each time step. Policies are implemented as memory-less, deterministic feedback controllers. The control input choice is hence directly dependent on the state the agent observes in the environment at the current time-step. The agent implements a policy π of the form $\mathbf{u} = \pi^{\boldsymbol{\theta}}(\mathbf{x})$, where $\boldsymbol{\theta}$ are the policy parameters. The package provides the two types of controllers described in Equations 2.27 and 2.28, the *linear* and the *radial-basis function* (RBF) based controller. The policies are parametric and optimising the values of these parameters, $\boldsymbol{\theta}$, constitutes the overall policy search objective.

Linear controllers have a fixed form, and do not introduce additional hyperparameters. The total number of parameters trainable parameters is $D \times E + E$. The only added design choice is the initialisation scheme for these weights, but in general initialising them normally following a zero mean Gaussian distribution with small variance has performed well, and as it is a fairly standard choice, alternative schemes were not investigated further. For RBF controllers the trainable parameters, denoted by $\boldsymbol{\theta}$ are $\{\mathbf{g}_i, \mathbf{W}_{\mathbf{c}i}\}_{i=1}^{n_{\text{bf}}}$ where n_{bf} is the number of basis functions used. There are $2 \times n_{\text{bf}} \times D \times E$ trainable parameters (the $\mathbf{W}_{\mathbf{c}}$ matrix is diagonal). The number of basis functions, n_{bf}, is a hyperparameter that is not optimised, but instead its value needs to be set in advance. Roughly speaking, a higher number of basis functions allows for more complex behaviours to be learnt, but it increases the computational demands of the algorithm and consequently its run time. We did not run any type of grid search for the number of basis functions, but rather chose sensible values, in general increasing as the (perceived) complexity of the task increases, and us-

ing roughly similar magnitudes with the original PILCO implementation.

For the inverted pendulum example used earlier both type of controllers would be applicable. The only parameter we would need to specify is the number of input and output dimensions (4 and 1 in this case).

4.2.4 Reward functions

The next point to be addressed is the choice of the reward function. We note that this is somewhat different from most of the RL literature: in SafePILCO, much like for the original PILCO [34] algorithm, the reward function is known analytically *a priori*. Having an analytic expression for the reward function is necessary for the GP model to estimate the reward of a proposed policy, without interacting with the environment. We use two types of reward functions, one with the exponential form of Equation 2.32, as used in the original PILCO implementation, and an additional linear one, given by:

$$r(\mathbf{x}) = \mathbf{w_r}^T \mathbf{x}, \tag{4.1}$$

where $\mathbf{w_r}$ is a weight vector.

We further note that it is the choice of reward function, along with the environment, that defines a task, not the environment alone. Indeed, we can design multiple tasks with a shared environment by varying the reward function. This setup is suitable for transfer learning, or for multi-task learning [31], since the same model is valid across multiple tasks.

Choosing a suitable reward function invariably requires some, at least high level, understanding of the agent's desired behaviour for a given task, and specifically, an understanding of which are the desirable system states. The two types of reward functions we have implemented, despite their simplicity,

are flexible enough to tackle a number of diverse tasks, as they represent two fundamental and complimentary requirements: the exponential form allows us to specify target states for the system to reach and stabilise at, whereas the linear reward specifies directions along which the agent should move; for example, a requirement for maximising forward movement could be encoded with a linear reward on the agent's position. For the inverted pendulum case, a reward function with an exponential form would be most suitable. The exponential would be centered at the origin, giving maximal reward at $\mathbf{0}$.

Figure 4.3 shows a possible implementation of a Python script using the software tool for the inverted pendulum task. Lines 13-19 collect an initial dataset with a random controller. In lines 22-30, the necessary components are defined. Note that the reward function is left undefined, as by default an exponential placed at the origin is used. The commented lines 29 and 30 define an alternative reward function (suitable for a different task, one where the pendulum needs to be moved and stabilised in a different position than the initial state). Lines 32-39 contain all the main algorithmic steps.

4.2.5 Libraries

The tool relies on other Python packages, allowing us to leverage their optimised functionalities and to keep the codebase succinct. Furthermore, this allows users to easily apply our algorithm to new tasks, helping with more straightforward comparison.

We use Tensorflow [90] to obtain automatic gradient computations (often referred to as auto-diff) so significantly simplifying the policy improvement step.[4] GPflow [93] is a Python package for Gaussian Process modelling

[4]By way of comparison, all gradient calculations in the Matlab implementation are hand-coded, thus extensions are laborious as any additional user-defined controller or reward func-

```python
import numpy as np
import gym
from pilco.models import PILCO
from pilco.controllers import RbfController, LinearController
from pilco.rewards import ExponentialReward
import tensorflow as tf
from gpflow import set_trainable
# from tensorflow import logging
np.random.seed(0)

from utils import rollout, policy

env = gym.make('InvertedPendulum-v2')
# Initial random rollouts to generate a dataset
X,Y, _, _ = rollout(env=env, pilco=None, random=True, timesteps=40, render=True)
for i in range(1,5):
    X_, Y_, _, _ = rollout(env=env, pilco=None, random=True,  timesteps=40, render=True)
    X = np.vstack((X, X_))
    Y = np.vstack((Y, Y_))


state_dim = Y.shape[1]
control_dim = X.shape[1] - state_dim
controller = RbfController(state_dim=state_dim, control_dim=control_dim, num_basis_functions=10)
# controller = LinearController(state_dim=state_dim, control_dim=control_dim)

pilco = PILCO((X, Y), controller=controller, horizon=40)
# Example of user provided reward function, setting a custom target state
# R = ExponentialReward(state_dim=state_dim, t=np.array([0.1,0,0,0]))
# pilco = PILCO(X, Y, controller=controller, horizon=40, reward=R)

for rollouts in range(3):
    pilco.optimize_models()
    pilco.optimize_policy()
    import pdb; pdb.set_trace()
    X_new, Y_new, _, _ = rollout(env=env, pilco=pilco, timesteps=100, render=True)
    # Update dataset
    X = np.vstack((X, X_new)); Y = np.vstack((Y, Y_new))
    pilco.mgpr.set_data((X, Y))
```

Figure 4.3: The full Python script necessary to run the algorithm for the inverted pendulum case study used as an illustrative example.

built on Tensorflow. GPflow provides a full set of basic GP functionalities, and gives access to many specialised models. Having a Tensorflow back-end, gradients in all the GPflow models are also calculated automatically. Additionally, GPflow allows the user to readily define priors and to employ different optimisers or alternative implementations of sparse approximations for GPs. Finally, our implementation is interfaced with the Open-AI gym [20], a suite of RL tasks widely used in the community. Gym tasks have consistent interfaces

tion has to include these gradient calculations too.

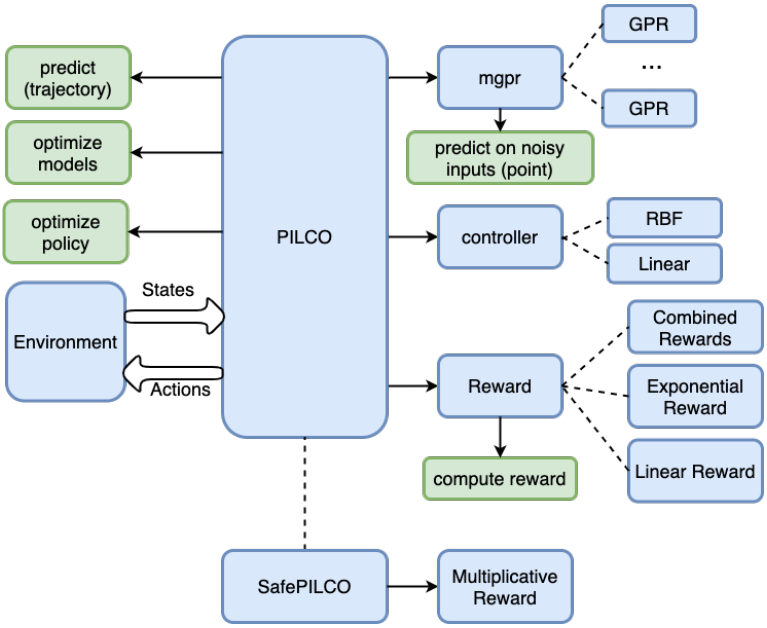


Figure 4.4: The basic structure of the SafePILCO implementation. Black arrows correspond to object-attribute relationship, dashed lines to inheritance, and wide arrows to data flow. Classes are represented by blue boxes and key functions by green boxes.

and detailed visualisation capabilities. Importantly, Open-AI gym includes a wide range of tasks of different difficulties, varying in different dimensions of complexity: dimensionality, smoothness of dynamics, length of episodes and so on. Users can quickly prototype their algorithms using easier tasks and move to more complex, time-consuming experiments as the project matures.

4.2.6 Structure

We examine the major components of the package from a software perspective, highlighting how they relate to each other and which component implements each algorithmic step. We note that this is not a complete description of every function, class, and method, but it should help in understanding the main interaction between the various components and in navigating the source code. Figure 4.4 the overall structure of the software tool.

The `PILCO` class is the central object of the package, encapsulating the GP

model, the controller, and the reward function as attributes. `PILCO` employs the model and the controller to predict a trajectory, elicits the reward function to evaluate it, and uses the gradients calculated through automatic differentiation in combination with an external optimiser, to improve the controller parameters.

The `mgpr` class implements the multi-input, multi-output Gaussian process regression that underpins the model. Specifically, `mgpr` combines several, multi-input/single-output GP models. These GP models are provided by GPflow, so avoiding the need to implement any standard GP inference and prediction from scratch. Our code provides, however, GP predictions for multiple output dimensions, when the inputs are multi-dimensional and noisy. This functionality is necessary for obtaining multi-step trajectory predictions. This methodology is implemented in `mgpr`, following the derivation in 2.2.3 and PILCO. Priors for the GP hyperparameter inference can also be included in this object. GPflow provides a number of standard statistical distributions (gamma, Laplace, beta etc.) to be used as priors, as well as numerous helpful transformations, to constrain trainable parameters (e.g. positive values only).

The `controller` class, including linear and RBF sub-classes, mainly implements action selection: for a given state, an appropriate control input is chosen. The only extra requirement from the controller is that it should be able to calculate, for a Gaussian-distributed state (as are the predicted states during the planning phase), a similarly Gaussian-distributed control input, so that the state and input are *jointly* Gaussian.

The `reward` class implements the reward function and encodes the risk of violating safety constraints. It provides, for any given state, a scalar that captures the expected reward or the constraint violation probability.

The above components are employed on all the different case studies. They

are tailored to a specific case study when combined with a particular environment. Such combination is obtained in separate files that can be run as simple scripts. We do not in general wrap this process within a function or new class, as in our experience this extra layer is not beneficial when prototyping. However, for large-scale experiments we provide such functionality as part of a separate repository, to readily reproduce all the results presented in the experimental section.

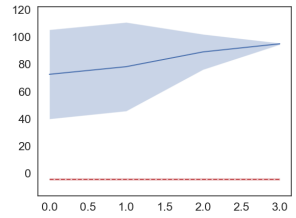
4.3 Case Studies

To evaluate the performance of the package we run a set of experiments on different tasks. The results reported are averaged over 10 random seeds (standard deviations over random seeds are reported too). To get a more accurate evaluation of the controller at each iteration, for each random seed, we test it 5 times and take the mean (the variance is not used).

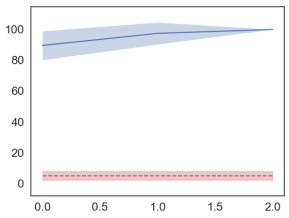
Details of the OpenAI gym tasks that are used with no modification are in [20] or on the gym website[5]. All hyperparameters used and some environment characteristics are summarised in Table 4.1. Experiments are presented in order of increasing complexity. Each task has its own characteristics, some of which can be straightforwardly compared, like dimensionality and the number of time steps per episode, while others, like the existence of local minima, or the smoothness and regularity of the dynamics, cannot be so easily evaluated and we only describe them qualitatively. Having a number of different tasks, we showcase how the algorithm can deal with these distinct challenges without relying excessively on ad hoc solutions.

A quick note on the reward functions used: as mentioned previously in

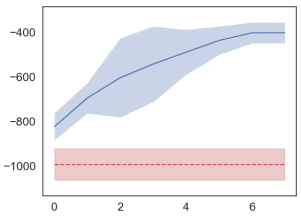
[5] `https://gym.openai.com/`



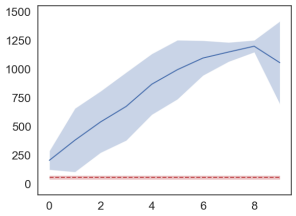
(a) Mountain Car



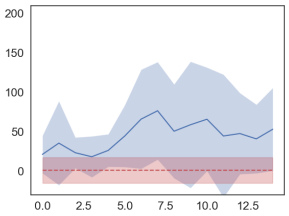
(b) Inverted Pendulum



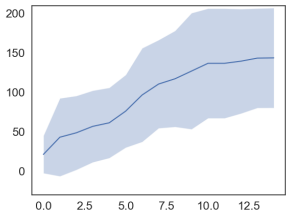
(c) Pendulum swing-up



(d) Double pendulum



(e) Swimmer, average performance



(f) Swimmer, best performance so far

Figure 4.5: Experimental results for different OpenAI gym tasks. Episode returns are on the y-axis and algorithm iteration numbers on the x-axis. We plot the mean and two standard deviations around it. The performance of a random policy is also shown (red dashed line) for comparison. For the swimmer we report both the average performance of 10 random seeds at each iteration, and the best performance so far in all previous iterations of each random seed.

Section 4.5.2, SafePILCO assumes a predetermined, closed-form reward function. Often it can be hard to come up with a reward function that accurately captures the desired behaviour of the system for a given task and that has the required structure. In all of the above examples, we evaluate our algorithm on the native reward function of the environments, despite training them with a closed-from reward function of our design. Of course one needs to incorporate some prior knowledge about the task, but in our experience and for the environments used, that was not too involved. We give specific information for each environment, its reward function and the one used by SafePILCO, along with the descriptions of the environments themselves.

Mountain Car

The mountain car experiment uses the `MountainCarContinuous-v0` gym environment. Small negative rewards are given at those step when the car has not reached the top of the hill corresponding to the goal state, and a large reward is given only once, as the agent gets to the top. The goal state is the terminal state for the environment and no further reward is obtained. This is easily captured with an exponential reward, centered at the goal state.

The mountain car task is a fairly simple problem, with low dimensionality (2 dimensional state space, and 1 dimensional control), and fairly smooth dynamics, but it still poses its own unique challenges. There is an obvious local minimum where local search algorithms can easily get stuck, trying to get scale the goal state hill directly, to greedily decrease the distance to the goal position. Additionally, the scale of the two state variables is quite different with the position taking values between -1.2 and 0.6, while the velocity is restricted between -0.07 and 0.07. This needs to be accounted for in the controller initialisation, otherwise if the initial centers of the basis functions are drawn from the same Gaussian distribution, a variance that is large enough to encourage exploration in the first dimension can lead to irrelevant values for the second dimension, and consequently to zero-valued gradients, hampering policy optimisation.

There are two mechanisms that allow our algorithm to deal with these sort of local minima. Firstly, we employ random restarts, so that one (or more) additional policy optimisation runs are executed, and the best performing solutions, as evaluated by the model is executed. Secondly, since we are optimising in the space of parameters, and not actions, and the RBF controllers employed can have a significantly higher number of parameters than a linear controller for the same task, there are less local minima to get stuck at, according to rel-

evant research [132].

To deal with the difference in magnitude of the state variables we normalise the data, in tasks that pose such issues. Note that the model also employs automatic relevance determination, that also helps with dealing with potential differences in scale between the state variables. Still we, have empirically seen that normalisation can help performance, and we hypothesise this can be due to other components than the model, such as the controller initialisation.

Cart-pole inverted pendulum

In this experiment the OpenAI gym [20] `InvertedPendulum-v2` environment is used. It is a variant of the classic cart-pole stabilisation task, where a pendulum is attached to a cart on a rail, and the controller applies a force to the cart. The pendulum starts close the upright position and the controller's task is to stabilise it by moving the cart to the left or to the right on the rail. For this scenario, as well as for the double inverted pendulum scenario (see below), the original reward function simply gives $+1$ reward for every time-step the pendulum angle is less than some threshold. Once out of this area the episode terminates as the controller cannot exert a stabilising input. An exponential reward centered at the upright position is again used.

In this scenario, as in other stabilisation tasks, planning has a distinct nature: the mean of the initial distribution coincides with the goal position. The controller's task then is not to push the predicted trajectory towards a goal state, but to reduce the variance of the predictions, while keeping their mean close to its initial value. Experimental results show (Figure 4.5) that the algorithm can successfully perform tasks of that nature too.

Pendulum swing-up

The pendulum swing-up task is based on the gym `Pendulum-v0` environment, where a pendulum, starts from a random initial position, and the task requires the controller to bring it and stabilise it in the upright position. For our own version of the task, we modify part of the default behaviour of the gym `Pendulum-v0` environment: the initial starting state distribution of the pendulum is too wide (any eligible angle with different angular velocities), which makes unimodal planning infeasible. We restrict this initial distribution to the pendulum starting close to the downward position. For this task, the environment penalises the agent with negative rewards correlated to the distance from the goal position, where the pendulum is upright. Similarly, an exponential reward is employed, centered at the goal position.

A successful trajectory for the swing-up task includes a dynamic, high velocity part, which can be hard for the model to accurately predict, followed by a stabilisation sub-task, that requires more precise control. A local minimum can also be encountered, since the controller does not have enough power to bring the pendulum upright directly, but needs to gather momentum first. Results indicate that the algorithm can successfully learn a policy for the task.

Inverted double pendulum

For the inverted double pendulum task we use the `InvertedDoublePendulum-v2` environment. This is similar to the cart pole task, except that the pendulum now consists of two links. We only apply force to the cart and have to stabilise the system to the upright positions. We add a wrapper to the default environment that is not modifying its behaviour, but changes slightly the interface, unrolling a state variable corresponding

to an angle to its sine and cosine values. This corresponds to an existing functionality from PILCO [34] (also see [30]). The task is overall very similar to the cart-pole task, with the increased dimensionality and the more complex dynamics posing a more challenging problem.

Swimmer

In the `Swimmer-v2` environment from the OpenAI gym, a robot with two joints navigates a 2-d plane by "swimming" in a viscous fluid. Both joints are controlled by an actuator each, and the system is rewarded for moving in the direction of the x-axis. This is a more challenging task, with an 8D state space, 2D control space, and nonlinear dynamics. Furthermore, the system is severely under-actuated, requiring coordination between the two controllers for the robot to start moving towards the right direction: this makes the acquisition of a reward signal at the early stages of training hard [84]. The rewards are given for distance travelled in the positive direction of the x-axis, based on the position of the root link (to the right of Figure 4.1e). This position variable however is not one of the state variables directly observed and, for this task, there is no specific goal position. Thus we define a linear reward for PILCO, based on the x-axis velocity of the agent. Also (even in the plain PILCO version) we lightly penalise extreme angle at the joints, which leads to a smoother gait, allowing the policy to generalise outside of the planning horizon, and the agent to cover longer distances.

4.4 Discussion

in this chapter we presented SafePILCO a new, flexible and extensible software tool based on the PILCO framework, for data-efficient policy search. We eval-

Variable	Tasks				
-	MountainCar	InvPend	PendSwing	DoublePend	Swimmer
State dim	2	4	3	6	8
Control dim	1	1	1	1	2
J	2	5	4	5	15
N	4	3	8	10	10
Controller Type	RBF	RBF	RBF	RBF	RBF
Basis Functions	25	5	30	40	40
SUBS	5	1	3	1	5
T	25	30	40	40	15
Σ_0	$0.1\mathbf{I}$	$0.1\mathbf{I}$	0.01diag[1,5,1]	$0.5\mathbf{I}$	$0.005\mathbf{I}$
maxiter	100	100	50	120	

Table 4.1: List of hyperparameter values employed in the experiments

uated the software's performance in a variety of standard benchmarks, and we have released a modular, extensible open source implementation for reproducibility and further use by the community.

In the following sections, 4.5 and 4.5.2, we investigate some practical consideration and we discuss restrictions associated with the current implementation of the algorithm, as well as possible extensions, including modelling the reward function and joint training in a number of predicted trajectories, and making planning more effective in highly uncertain settings (including partial observations).

In the next Chapter we explain how we extend the current algorithm and tool to incorporate constraints on the state space, which we respect with high probability both throughout training and at test time. This way, we leverage the model-based, uncertainty aware approach to reinforcement learning, to perform safe policy synthesis while maintaining high data efficiency, characteristics that are favourable for physical systems dynamics.

Notation	Description	Python Variable
-	initial random rollouts	J
N	# training episodes per run	N
-	Type of Controller	Linear or RBF
dt	sampling period	SUBS
T	time steps per episode	H
μ_0	initial state mean	m_{init}
Σ_0	initial state variance	S_{init}
maxiter	optimiser iterations	maxiter
ϵ	max tolerable risk	th

Table 4.2: List of hyperparameters - notation, meaning and source-code variable.

4.5 Practical considerations and user advice

In this section we address some more practical concerns about the use of the implementation we presented, going briefly into hyperparameter selection and outlining some of the current restrictions of the algorithm and how further work and extensions can amend them.

4.5.1 Setting hyperparameter values and troubleshooting

In this section we provide practical advice to users who wish to solve new tasks after successfully installing the package and working through the examples provided. There are several hyperparameters that need to be set in advance, but these (in general) are related to aspects of the problem at hand: early experimentation can thus help to avoid exhaustive hyperparameter searches. We organise the rest around the major components of the framework and we comment on hyperparameter settings and possible troubleshooting for each major component in turn.

We note that although the model is a crucial component of the algorithm, the associated hyperparameters do not need to be set in advance: indeed, signal variance, signal noise, and length scales can all be optimised when training

the model. In a low data-regime, however, optimisation can result in extreme values which lead to numerical instabilities. To avoid these issues we recommend:

- setting the signal noise to a fixed value (as done in the examples),
- putting priors can be used to regularise hyperparameter values, such as the Gamma priors on the lengthscale hyperparameters and signal variance,
- increasing the amount of data collected before the first run of the algorithm
- reducing the iterations of the optimisation runs.

The exponential reward function from Equation (2.32), used for most experiments has two hyperparameters, the *target*, $\mathbf{x}_{target}$ and the *weight matrix*, $\boldsymbol{\Sigma}_r^2$. The hyperparameter values are not estimated from training data, so careful prior selection is required. The target value should be the goal state to which a successful policy should drive the system. Intuitively, the weight matrix defines how quickly the reward is reduced as the distance between the current state and the target state increases. While the weight matrix can, in general, be any symmetric positive definite matrix, in all our test cases we use diagonal matrices, as we did not see a need for off-diagonal elements. For diagonal weight matrices, each value dictates how quickly the reward decays for a corresponding state variable. We recommend these weights should take reasonable values between two extremes:

- high-magnitude values make the reward decay faster and make exploration harder (the reward signal becomes sparser);

- low-magnitude values can make the reward gradient uninformative, or very small in magnitude, slowing down learning.

Finally, users inevitably face the ubiquitous RL challenge of ensuring that the reward function actually rewards behaviour that is desirable: RL agents are known to exploit the reward functions provided, an issue often referred to as *reward hacking* or *goal misalignment* [7, 39]. A strong indication that is the case, is when the agent consistently collects high reward, but its behaviour is far from what a human would consider as solving the task. In such cases, reconsidering the reward function hyperparameters (target and weight matrix) is advised.

Another issue we find can arise is that of learning being inhibited by a bad controller initialisation. While we try to resolve this issue automatically, it can still happen that the controller parameters are so chosen that the agent takes no actions whatsoever, or that the policy function ($\boldsymbol{u} = \pi^{\theta}(\boldsymbol{x})$, as introduced in Section 2.3) is close to flat in relevant regions of the state space visited, resulting in (near) zero gradients. This can be due to the environment dimensions being multiple orders of magnitude apart, in which case normalisation of the data (included in the functionality of the package) can help.

More extensive advice on troubleshooting, with examples and code, can be found in a Jupyter notebook associated to the SafePILCO package.

4.5.2 Algorithm assumptions and restrictions

We delineate here some of the restrictions of the algorithm, for two purposes: firstly, for users to easily assess whether the current version fits their application, and secondly, to outline directions for future research.

A strong assumption of the PILCO algorithm is that the environment is fully observable and Markovian. That means that the full state can be observed by the agent at every time step, and that all information relevant for predicting the next state is captured by the current state measurement and selection of the present control input. Empirically we have seen that small amounts of noise in the state measurements, despite violating this assumption, can be beneficial, improving numerical stability, but performance deteriorates as the magnitude of the noise increases (resulting in actual lack of observability). Since extensions in the direction of partial observations exist for the original PILCO algorithm [95], we expect they might as well apply to this setup, and would constitute a relevant extension of this project.

The reward function is assumed to be predefined, in closed form, so that for a Gaussian distributed state, the expectation of the reward can be efficiently calculated. As we show in the experimental section, for a range of RL tasks (not designed for our algorithm), such a reward function can be provided, without extensive hyperparameter searches or tuning. On the other hand, many tasks have complicated reward functions that cannot be easily captured in such a form. Reward shaping can mitigate this issue, however it is out of the scope of this paper. In future work, the reward function can be approximated with a Gaussian Mixture Model, which would maintain Gaussian features for noisy states. Learning the reward function from observations has been used in other model-based RL approaches [43].

Our model is based on GPs with squared exponential kernels, which are underpinned by an assumption of universal smoothness and differentiability of the system dynamics. This does not hold for all environments, e.g. whenever contact dynamics need to be modelled, or under hybrid/switching dynamics. This is a significant challenge for the kind of model we are using, and

replacing this component would require rethinking of the moment matching approximation used for multi-step planning. Work in this direction [148], replacing moment matching by numerical quadrature, can be a promising approach.

A final consideration is that the planning step is based on Gaussian-distributed predictions. This assumption can be limiting in several cases, particularly when the task at hand has high initial uncertainty, e.g. when each episode starts from an arbitrary state. Then, the unimodal, Gaussian-distributed trajectory prediction is uninformative, and PILCO often fails to estimate useful gradients of the objective function with respect to the policy parameters. Training from multiple distinct initial states can help to mitigate this issue [30].

5

Safe PILCO - Safe Policy Search Using Gaussian Process Models

5.1 Chapter Overview

Reinforcement learning, as previously discussed, is a widely applicable methodology and model based approaches to RL can support data efficient learning [1]. In many application domains a primary concern, alongside those of performance and data-efficiency, is that of *safety*. The notion of safety in this context can take many forms and the related literature is extensive and rapidly growing (see Section 3.2). In this chapter we present an extension to the PILCO framework that directly addresses safety concerns, particularly the avoidance of specific states, or sets of states, which are considered dangerous or undesirable for the system (for example, obstacles). These constraints are defined *a priori* on the state space and we require them to be respected even during training. We introduce a composite objective function that combines

[1] This chapter is based on work presented in [113].

the return (as in PILCO), with the introduced safety requirements, in a standard policy optimisation process. We make use of the learnt Gaussian process model of the system dynamics to estimate the risk of violating the constraints, before any candidate policy is implemented in the actual system. If the risk is too high, we search for safer policies by updating the objective function used for policy optimisation. Once a policy is found that is adequately safe, the agent follows it for the duration of an episode, collecting new data that are then used to update the model and the policy.

5.2 Problem Statement

Our goal is to design a controller for an unknown, non-linear dynamical system that collects maximal return, as indicated by the reward function, whilst avoiding unsafe states.

We follow the standard MDP framework, modified to incorporate state space constraints:

- a state space $\mathcal{X} \subset \mathbb{R}^n$,
- an input space $\mathcal{U} \subset \mathbb{R}^m$ as the set of all legal inputs,
- the dynamical system with a transition function $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{v}_N$, where $\mathbf{v}_N$ is assumed to be i.i.d. Gaussian noise,
- a set $\mathcal{S} \subset \mathcal{X}$ of safe states and a corresponding $\mathcal{D} = \mathcal{X} \setminus \mathcal{S}$ of unsafe (dangerous) states, where one of the two sets takes the form of an axis aligned hyper-rectangle,
- a reward function $r : \mathcal{X} \to \mathbb{R}$.

Our task is to design a policy, $\pi^{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{U}$, with parameters $\boldsymbol{\theta}$, that maximises the expected total reward over time T, while the system remains in safe parts of the state space at all times. We require the probability of the system lying in safe states to be higher than some threshold, $1 - \epsilon > 0$, with $\epsilon > 0$ being the *risk tolerance*, the highest acceptable probability of constraint violation. The sequence of states the system passes through, namely the *trajectory*, is denoted $\mathbf{X_{tr}} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$, and we require all $\mathbf{x}_i \in \mathbf{X_{tr}}$ to be safe, meaning that $\mathbf{x}_i \in \mathcal{S}$. Considering the probability associated with the trajectory as the joint probability distribution of the T random variables $\mathbf{x}_i$, p, we focus on the probability:

$$\begin{aligned} Q^{\pi}(\boldsymbol{\theta}) &= \Pr(\mathbf{x}_1 \in \mathcal{S}, \mathbf{x}_2 \in \mathcal{S}, ..., \mathbf{x}_T \in \mathcal{S}) \\ &= \int_{\mathcal{S}} ... \int_{\mathcal{S}} p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T) d\mathbf{x}_1 d\mathbf{x}_2 ... d\mathbf{x}_T, \end{aligned} \tag{5.1}$$

which is the probability of all states in the trajectory being in the safe set of states $\mathcal{S}$. As the model used is not perfect but probabilistic, and the predictions produced have infinite support as Gaussian distributions, certifying safety with perfect certainty is infeasible. Instead, we require that each policy deployed on the real system (but not the model), respects the constraints with high enough probability $Q^{\pi}(\boldsymbol{\theta}) > 1 - \epsilon$.

Our second goal is, as in Section 2.3, to maximise the performance of the system, specifically by maximising the expected return. Once more this return expectation is evaluated via the joint probability distribution over the set of the states the system passes through, using (2.34), which we rewrite here:

$$R^{\pi}(\theta) = \mathbb{E}_{\mathbf{X_{tr}}} \left[\sum_{t=1}^{T} r(\mathbf{x}_t) \right] = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}_t}[r(\mathbf{x}_t)] \tag{5.2}$$

5.3 Algorithm

5.3.1 Policy and risk evaluation

So far, for any parameter value $\boldsymbol{\theta}$, we can produce a sequence of mean and variance predictions for the states the system is going to be in the next T time steps. As in PILCO [34], we use this prediction to estimate the reward that would be accumulated by implementing the policy, and we additionally estimate the probability of violating the state space constraints. The reward function can be assumed to take either the exponential form of Equation (2.32) or the linear form of Equation (4.1). The policy evaluation step, with regard to the expected reward, remains practically unchanged from PILCO, so Equations (2.34), (2.35) continue to be valid. For convenience, we also note here that:

$$R^{\pi}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{X}_T}[\sum_{t=1}^{T} r(\mathbf{x}_t)] = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}_t \sim p(\mathbf{x}_t)}[r(\mathbf{x}_t)], \tag{5.3}$$

where:

$$\mathbb{E}_{\mathbf{x}_t}[r(\mathbf{x}_t)] = \int r(\mathbf{x}_t)\mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)d\mathbf{x}_t. \tag{5.4}$$

Similarly, for the probability of the system being in safe states during the episode, using the prediction for the trajectory:

$$\begin{aligned} Q^{\pi}(\boldsymbol{\theta}) &= Pr(\mathbf{x}_1 \in \mathcal{S}, \mathbf{x}_2 \in \mathcal{S}, ..., \mathbf{x}_T \in \mathcal{S}) = \\ &= \int_{\mathcal{S}} ... \int_{\mathcal{S}} p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T) d\mathbf{x}_1 d\mathbf{x}_2 ... d\mathbf{x}_T. \end{aligned} \tag{5.5}$$

According to the moment matching approximation we use, the distribution over states at each time step, is given by a Gaussian distribution, given the

previous state distribution, hence:

$$p(\mathbf{x}_t|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}) \approx \mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t),$$

and

$$Q^{\pi}(\boldsymbol{\theta}) \approx \int_{\mathcal{S}} ... \int_{\mathcal{S}} p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)...p(\mathbf{x}_T|\mathbf{x}_{T-1})d\mathbf{x}_1...d\mathbf{x}_T, \tag{5.6}$$

thus

$$Q^{\pi}(\boldsymbol{\theta}) \approx \prod_{t=1}^{T} \int_{\mathcal{S}} \mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)d\mathbf{x}_t = \prod_{t=1}^{T} q(\mathbf{x}_t), \tag{5.7}$$

where $q(\mathbf{x}_t)$ is the probability of the system being in the safe parts of the state at time step t.

$$q(\mathbf{x}_t) = \int_{\mathcal{S}} \mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)d\mathbf{x}_t. \tag{5.8}$$

The integral in Equation (5.8) is, in general, not available in closed form. We assume axis-aligned constraints, and that either the set of safe states $\mathcal{S}$ or the set of unsafe states $\mathcal{D}$ take the form of a hyper-rectangle, so that $\mathcal{S}$ or $\mathcal{D}$ can be written as $\{\boldsymbol{x} \in \mathcal{X} : l^i \leq x^i \leq u^i\}$, where l^i, u^i are lower and upper bounds for the i-th component of the state vector $\boldsymbol{x}$, x^i. The hyper-rectangular constraints can be a restrictive assumption. It is important however to keep the integration of equation (5.8) tractable. The integrated function is a Gaussian distribution, and the constraints correspond to the integration region. Despite the overall very convenient properties of Gaussians, this integration in high dimensional spaces is not straightforward. With the hyper-rectangular on the other hand, standard techniques [49] (and off the self libraries [150]) can provide highly accurate numerical approximations for the multivariate Gaussian integral in Equation (5.8). Further research could investigate use of other numerical in-

tegration techniques that can handle more general classes of constraints, and the associated computational cost. We should also note here that multiple, disjoint, hyper-rectangular constraints can be used in the current version of the algorithm, in order to (over) approximate constraints of different shapes.

Another important note, is that as all estimates regarding safety are combined in a single estimate, we lose some flexibility, as we cannot vary the safety requirements on a step-by-step basis, but only on the level of a whole episode. This could be restrictive, as different constraints, relevant at different timesteps may be qualitatively different. Furthermore, there might be some computational redundancy here: predicted states that are multiple standard deviations [2] away from the constraints are irrelevant from a practical perspective, but they are still considered in the computations, both of the objective function and of its gradients.

5.3.2 Policy improvement

After evaluating a candidate policy, our algorithm moves to a policy improvement step that seeks a better policy. This improvement can represent a higher probability of respecting the constraints (making the policy safer) or an increase in the expected return. As a secondary reward, we use a scaled version of the probability of respecting the constraints throughout the episode. We investigate alternatives ot this approach, and report the results in Section 5.4.1.

The objective function, capturing both safety and performance (a risk-sensitive criterion according to [48]) is defined as:

$$J^{\pi}(\boldsymbol{\theta}) = R^{\pi}(\boldsymbol{\theta}) + \xi Q^{\pi}(\boldsymbol{\theta}), \tag{5.9}$$

[2] We refer to the standard deviation of the model's predictions

where we introduce hyperparameter $\xi \in \mathbb{R}^+$. This hyperparameter controls the balance between safety and performance, and by combining the two objectives, it allows us to compare different policies based on a single numerical score.

There are alternatives to scalarising the two objectives into a single one, mainly from the field of multi-objective optimisation [120]. Another approach would be to formulate an optimisation problem that also controls the value of ξ, with a gradient ascent/descent scheme, or as a Lagrange multiplier. We opted for the most straightforward approach, of taking a weighted sum of the objectives, as is often the case in RL. We hypothesise that even a more thorough treatment would lead to relatively small gains, as there are other (unavoidable) sources of suboptimality, mainly the fact that the optimisation problem is not convex but we use a local algorithm, and the model inaccuracies that are unavoidable in the limited data setting.

The same gradient-based optimisation algorithm used for PILCO, namely L-BFGS, is used to propose a new policy. The gradient of the objective function J with respect to the parameters $\boldsymbol{\theta}$ can be calculated using the model; the calculations are similar to the calculation of the reward gradient, which we presented in Section 2.3, and can also be found in Deisenroth and Rasmussen [34] and in more detail in Deisenroth [30]. As noted previously most policy gradient algorithms use a stochastic estimator of the expected return gradient. However, in PILCO and in our case, we do not have to resort to stochastic estimation, which is a major advantage of using (differentiable) models in general and GP models in particular.

The return $R^\pi(\boldsymbol{\theta})$ accumulated over an episode, following PILCO, is a sum of the expected rewards received at each time step. In order to calculate its gradient over the parameters we need to sum the gradients over all time steps.

The same applies when penalties are used to discourage visiting unsafe states, as in [33]. In that case, instead of calculating a probability of being in an unsafe state, the system receives an (additive) penalty for getting to unsafe states. The penalty is of the same form with the reward, and its gradients, gradients of the error with respect to the parameters, are calculated the same way:

$$\frac{dR^{\pi}(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \sum_{t=1}^{T} \frac{d\mathbb{E}_{\mathbf{x_t}\sim p(\boldsymbol{x_t})}[r(\mathbf{x}_t)]}{d\boldsymbol{\theta}}. \tag{5.10}$$

We hence follow the steps outlined in Section 2.3.5 and specifically Equations (2.36-2.40).

The probability of the system remaining in safe parts of the state space, $Q^{\pi}(\boldsymbol{\theta})$, on the other hand, is the product of the probabilities of the system being safe at every time step $q(\mathbf{x}_t)$, hence:

$$\frac{dQ^{\pi}(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \sum_{t=1}^{T} \frac{dq(\boldsymbol{x}_t)}{d\boldsymbol{\theta}} \prod_{j\neq t}^{N} q(\boldsymbol{x}_j). \tag{5.11}$$

This change only affects the first step of the derivation. We need to calculate the gradients of $q(\boldsymbol{x}_t)$. This can be broken down, similarly to Equation (2.37), as:

$$\frac{dq(\boldsymbol{x}_t)}{d\boldsymbol{\theta}} = \frac{\partial q(\mathbf{x}_t)}{\partial \boldsymbol{\mu}_t}\frac{d\boldsymbol{\mu}_t}{d\boldsymbol{\theta}} + \frac{\partial q(\mathbf{x}_t)}{\partial \boldsymbol{\Sigma}_t}\frac{d\boldsymbol{\Sigma}_t}{d\boldsymbol{\theta}} \tag{5.12}$$

The terms $\frac{d\boldsymbol{\mu}_t}{d\boldsymbol{\theta}}$ and $\frac{d\boldsymbol{\Sigma}_t}{d\boldsymbol{\theta}}$ appeared already in Equation (2.37), and they are recursively calculated. The partial derivative terms, $\frac{\partial q(\boldsymbol{x}_t)}{\partial \boldsymbol{\mu}(t)}$ and $\frac{\partial q(\boldsymbol{x}_t)}{\partial \boldsymbol{\Sigma}(t)}$, for rectangular constraints, can be calculated straightforwardly using standard matrix identities [112] and the approximation used for estimating $q(\mathbf{x}_t)$ [49].

With the gradients of reward and risk in place, we can calculate the gradient of the full objective function, J, and use it in the gradient-based optimisation

algorithm.

5.3.3 Safety check and adaptively tuning ξ

When the optimisation terminates, we have a new candidate policy to be deployed on the real system. However before doing so, we need to ensure that it is safe. It is possible for an unsafe policy to be optimal in terms of J, as long as the expected reward is high enough. Here, we add a *safety check*: an additional step to the algorithm that evaluates the safety of the candidate policy. In the event that the policy is unsafe it prohibits implementation, increases ξ by a multiplicative constant and restarts the optimisation's policy evaluation-policy improvement steps. Further, we check whether the policy is too conservative: if the policy is indeed safe enough we implement it, and we also reduce ξ by a multiplicative constant, allowing for a more performance-focused optimisation in the next iteration of the algorithm.

One can interpret the resulting optimisation through the lens of Pareto optimality [103]. The original problem is finding a policy that maximises return while it respects the constraints with a probability higher than some threshold. A given model, and a given ξ correspond to a Pareto front of optimal policies. Assuming temporarily that the optimisation algorithm reaches a policy in the Pareto front, by changing the values of ξ, we aim to align that Pareto front with the set of optimal solutions of the original problem.

When the policy is implemented, we record new data from the real system. If the task is performed successfully the algorithm terminates. If not, we use the newly available data to update our model and repeat the process.

This adaptive tuning of the hyperparameter ξ guarantees that only safe policies (according to the current GP model of the system dynamics) are im-

plemented, while mitigating the need for a good initial value of ξ. Indeed, using this scheme, we have observed that starting with a high initial value of ξ, focusing on safety, leads to safer policies that, via interaction with the system, gather more data, thus leading to a more accurate model, steadily discovering high performing policies as ξ decreases. For a succinct sketch of this approach see Algorithm 2.

Figures 5.1 and 5.2 show how the algorithm takes into account safety considerations (both figures use the collision avoidance case study). In Figure 5.1 the model predicts three trajectories, for three different controllers. Since only two state variables are plotted, this projection of the state space can be plotted in 2D. The means of the model's prediction at each time step are shown as points, and ellipses around them represent the variance of each prediction. The predicted trajectory in blue comes from a model at the initial iterations of the algorithm, with less available data, leading to higher predictive uncertainty. The trajectory shown in red, illustrates a case where the controller, according to the model, is deemed unsafe, as it passes through the constrained (unsafe) area near the origin. The trajectory in green is deemed safe, as the model predicts that the system has very low probability of entering the constrained area.

Figure 5.2 focuses on the estimated risk of constraint violation, for two runs of the algorithm of 20 policy optimisation iterations each. In both cases the algorithm starts by proposing policies that are deemed unsafe for the system, as the probability for the constraints to be respected throughout the episode is too low. As ξ increases, in one of the runs the algorithm proposes a safe enough policy, and that policy is implemented (green). In the other case (red), the algorithm fails to find a safe enough policy and interaction with the system is prohibited. In this scenario the algorithm fails to proceed any further. This

is a legitimate outcome, indicating either infeasibility of the problem (given the available policies), or, more commonly, lack of training data for the model. Such a lack leads to higher variance in the model's predictions, and consequently policies that could very well be safe are evaluated as unsafe by the model.

Algorithm 2 Main SafePILCO algorithm

Initialize $\boldsymbol{\theta}, \xi$
Interact with the system, collect data
Train GP model on the data
repeat
    repeat
        Evaluate policy as $J^{\pi}(\boldsymbol{\theta}) = R^{\pi}(\boldsymbol{\theta} + \xi Q^{\pi}(\boldsymbol{\theta})$
        Update policy using gradient of $J^{\pi}(\boldsymbol{\theta})$
    until Convergence or a time limit is reached
    Calculate $Q^{\pi}(\boldsymbol{\theta})$
    if $Q^{\pi}(\boldsymbol{\theta}) > 1 - \epsilon$ **then**
        if $Q^{\pi}(\boldsymbol{\theta}) >$ upper_limit **then**
            Decrease $\xi \leftarrow 0.75\,\xi$
        Interact with the system, collect data
        Retrain GP model on the new data set
    else
        Increase $\xi \leftarrow 1.5\,\xi$
until task learned (or run out of time, interactions budget etc.)

5.4 Experiments

5.4.1 Simple Collision Avoidance

In this scenario, we consider the case of two cars approaching a junction. We assume the point of view of one of the two drivers with the objective of crossing the junction safely by accelerating or braking.

The system's state space $\mathcal{X}$ is 4 dimensional (2 position and 2 velocity vari-

ables), thus:

$$\mathbf{x}_t = \left[x_t^1, \ldots, x_t^4\right]^T . \tag{5.13}$$

The input u has one dimension, proportional to the force applied to the first car.

In this example the differentiation between safe and unsafe states is intuitive and straightforward. In order to not collide, the cars must not simultaneously be at the junction (set to be the origin (0,0)) at any point in time. This can be set as a constraint of the form:

$$|x_t^1| > a \quad \textbf{OR} \quad |x_t^3| > a \tag{5.14}$$

for the two cars positions, where a is a reasonably valued constant (10m for example). We hence denote the set of safe states as,

$$\mathcal{S} = \left\{\mathbf{x} \in \mathcal{X} : \ |x^1| > a \ \textbf{OR} \ |x^3| > a\right\} . \tag{5.15}$$

If we ignore velocities and consider the state space as a plane with the two position variables as the axes, the unsafe set of states forms a rectangle around the origin: $\mathcal{D} = \{\mathbf{x} \in \mathcal{X} : -a \leq x^1 \leq a, -a \leq x^3 \leq a\}$. The legal inputs are one dimensional corresponding to the force applied to the controlled car (car1) which we assume bounded at 2000N, accelerating or decelerating the car.

The controller used is a normalised RBF network with 20 basis functions as in [34], initialised randomly. The reward function is an exponential, (see Equation (2.32)) with the target, $\boldsymbol{x}_{\text{target}}$, chosen so that car1 is rewarded for crossing the junction (by setting the $x^1_{\text{target}} > 0$), while the weight matrix $\boldsymbol{\Sigma}_r^2$ is chosen so that the value of the reward is largely independent to other three

state variables by setting

$$(\boldsymbol{\Sigma}_r^2)^{1,1} >> (\boldsymbol{\Sigma}_r^2)^{2,2}, (\boldsymbol{\Sigma}_r^2)^{3,3}, (\boldsymbol{\Sigma}_r^2)^{4,4},$$

with $(\boldsymbol{\Sigma}_r^2)^{i,i}$ the ith component of the diagonal of $\boldsymbol{\Sigma}_r^2$. The risk tolerance, ϵ is set to 0.10.

The first approach we employ is inspired by [33], and is a variant of the standard PILCO framework, adding penalties on states that need to be avoided. The penalties are smooth, based on exponential functions, much like the rewards. We denote this as PILCOPen.

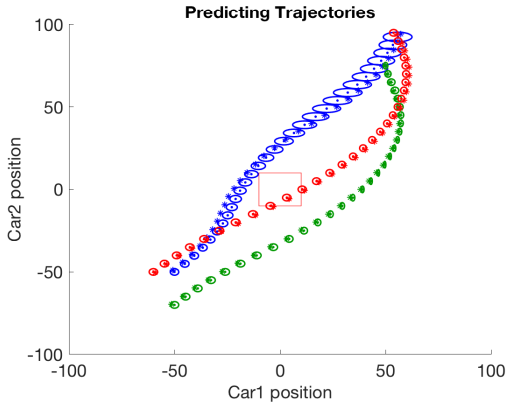


Figure 5.1: Three trajectories and associated predictions. In blue an inaccurate model captures the underlying uncertainty in a scenario where the first car accelerates and passes through the junction first. In green, an accurate model predicts well the trajectory in a scenario where the second car crosses the junction first. In red, both cars cross the junction at the same time, resulting in a collision.

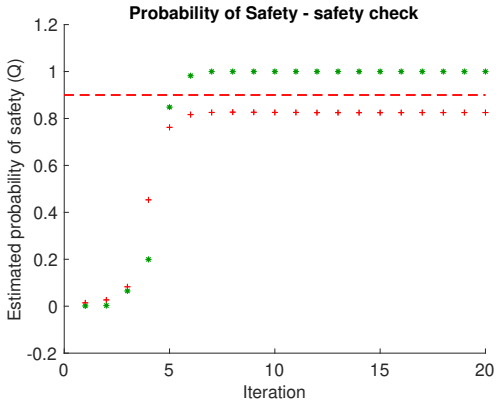


Figure 5.2: Evaluations of $Q^{\pi}(\boldsymbol{\theta})$, at step 9 of Algorithm 1, for two training runs. In the first case, in green, Q is less than the threshold $1 - \epsilon = 0.9$ for the the first 5 policies proposed. Onwards, safe enough policies are proposed by the algorithm. With red we highlight a potentially problematic scenario, where the algorithm fails to propose a safe enough policy after 20 cycles. Still collisions are avoided, since while $Q^{\pi}(\boldsymbol{\theta}) < 1 - \epsilon$ interaction with the physical system is prohibited. Experimentally, this behaviour is uncommon but possible.

	SafePILCO	PILCOPen		
ξ	-	1	10	20
collisions	30	614	32	81
av. cost	8.89	5.06	9.87	9.96
interactions	526	720	720	720

Table 5.1: Comparison between SafePILCO (our algorithm) and PILCOPen. The number collisions captures overall safety during training, while the average cost refers only to the performance component R of the objective function J (a convention we follow for the rest of the paper). Both methods are evaluated on 48 runs, with a maximum of 15 interactions with the real system per run.

Table 5.1 shows a performance comparison between our proposed method (SafePILCO) and the baseline PILCOPen. For PILCOPen, we report results for different values of the hyperparameter ξ. We note how sensitive the baseline's preformance is to the choice of ξ. Our approach however, by not allowing unsafe policies to be implemented, while optimising ξ adaptively, achieves good performance and respects the risk threshold of 0.1. Furthermore the hyperparameter initialisation does not have a significant effect on performance. We note that SafePILCO interacts with the system fewer times, due to avoiding interactions when the policy is not safe enough. In one case out of the 48 runs, no interaction took place, since the algorithm failed to propose a safe policy (see Figure 5.2).

Surrogate loss functions for the safety objective

As discussed previously 5.3.3, the probability of the system violating the constraints during an episode has a dual role:

- In the policy-evaluation policy-improvement iterations, the probability is a component of the objective function, along with the expected reward.

- In the safety check step, the probability is evaluated in order to decide whether the policy is safe enough to be implemented in the physical system.

For most problems we need to keep the probability of violating constraints under some tolerable risk threshold every time the learning algorithm interacts with the physical system. This requirement corresponds to the second case described above. In the first case, on the other hand, evaluating this probability is not a necessity so long as the policy resulting from the optimisation procedure verifies the safety requirement. We therefore explore substituting this probability, namely $Q^{\pi}(\boldsymbol{\theta})$, with a surrogate loss function which we denote $\hat{Q}^{\pi}(\boldsymbol{\theta})$. We investigate whether this substitution increases computational efficiency without impeding performance. Increased performance is expected if the surrogate creates a loss landscape that facilitates the optimisation process. Indeed, we can interpret the exponential penalties used in [33] as an example of a surrogate loss function and we evaluate its performance in the experiments that follow.

Two loss functions have been considered thus far: a scaled version of the probability itself (no surrogate in this case) and exponential penalties. For completeness, we consider two more: an additive cost function based on the sum of the probabilities of violating the constraints at each time step (ProbAdd for short) and one based on a logarithmic transform of the original (multiplicative) probability (LogProb).

The additive cost based on the risk of constraint violation (ProbAdd) is defined as:

$$\hat{Q}^{\pi}(\boldsymbol{\theta})_{\text{add}} = \sum_{t=1}^{T} q(\mathbf{x}_t), \tag{5.16}$$

with $q(\mathbf{x}_t)$ as defined in Equation (5.8).

Assuming perfect safety is feasible, with all $q(x_t) = 1$, maximising this surrogate loss function leads to a maximisation of the original objective, which is the product of $q(x_t)$ for all t. When this is not feasible on the other hand, the maximum is not necessarily the same. Keeping in mind that convergence to the global maximum is not guaranteed for any of the methods used here for policy optimisation, the effectiveness of the surrogate cost function is evaluated empirically.

Taking the logarithm of the original multiplicative probability allows us to create an additive cost function, while maintaining the same maximum, since the logarithm is a concave function, defined as:

$$\hat{Q}^{\pi}(\boldsymbol{\theta})_{\log} = \log \prod_{t=1}^{T} q(\mathbf{x}_t) = \sum_{t=1}^{T} \log(q(\mathbf{x}_t)). \tag{5.17}$$

Differentiating the above is not significantly different than the process described previously and the gradients are used in the same optimisation algorithm.

We present results in Table 5.2. The four cost functions are evaluated over 32 runs, with 15 maximum allowed interactions per run with the physical system (480 interactions allowed in total). We can see that all methods achieve a fairly low number of collisions (much lower than the maximum allowed risk of 10% per interaction), and the Prob and Log surrogate cost functions achieve slightly better performance. The Log and the ProbAdd cost functions fail to propose a controller for a non-negligible number of cases (4 and 6 out of 32), a fact suggesting that the optimisation proved harder using these cost functions. For the rest of this thesis the Prob cost function is used (and thus denoted simply SafePILCO, except when explicitly stated otherwise), since it achieves good performance and is in general a simpler approach, in the sense that objective

	Prob (**SafePILCO**)	Exp	Log	ProbAdd
collisions	12	8	4	11
av. cost	8.75	9.19	8.78	9.28
interactions	327	412	166	315
unsolved	0	0	4	6

Table 5.2: Surrogate cost function comparison. 'Prob' denotes the probability of collision used as a multiplicative cost, 'Exp' smooth exponential penalties, 'Log' the log of the probability of collision, and 'ProbAdd' the sum of the probabilities of collision at each time step (additive cost).

(low risk) and cost function match, other than the multiplicative constant (ξ).

Hyperparameter tuning

Here we examine the effects of the method we introduce for tuning the hyperparameter ξ. Using the same objective function, we change only the way ξ is set and tuned, isolating and evaluating its effects on performance. The simplest version we evaluate uses a fixed value for ξ, as in [33] and the PILCOPen algorithm we employed in Section 5.4.1. We compare the latter with a version of our algorithm that checks whether the policy is safe enough, increasing ξ if not (but never decreasing it). Finally, we compare with a version of the algorithm that adapts ξ by increasing or decreasing its value accordingly (SafePILCO in Section 5.4.1). To provide a fairer comparison we use the same surrogate loss in all cases, namely that of exponential penalties on the unsafe parts of state space. Since the dynamics of the system are simple and linear, in this case an optimal controller could be designed with standard methods, and a comparison with the average cost our methods incurred would be informative.

We see in Table 5.3 that an adaptively tuned ξ leads to similar, or better, performance and more robustness to the initial choice of ξ.

	PILCOPen		PenCheck	SafePILCO	
(starting) ξ	5	10	5	5	10
collisions	36	0	0	1	2
av. cost	7.54	13.31	14.45	14.51	12.66
interactions	240	240	138	97	138

Table 5.3: Different strategies of setting ξ and their effects for different initial values. PILCOPen picks one value for ξ and implements all policies that are proposed by the policy optimisation algorithm, PenCheck uses the safety check before implementing a policy and increases ξ if the policy is unsafe, and SafePILCO, increases and decreases the hyperparameter adaptively.

5.4.2 Building Automation Systems

We here apply our approach to a problem in the domain of *building automation systems*, often referred to as smart buildings. The usual aim here is to efficiently use the air conditioning system, keeping the temperature and possibly CO_2 emission levels within given limits, or close to specified values. These values often correspond to comfortable conditions for occupants, but can vary significantly depending on the building's purpose. As the number of available sensors and their connectivity increase, so does the sophistication of the controlling mechanisms of these systems. Increased emphasis on energy consumption, with its associated financial and environmental cost, and of course the desire to improve the experience of those in the buildings, contribute to the rising interest in the area [24]. Finally, it's worth noting that one of the most well-known real-world applications of reinforcement learning was in this domain, with Deepmind offering an RL optimised system for cooling Google's data centres [47].

For our experiment we use as ground truth an open source simulator [3] that

[3]Code for the simulator can be found at `https://gitlab.com/natchi92/BASBenchmarks`.

models air conditions, released as part of a set of benchmarks [24, 2] for the verification of stochastic systems. Following the same approach outlined in Section 5.4.1, we treat the simulator as a black box: our algorithm sees the data generated, as sequences of states and actions, but has no access to the simulator's internal parameters. With the data collected, we train a GP-based dynamics model, and follow Algorithm 2.

In this setting, matching Case Study 2 from Cauchi and Abate [24], we control the temperature in two adjacent rooms. The cost we minimise is the quadratic error between the temperatures in the two rooms and the reference temperature. The total number of state variables is 7 (including room temperatures, wall temperatures etc.) while the control input we have at our disposal is one-dimensional and corresponds to the (common) air supply temperature for the two rooms. The measurements have a sampling period of 15 minutes. We collect 72 hours worth of data to start training, and use a simple linear controller. The initial temperature in room 1 is 15°C and the target temperature is 20°C, while room 2 starts with the target temperature. The task is to gradually increase temperature in room 1, while keeping the temperature in room 2 below 20.5°C. Since the air supply for the two rooms is shared, aggressive temperature control for room 1 would lead to the temperature in room 2 overshooting both the target of 20°C, and the constraint of 20.5°C, a hypothesis we verify by running plain PILCO.

In Table 5.4, we summarise the results obtained by plain PILCO, optimising performance exclusively (without regard to constraints), PILCOPen, using an exponential penalty of preset weight to encourage safety, and SafePILCO, using the adaptively weighted penalty and the safety check we introduced. All algorithms interact with the system for 5 episodes of 12 hours each. The results are averaged over 10 random seeds. Please note that in this case we

	PILCO	SafePILCO (Add)	PILCOPen ($\xi = 2$)
Con. Viol. (steps)	26.1 ± 8.7	0.0 ± 0.0	0.0 ± 0.0
Con. Viol. (epis.)	4.5 ± 1.5	0.0 ± 0.0	0.0 ± 0.0
RMSE	0.94 ± 0.32	0.95 ± 0.31	1.24 ± 0.41

Table 5.4: Results for the BAS environment. Regarding constraint violations, we report the number of time steps where constraints were violated during the 5 episodes of interaction in Con. Viol (steps), as well the number of unique episodes with at least one constraint violation in Con. Viol (epis.). The RMSE refers to the temperature difference of the two rooms from the target value and is used as an interpretable cost. Results are averaged over 10 random seeds, with one standard deviation shown.

report *cost*, and specifically root mean squared error. Again we operate under the assumption that the task has a specified reward function independent of PILCO, and we achieve similar behavior using one of the two reward functions implemented in the software tool. As expected, plain PILCO achieves the best score of all three methods, but also results in the highest number of constraint violations. We report two metrics with respect to constraint violations: the total number of timesteps the constraints were violated for, and the number of episodes where the constraints were violated for at least one timestep. We can see in 5.4 that SafePILCO manages to respect the constraints without incurring practically any extra cost, while PILCOPen (using a hand-crafted reward, whose shape and weight were tuned to good values to the best of our knowledge) respects constraints but with significant additional cost. Plain PILCO is included to show the performance of unconstrained policies, in terms of both cost and constraint violations. In Figure 5.3 we see a typical response of the two-room system (as predicted by the simulator) to the controller trained with the proposed algorithm (SafePILCO). Note that for this system too, designing an optimal controller that respects the constraints is possible with standard techniques, and potentially informative.

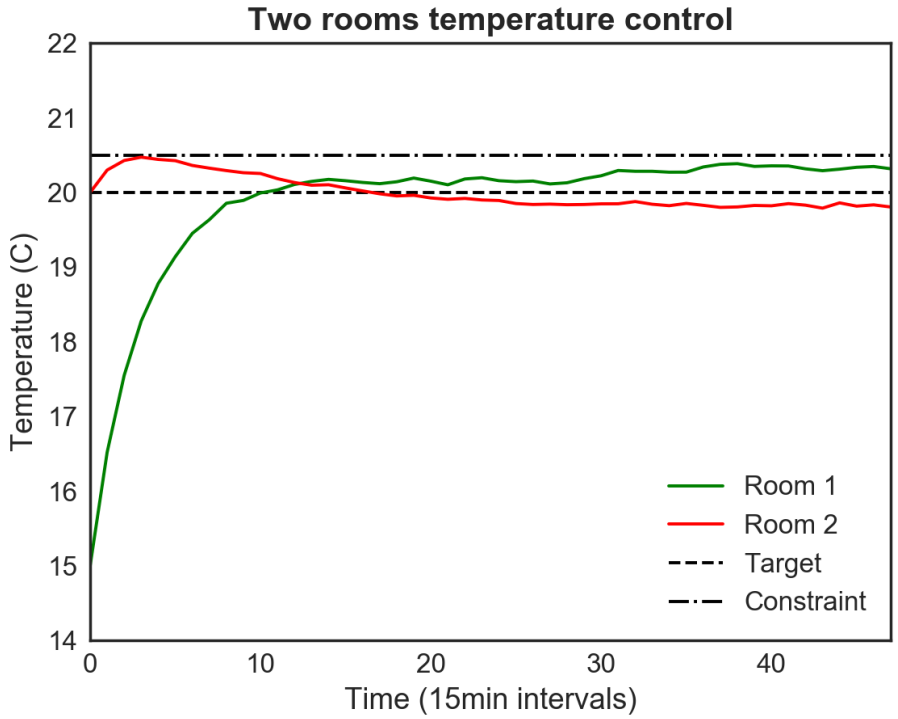
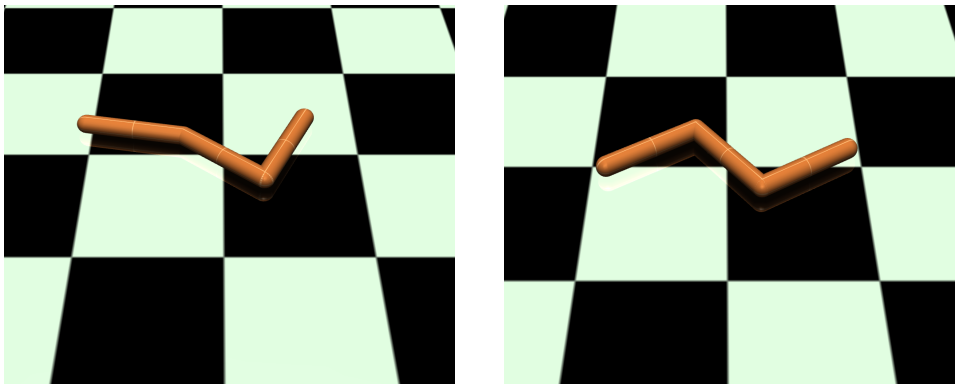


Figure 5.3: Temperature control scenario where room 1 starts with a significantly lower temperature than the target. The linear controller used has been trained with our algorithm (SafePILCO).



(a) Max angle at the first joint (b) Frame from a healthy gait

Figure 5.4: Various configurations of the Swimmer Robot

5.4.3 OpenAi Gym experiments - Swimmer

Next we evaluate our method using a challenging task from the popular OpenAi-gym set of benchmarks for RL. Specifically we work with Swimmer-

v2, a scenario where a robot with two joints navigates a 2-d plane by "swimming" in a viscous fluid. We used the same task to evaluate our PILCO implementation in Chapter 4.

The scenario does not originally provide safety constraints. We therefore impose constraints on the mechanism by limiting the angles of the two joints to a maximum value, with the intuition being that pushing the joints to the edge of their working range can lead to damage, either from the accumulated stress to the joints themselves or by having parts of the robot collide. Furthermore, an interesting qualitative observation is that in many runs of the simulation, in the absence of constraints, a gait emerges that has the robot bend its first joint with full force, effectively using it as single paddle, gaining significant speed and getting high reward early in the episode, but getting stuck in the resulting configuration - often with full force still being applied to one or both joints (see 5.4a). We consider this a characteristic example of the pitfalls that come with the use of RL where failure can have significant costs.

We evaluate two variants of PILCOPen and our proposed algorithm SafePILCO. In these experiments, every episode has 25 time steps (corresponding to 125 steps in the original environment since we subsample by a factor of 5), all algorithms obtain data from $J = 10$ episodes from a random policy at first and then a maximum of $N = 10$ interactions with the environment. The best return metric corresponds to the highest return of these N interactions, averaged over 10 runs with different random seeds. Similarly the number of constraint violations corresponds to the number of time steps with a constraint violation during training (for the N episodes where a trained policy is implemented and not the initial random rollouts), averaged over 10 random seeds. We also report the number of episodes where at least one constraint violation occurred, out of the 10 interactions with a simulator the algorithms are

	PILCOPen		SafePILCO (Add)
ξ	1	10	-
Con. Violations (steps)	59.2 ± 28.8	1.3 ± 3.6	3.2 ± 4.07
Con. Violations (epis.)	7.4 ± 2.6	0.3 ± 0.64	1.0 ± 1.0
Best Return	10.63 ± 0.87	4.80 ± 2.09	7.63 ± 2.15

Table 5.5: Results for the swimmer environment. Con. violations (steps) refer to the total number of time steps where a constraint is violated during training, while con. violations (epis.) counts the different episodes (out of 10) where a constraint violation occurred. All results are averaged over 10 runs, and reported along with one standard deviation.

allowed.

Table 5.5 summarises the experimental results. The proposed algorithm SafePILCO, ends up violating the constraints with the maximum allowed frequency ($\epsilon = 0.1$ here, and we have 1 constraint violating episode out of 10 on average). PILCOPen, when using a small penalty, achieves better performance with a higher number of constraint violations, while a higher penalty leads to safer behaviour and decreased performance, as expected.

5.5 Discussion

In this work we propose a method to integrate model-based policy search with safety considerations throughout the training procedure. Emphasis is given to data-efficiency, since we require our approach to be suitable for applications on physical systems. Using a state-of-the-art PILCO framework, we incorporated constraints in the training, estimating the risk of a policy violating the constraints and preventing high risk policies from being applied to the system. Our contribution uses probabilistic trajectory predictions obtained as model outputs in two ways: (a) to evaluate the probability of constraint violation and (b) as part of a cost function, to be combined with performance consid-

erations. Furthermore, the proposed adaptive scheme successfully allows a trade-off between the two objectives of safety and performance, alleviating the need for extensive hyperparameter tuning.

5.6 Supplementary material

5.6.1 Experiments - details and hyperparameters

Collision Avoidance

The original dynamical system is a very simple linear system, described by $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ with

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b}{M} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.18}$$

and $\mathbf{B} = [0, 1, 0, 0]^T$. Table 5.6 includes the values for other relevant parameters. The multiple (6) values for the initial state mean correspond to different variations of the collision avoidance scenario. Performance results reported in Section 5.4.1 are averaged over these variations.

Building Automation Systems

Detailed description of the simulated model used can be found in Cauchi and Abate [24]. The simulator's parameters were trained on data from an experimental setup within the Department of Computer Science at the University of Oxford. The library provided with [24] allows the user to create new models, with multiple rooms, independent or joint temperature control, deterministic dynamics, or stochastic, with additive or multiplicative disturbances. For our

Notation	Description	Value
b	Friction Coefficient	1.0
M	Car Mass	1000
J	# of initial rollouts	1
-	Type of Controller	RBF
bf	# of RBF basis functions	50
dt	sampling period	0.5(s)
H	episode duration	25(s)
T	time steps per episode	50
$\mu_{0,1}$	initial state mean	$10[-5, 1, -5, 1]$
$\mu_{0,2}$	initial state mean	$10[-5, 1, -6, 1]$
$\mu_{0,3}$	initial state mean	$10[-5, 1, -7, 1]$
$\mu_{0,4}$	initial state mean	$10[-6, 1, -5, 1]$
$\mu_{0,5}$	initial state mean	$10[-6, 1, -7, 1]$
$\mu_{0,6}$	initial state mean	$10[-7, 1, -7, 1]$
Σ_0	initial state variance	$\text{diag}([1, 0.01, 1, 0.01])$
maxiter	iterations of L-BFGS-B	50
ϵ	max tolerable risk	0.10

Table 5.6: Parameters for the collision avoidance scenario.

experiment we use one of the predefined models, simulating two rooms, with joint temperature control, and additive disturbances present (see section 3.2 in [24]). The simulator models the dynamics as a linear time-invariant dynamical system, described by $\mathbf{x}[k+1] = \boldsymbol{A}\boldsymbol{x}[k] + \boldsymbol{B}\boldsymbol{u}[k] + \boldsymbol{F}\mathbf{d}[k] + \boldsymbol{Q}_c$, where matrices $\boldsymbol{A}, \boldsymbol{B}$, according to usual conventions, refer to the inherent system dynamics with $\boldsymbol{u}[k]$ being the controller's input, while $\boldsymbol{F}$ model the effects of additive disturbances and $\boldsymbol{Q}_c$ captures constant terms of the model. For the exact numerical values of these parameters see the Appendix of Cauchi and Abate [24].

Note that in the original publication [24] not all state variables are observable. The observations come as $\mathbf{y} = \boldsymbol{C}\mathbf{x}$, with $\boldsymbol{C} = [1, 0, 0, 0, 0, 0, 0]$. Since our algorithm requires full observability we assume access to the full state $\mathbf{x}$. This is one of the reasons that our results are not directly comparable with that of other studies in the domain (see Abate et al. [2] for an evaluation of various

Notation	Description	Value
J	# of initial rollouts	6
N	# training episodes per run	5
-	Type of Controller	Linear
dt	sampling period	15 (min)
H	episode duration	12 (h)
T	time steps per episode	48
μ_0	initial state mean	$[15, 20, 18, \ldots, 18]$
Σ_0	initial state variance	$0.2I_d$
maxiter	iterations of L-BFGS-B	50
ϵ	max tolerable risk	0.05

Table 5.7: Parameters for the BAS scenario.

methods); the second main reason is that most of these approaches assume an *a priori* existing model. With the dynamical system defined, we set the initial and target state, and the cost function as described in Section 5.4.2. Table 5.7 includes further parameter values used for the experiments in that Section.

Swimmer

The swimmer environment we use is coming unmodified from the OpenAi-gym Python package, and is using the Mujoco physics simulator. The parameters used for training are summarised in Table 5.8.

Notation	Description	Value
J	# of initial rollouts	10
N	# training episodes per run	10
-	Type of Controller	RBF
bf	# of RBF basis functions	30
T	time steps per episode	25
μ_0	initial state mean	$\mathbf{0}$
Σ_0	initial state variance	$0.1I_d$
maxiter	iterations of L-BFGS-B	40
ϵ	max tolerable risk	0.10

Table 5.8: Parameters for the Swimmer experiment.

6

Safety Guarantees for Iterative Predictions with Gaussian Processes

6.1 Chapter Overview

Gaussian processes (GPs) have been extensively used for modelling due to the variety of suitable properties they possess [1]: they are probabilistic models, providing uncertainty estimates on their predictions, both within and outside of the RL research community; they are non-parametric, effectively adjusting the model complexity to the data, and finally they are usually data-efficient [118]. In plenty of scenarios (e.g. planning, forecasting, and time-series modelling) one needs to make several, possibly correlated, predictions at once (the second prediction is made before the first one can be evaluated versus a ground truth, and so on). For this we can discern two options: either train multiple models, each one predicting at different time-scales, or use a single model, that itera-

[1] This chapter is largely based Polymenakos et al. [115]

tively computes predictions that get in turn fed back as input to the model in the next step. We refer to the latter as *iterative predictions* and *iterative planning*.

Of particular interest for the iterative planning scenario is the model-based reinforcement learning setting, where a GP model is used to evaluate a candidate control policy on the system. The evaluation requires the model to provide predictions for the system's state over multiple time-steps under the proposed policy. It is important in these cases to have a realistic assessment of the error on the predictions, as this allows quantification of the risk of costly system failures, like collisions with obstacles or financial losses, and analysis of safety-critical applications. In such settings, we require predictions that are not only accurate on average, but also provide robust, (probabilistically) guaranteed worst-case accuracy.

Unfortunately, as GP models output probability distributions, iterative planning poses the problem of prediction over successive *noisy inputs*, i.e. with a distribution placed over the input space. This leads to an analytically intractable problem for such non-linear input-output mappings (see Section 2.2.3). While several approximation techniques have been proposed [52, 148], to the best of our knowledge, none of them provides guarantees, in the form of formal error bounds on their estimations, making it difficult to estimate reliability and trust predictions in application scenarios (Section 3.2.2).

In this chapter we provide a probabilistic bound for iterative predictions with GPs and develop a method for its explicit computation. Given a user-defined tolerance $\epsilon > 0$, our method works by computing probabilistic bounds at each prediction step and propagating them over multiple time-steps in the form of intervals. The GP trajectories are guaranteed to lie inside these intervals at each time step with probability at least $1-\epsilon$. In practice, this allows us to perform long-term predictions for the GP trajectory with the prediction prov-

ably staying within known bounds with a specified probability. We further show how the bound can be used within a reinforcement learning scenario, in order to guarantee the safety of proposed control policies. We provide an algorithmic framework for the explicit computation of every value involved in the bound calculation, directly and efficiently from data, so that the bound can be explicitly computed independently of the form of the learned GP.

On a set of case studies, we show how our method can correctly probabilistic bounds that account for the GP uncertainty over its trajectories. Finally, we illustrate how our bound can be successfully employed to verify both open loop and feedback policies and therefore guarantee the safety of proposed controllers for the learned GP. In summary, the paper makes the following main contributions:

- We develop a formal bound, for iterative prediction settings, on the probability that the trajectories of a GP lie inside a specific region. We provide explicit computational techniques for calculating the bound.
- We incorporate control laws and take into account their effects on the model's trajectories.
- We provide experimental validations of our method, highlighting cases in which a competitive state-of-the-art method fails to properly propagate the GP uncertainty. We provide case studies on certification of open-loop and feedback policies.

6.2 Related Work

Performing iterative predictions, and using them for planning, is an extensively studied problem across various model types [1, 55, 71, 151].

In particular, GP multi-step-ahead prediction is generally achieved using heuristic approximations [52]. The most widely used approach is Moment Matching (MM) which computes a Gaussian approximation over the (non-Gaussian) output distribution of a GP for a noisy input [52, 22]. The uncertainty estimated in this fashion can then be leveraged to learn control policies in frameworks such as PILCO [34, 30]. It is also the method we employed in Chapters 4 and 5, as well as in various extensions of PILCO that have been proposed [33, 31, 80, 94]. However, building on MM, all the cited approaches inherently fail to take into account multi-modal behaviour and tend to underestimate uncertainty. As such, the synthesised policies are not guaranteed to be safe. The method we present in this chapter on the other hand comes with probabilistic guarantees that allow us to compute the subregions of the input space in which the trajectories of the analysed GP are bound to lie with high probability. As such it provides formal, guaranteed bounds on the GP trajectories and makes no particular assumptions on the GP model, enabling its use in safe reinforcement learning scenarios.

In Section 3.2.2 we reviewed a number of alternative approaches to moment matching, with a similar focus to safety as our method [149, 148, 78]. Numerical approximations have been proposed for multi-step-ahead predictions [148] where the output distribution is directly approximated by using quadrature formulas and, in principle, worst-case scenario error bounds could be computed using existing techniques for numerical quadrature [149]. However, the analysis that leads to the bounds proposed in [149] is focused on stability, with the assumption that trajectories monotonically decrease the distance to a target state, and the authors explicitly exclude trajectories that move away from the target state before eventual convergence and stabilisation. In [148], where more general tasks are solved, no formal bounds are provided.

Our algorithm instead provides valid probabilistic bounds for the general case. Furthermore, the bounds provided by Koller et al. [78] require the computation of constants very difficult to compute in practice. In contrast, in this paper we assume that the underlying function is a sample from a GP (and hence we do not consider any possible model mismatch) and derive formal bounds whose required constants are directly computed.

Formal and probabilistic guarantees for GPs have been discussed in [23] and [19] for regression and classification with GPs, respectively. Albeit formal, these methods cannot be directly applied to multi-step-ahead predictions scenarios as they are designed for GPs over single input points. Whereas, our method, by propagating probabilistic bounds through each time step is applicable to multi-step ahead prediction scenarios and can be used in reinforcement learning settings to verify controller safety.

6.3 Bounds for Multi-step Ahead Predictions with Gaussian Processes

Given an input space $\mathcal{U} \subset \mathbb{R}^m$ and a time horizon $[0, T]$, for $t \in \{0, \dots, T-1\} \subset \mathbb{N}$ we consider a stochastic dynamical system

$$\mathbf{x}_{t+1} = \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{u}_t), \quad \boldsymbol{u}_t \in \mathcal{U}, \tag{6.1}$$

where we assume that for $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^n$, $\mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t) \sim \mathcal{N}(\boldsymbol{\mu}^{\mathbf{f}}_{\boldsymbol{x}}, \boldsymbol{\Sigma}^{\mathbf{f}}_{\boldsymbol{x},\boldsymbol{x}})$ that is, $\mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t)$ is normally distributed with mean vector $\mu^{\mathbf{f}}_{\boldsymbol{x}}$ and covariance matrix $\Sigma^{\mathbf{f}}_{\boldsymbol{x},\boldsymbol{x}}$[2]. Mean and variance of $\mathrm{f}^i(\boldsymbol{x}, \boldsymbol{u}_t)$, the i-th component of $\mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t)$, are denoted with $\boldsymbol{\mu}^{\mathbf{f},i}_{\mathbf{x}}$ and $\boldsymbol{\Sigma}^{\mathbf{f},(i,i)}_{\boldsymbol{x},\boldsymbol{x}}$. Intuitively, x_t is a discrete-time stochastic process,

[2]For simplicity we drop the dependence on $\boldsymbol{u}_t$ in both mean vector and covariance matrix.

whose time evolution depends on an input signal taking values in $\mathcal{U}$. A parametric memory-less and deterministic policy $\pi^{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{U}$ with parameters $\boldsymbol{\theta}$ is a function that assigns a control input given the current state. By iterating equation (6.1), we have that, for $t > 0$, $\mathbf{x}_t$ is a random variable as it is the output of process $\mathbf{f}$. Subsequently, the problem of multi-step ahead predictions is cast as a problem of prediction over noisy inputs.

6.3.1 Prediction over noisy inputs

For a given $\boldsymbol{x} \in \mathcal{X}, \boldsymbol{u} \in \mathcal{U}$ we have that $\mathbf{f}(\boldsymbol{x}, \boldsymbol{u})$ is a Gaussian random variable. However, if $\mathbf{x}_t$ is a random variable itself (which is the case for prediction over noisy inputs), then $\mathbf{f}(\mathbf{x}_t, \boldsymbol{u})$ is generally not Gaussian and its distribution is more often than not analytically intractable. In particular, we have that

$$\mathbf{f}(\mathbf{x}_t, \boldsymbol{u}) \sim \int p(\mathbf{x}_{t+1}|\boldsymbol{x}, \boldsymbol{u}) p(\mathbf{x}_t = \boldsymbol{x}) d\boldsymbol{x} \tag{6.2}$$

where $p(\mathbf{x}_{t+1}|\boldsymbol{x}, \boldsymbol{u})$ is the (normal) distribution of $\mathbf{f}(\mathbf{x}, \boldsymbol{u})$ and $p(\mathbf{x}_t = \boldsymbol{x})$ is the distribution of $\mathbf{x}_t$. As a consequence, the predictive distribution for $\mathbf{x}_{t+1}$ is not Gaussian and approximations are required [52].

In this chapter, given $\mathbf{x}_t$, we consider a predictor $\hat{\boldsymbol{x}}_t$ for $\mathbf{x}_t$, such that

$$\hat{\boldsymbol{x}}_t = g(\hat{\boldsymbol{x}}_{t-1}, \boldsymbol{u}_{t-1}), \tag{6.3}$$

where $g(\hat{\boldsymbol{x}}_{t-1}, \boldsymbol{u}_{t-1})$ is a deterministic function. That is, $\hat{\boldsymbol{x}}_t$ is a deterministic process that predicts the value of $\mathbf{x}_t$. We set $\hat{\boldsymbol{x}}_t$ equal to the mean of $\mathbf{x}_t$, as estimated with moment matching techniques [52].

In what follows, in Theorem 1 we compute a probabilistic bound on the error between $\hat{\boldsymbol{x}}_t$ and $\mathbf{x}_t$. The bound has a recursive structure, as the uncertainty

needs to be propagated over multiple prediction steps. Note that this is not a modelling error, coming from the GP imperfectly capturing the behaviour of an underlying system, but comes solely from propagating the uncertainty while performing iterative predictions. Then, in Corollary 1 we show that, given an $\epsilon > 0$, this bound can be used to build a tube around $\hat{\boldsymbol{x}}_t$ such that, at each time step, the trajectories of $\mathbf{x}_t$ are guaranteed to be within such tube with probability at least $1 - \epsilon$. For any safe region $\mathcal{S} \subset \mathcal{X}$ we can hence produce certificates on whether GP trajectories will lie inside that region with high probability or not.

6.3.2 Bounds for Multi-Step Ahead Predictions

Consider as a random variable the error at time t, i.e. $\mathrm{e}_t = |\mathbf{x}_t - \hat{\boldsymbol{x}}_t|_1$ and a constant $K_t > 0$. In Theorem 1 we compute the probability that the error between $\mathbf{x}_t$ and $\hat{\boldsymbol{x}}_t$ is greater than K_t, namely $P(\mathrm{e}_t > K_t)$.

Theorem 1 *For any $K > 0$ and $\boldsymbol{x}^* \in \mathcal{X}$, let $I^K_{\boldsymbol{x}^*} = \{\boldsymbol{x} \in \mathcal{X} \,:\, |\boldsymbol{x}^* - \boldsymbol{x}|_1 \leq K\}$. Assume $\mathbf{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_{0,0})$. Then, for arbitrary constants $K_{t+1}, K_t > 0$, it holds that*

$$\begin{aligned} P(\mathrm{e}_{t+1} > K_{t+1}) \;\leq\; \\ P(\sup_{\boldsymbol{x} \in I^{K_t}_{\hat{x}_t}} |\hat{x}_{t+1} - \mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t)|_i > K_{t+1}) P(\mathrm{e}_t \leq K_t) \\ + P(\mathrm{e}_t > K_t), \end{aligned}$$

with

$$P(\mathrm{e}_0 > K_0) = 1 - \int_{I^{K_0}_{\hat{x}_0}} \mathcal{N}(z \,|\, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_{0,0}) d\boldsymbol{z}$$

for any $K_0 > 0$, where $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_{0,0}$ are respectively the mean and covariance of $\mathbf{x}_0$.

The proof of the above theorem is reported in Section 6.6. The resulting bound

in Theorem 1 is recursive. Hence, in order to estimate the prediction error at time t, we need to compute the prediction error at the previous time steps, which is propagated over time through the bound. The recursion terminates as the distribution for $\mathbf{x_0}$, that is the initial condition, is given. Intuitively K_t is a parametric cutoff threshold for the distance at time t, and the resulting bound at time $t+1$, that is e_{t+1}, is the sum of the contribution obtained by assuming that $\mathrm{e}_t \leq K_t$ and by the contribution of assuming $\mathrm{e}_t > K_t$ (and remains valid for any value of K_t).

Note that the bound in Theorem 1 requires the computation of $P(\sup_{\boldsymbol{x}\in I_{\hat{\boldsymbol{x}}_t}^{K_t}} |g(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t) - \mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t)|_1 > K_{t+1})$ that is, the probability that the supremum of a stochastic process is greater than a given threshold. This is in general a difficult problem [5]. However, $\mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t)$ is a Gaussian process and $g(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t)$ a constant. Therefore, we can use the result from [23], where bounds for the supremum of a GP have been derived. These are extended to the current setup in the following proposition.

Proposition 1 *Let $\mu(\boldsymbol{x}, \hat{\boldsymbol{x}}_t) = g(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t) - \mu_{\boldsymbol{x}}^{\mathbf{f}}$. Assume $I_{\hat{\boldsymbol{x}}_t}^{K_t}$ is a hyper-cube with sides of length $D > 0$. For $i \in \{1, ..., n\}$ let*

$$\bar{\eta}^i = \frac{K_{t+1} - \sup_{\boldsymbol{x}\in I_{\hat{\boldsymbol{x}}_t}^{K_t}} |\mu(\boldsymbol{x}, \hat{\boldsymbol{x}}_t)|_1}{n} - 12 \int_0^{\lambda^i} \sqrt{\ln\left(\left(\frac{\sqrt{n} L_{\hat{\boldsymbol{x}}_t}^i D}{z} + 1\right)^n\right)} dz,$$

with $\lambda^i = \frac{1}{2} \sup_{\boldsymbol{x}_1, \boldsymbol{x}_2 \in I_{\hat{\boldsymbol{x}}_t}}^{K_t} d_{\hat{\boldsymbol{x}}}^{(i)}(\boldsymbol{x}_1, \boldsymbol{x}_2)$ and n being the dimension of the state space. For each $i \in \{1, ..., n\}$ assume $\bar{\eta}^i > 0$. Then, it holds that

$$P(\sup_{\boldsymbol{x}\in I_{\hat{\boldsymbol{x}}_t}^{K_t}} |g(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t) - \mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t)|_1 > K_{t+1}) \leq 2 \sum_{i=1}^{n} e^{-\frac{(\bar{\eta}^i)^2}{2\xi^{(i)}}},$$

where $\xi^{(i)} = \sup_{\boldsymbol{x} \in I^{K_t}_{\hat{\boldsymbol{x}}_t}} \boldsymbol{\Sigma}^{\mathbf{f},(i,i)}_{\boldsymbol{x},\boldsymbol{x}}$,

$$d^{(i)}_{\hat{\boldsymbol{x}}_t}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{\mathbb{E}\left[\left(\mathbf{f}^i(\boldsymbol{x}_2, \boldsymbol{u}_t) - \mu^{\mathbf{f},i}_{\boldsymbol{x}_2} - (\mathbf{f}^i(\boldsymbol{x}_1, \boldsymbol{u}_t) - \mu^{\mathbf{f},i}_{\boldsymbol{x}_1})\right)^2\right]},$$

and $L^i_{\hat{\boldsymbol{x}}_t}$ *is a local Lipschitz constant for* $d^{(i)}_{\hat{\boldsymbol{x}}_t}$.

By using the upper bound of Proposition 1 in Theorem 1 we can propagate the bound through time for any value of $K_t > 0$, $t = 0, \ldots, T$. This give us the degree of freedom necessary to iteratively select, given K_t, the values for K_{t+1} that meet an a priori specified probabilistic error $\epsilon > 0$. To do this it suffices to evaluate the one-step bound resulting from the combination of Proposition 1 and Theorem 1, and choose the smallest value of K_{t+1} such that $P(\mathrm{e}_{t+1} > K_{t+1}) < \epsilon$.

Corollary 1 (*of Theorem 1*) *For any* $\epsilon > 0$ *pick the smallest* $K_0, ..., K_T$ *such that for any* $t \in \{0, ..., T\}$ *we have that* $P(\mathrm{e}_t > K_t) < \epsilon$. *Then, this implies that*

$$\forall t \in \{0, ..., T\}, \quad P(\mathbf{x}_t \in I^{K_t}_{\hat{x}_t}) > 1 - \epsilon.$$

Proof of Corollary 1

Proof 1 *Consider a timestep* $t+1$, *for which we need select a* K_t *for* $t \in \{0, \ldots, T-1\}$ *so that* $P(\mathrm{e}_{t+1} > K_{t+1}) < \epsilon$. *Also assume that* K_t *is known. Using Theorem 1, this is equivalent to showing*

$$P(\sup_{\boldsymbol{x} \in I^{K_t}_{\hat{x}_t}} |\hat{x}_{t+1} - \mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t)|_i > K_{t+1}) P(\mathrm{e}_t \leq K_t) + P(\mathrm{e}_t > K_t) \leq \epsilon$$

$$P(\sup_{\boldsymbol{x} \in I^{K_t}_{\hat{x}_t}} |\hat{x}_{t+1} - \mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t)|_i > K_{t+1}) \leq \frac{1}{P(\mathrm{e}_t \leq K_t)}(\epsilon - P(\mathrm{e}_t > K_t))$$

But by Proposition 1

$$P(\sup_{\boldsymbol{x}\in I_{\hat{x}_t}^{K_t}} |\hat{x}_{t+1} - \mathbf{f}(\boldsymbol{x}, \boldsymbol{u}_t)|_i > K_{t+1}) \leq 2\sum_{i=1}^{n} e^{-\frac{(\bar{\eta}^i)^2}{2\xi^{(i)}}},$$

so we need to show that

$$2\sum_{i=1}^{n} e^{-\frac{(\bar{\eta}^i)^2}{2\xi^{(i)}}} leq \frac{1}{P(\mathrm{e}_t \leq K_t)}(\epsilon - P(\mathrm{e}_t > K_t)),$$

and we can make $\sum_{i=1}^{n} e^{-\frac{(\bar{\eta}^i)^2}{2\xi^{(i)}}}$ *arbitrary small by choosing a large enough* K_{t+1}*, since as* K_{t+1} *increases,* $\bar{\eta}^i$ *also increases.*

As a result we can compute a sequence of subsets $I_{\hat{x}_t}^{K_t}$ of the state space such that the GP trajectories are bounded to stay inside them with probability at least $1-\epsilon$ at each time step. Given a safe region $\mathcal{S} \subseteq \mathcal{X}$ we can hence produce a certificate on the GP trajectories lying inside S with probability at least $1-\epsilon$ by checking the intersection between the $I_{\hat{x}_t}^{K_t}$ and $\mathcal{S}$.

6.3.3 Background on Bounds in Bayesian Learning Settings

GPs have been studied and used widely for many years [118] but the seminal work explicitly bounding modelling error, and subsequently regret, in an optimisation setting using GPs is Srinivas et al. [136]. A key assumption is that the function modelled by the GP is either sampled from a GP, or has a bounded reproducing kernel Hilbert space norm. This work had great impact, both from a practical perspective, as it grounded theoretically and provided an efficient algorithm for the use of GPs for Bayesian optimisation, but also from a methodological standpoint, as it presented a general technique for obtaining regret bounds for kernel methods. The result we are using and expanding on

here from Cardelli et al. [23] refers to the distinct problem of bounding not the modelling error, but the output of the GP for a bounded disturbance in the input. This key result is presented here in Proposition 1. The derivation of the bound makes use of the Borell-TIS inequality and Dudley's entropy integral [5]. The full derivation can be found in the supplementary material of Cardelli et al. [23]. In order to use the result in practice, several quantities of interest need to be computed explicitly for the given GP.

Specifically, the bound in Proposition 1 requires the computation of $\sup_{\boldsymbol{x}\in I^{K_t}_{\hat{\boldsymbol{x}}_t}} |\mu^i(\boldsymbol{x}, \hat{\boldsymbol{x}}_t)|_1$, $\xi^{(i)}$, $L^i_{\hat{\boldsymbol{x}}_t}$ and λ_1, which are related to the extrema of the mean and variance of the GP $\mathbf{f}$ in $I^{K_t}_{\hat{\boldsymbol{x}}_t}$ and to a Lipschitz constant on $d^{(i)}_{\hat{\boldsymbol{x}}_t}$. In a Bayesian learning setting, these can be computed by relying on the methods discussed in [23] and applying them to the GP of Equation (6.1). We here briefly review and adapt to the current settings the methods for the bounding of $\sup_{\boldsymbol{x}\in I^{K_t}_{\hat{\boldsymbol{x}}_t}} |\mu^i(\boldsymbol{x}, \hat{\boldsymbol{x}}_t)|_1$, $\xi^{(i)}$, while we refer to Cardelli et al. [23] for a detailed explanation of how to compute $L^i_{\hat{\boldsymbol{x}}_t}$ and λ_1 (which are not changed by the control input) Let $k^i(\cdot,\cdot)$ be the GP kernel function for the i-th output dimension, $\boldsymbol{x} \in I^{K_t}_{\hat{\boldsymbol{x}}_t}$ be a test point and $\mathcal{D} = \{(\boldsymbol{x}_j, \boldsymbol{y}_j) | j = 1, \ldots, M\}$ a training data set. Then the mean and variance of the Gaussian process $\mathbf{f}$ conditioned on the training data is given by the set of Equations (2.8) (also in Rasmussen and Williams [118]). Reiterating here, if $\boldsymbol{K}^i$ is a matrix with $\boldsymbol{K}^i = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{D}$, and $\boldsymbol{K}^i_*$ a vector with $\boldsymbol{K}^i_j = k(\boldsymbol{x}_j, \boldsymbol{x})$ for $\boldsymbol{x}_j \in \mathcal{D}$:

$$\mu^{\mathbf{f},i}_{\boldsymbol{x}} = \boldsymbol{K}^i_*(\boldsymbol{K}^i)^{-1}\boldsymbol{y} \tag{6.4}$$

$$\Sigma^{\mathbf{f},(i,i)}_{\boldsymbol{x},\boldsymbol{x}} = k(\boldsymbol{x},\boldsymbol{x}) - \boldsymbol{K}^i_*(\boldsymbol{K}^i)^{-1}(\boldsymbol{K}^i_*)^T \tag{6.5}$$

where $\boldsymbol{y} = [y_1, \ldots, y_M]^T$. Assuming continuity and differentiability of the kernel function $k(\cdot,\cdot)$, it is possible to find linear upper and lower bounds on the

covariance between a test point and a point in the training dataset. In the case of squared exponential kernel it suffices to see that the covariance between a test point $\boldsymbol{x}$ and a training point $\boldsymbol{x}_j$ can be written as a differentiable, convex function of the uni-dimensional auxiliary variable $z_j = ||\boldsymbol{x} - \boldsymbol{x}_j||$. As such, by inspection of the derivatives it is possible to find linear coefficients a_j^L, b_j^L, a_j^U and b_j^U such that[3]:

$$a_j^L + b_j^L||\boldsymbol{x} - \boldsymbol{x}_j|| \leq k^i(\boldsymbol{x}, \boldsymbol{x}_j) \leq a_j^U + b_j^U||\boldsymbol{x} - \boldsymbol{x}_j||, \ \forall \boldsymbol{x} \in I_{\hat{\boldsymbol{x}}_t}^{K_t}. \tag{6.6}$$

These bounds can be propagated through the inference formula for $\mathbf{f}$ by performing the matrix multiplication involved in Equations (6.4) and (6.5). The resulting equation for the mean and variance are respectively linear and quadratic on the auxiliary variable $z_j = ||\boldsymbol{x} - \boldsymbol{x}_j||$, and can hence be optimised analytically by inspection of the derivatives. This can then be further refined using a branch and bound optimisation approach over $I_{\hat{\boldsymbol{x}}_t}^K$.

This can be straightforwardly generalised to take into account the extra input dimensions coming from a deterministic control strategy $\pi(\boldsymbol{x}) = \boldsymbol{u}$, without increasing the size of the branch and bound search space, that is without significantly change the computational time. To do so it suffices to solve the optimisation problems $u_j^L = \min_{\boldsymbol{x} \in I_{\hat{\boldsymbol{x}}_t}^{K_t}} \pi_j(\boldsymbol{x})$ and $u_j^U = \max_{\boldsymbol{x} \in I_{\hat{\boldsymbol{x}}_t}^{K_t}} \pi_j(\boldsymbol{x})$ for $j = 1, \ldots, m$, that is computing maximum and minimum of the control allowed in the current state space sub-region. Notice that for the classes of policy functions that we use (e.g. linear or sum of radial basis functions) this can be computed analytically and in constant time [34]. The bounds can then be used by treating $\boldsymbol{u}$ and $\boldsymbol{x}$ symmetrically.

[3]By definition of convex function, a lower bound is given by any tangent to the function (computed through derivative calculations) and an upper bound is given by connecting the extrema of the function in $I_{\hat{\boldsymbol{x}}_t}^{K_t}$.

Computational Complexity of Bound

Computation of the bound involves the calculation of Equation (6.6) for each point in the training set $\mathcal{D}$, and the computation of the inference formulas (6.4) and (6.5) on the resulting bounds. This is $\mathcal{O}(M)$ for the mean function and $\mathcal{O}(M^2)$ for the variance (as the latter is quadratic), where $M = |\mathcal{D}|$ is the number of training samples used. Refining the bounds with a branch and bound approach has a worst-case cost that is exponential in n, the dimension of the variable $\boldsymbol{x}$. Bounding of $L^i_{\hat{\boldsymbol{x}}_t}$ and λ_1 is done in constant time. This is iterated for any output dimension of the GP. After branch and bound converges, computation of the optimal value for K_{t+1} is linear on the number of candidate values explored, as it involves the computation of the integral in Proposition 1 with known constants. Finally the procedure is identically repeated for each time step t.

6.3.4 Using the Safety Guarantees for PILCO

In this section we briefly examine how the safety guarantees can be used in conjunction with the Safe PILCO framework of Chapter 5. Safe PILCO introduces constraints that demand the system to stay in a safe subset of the state space $\mathcal{S} \subseteq \mathcal{X}$ with high probability. Specifically, after a controller is trained using a learned GP model, and before the controller is applied to the controlled system, the probability that this controller violates the constraints is estimated using moment matching. Since moment matching is an approximation that might lead to underestimating the true uncertainty of the iterative predictions (as we show below) controllers that violate the constraints can be allowed to be implemented. We therefore suggest to replace this step, referred to in Chapter 5 as *a safety check*, with the bounds estimated from Corollary 1. This replace-

ment is straightforward and provides better protection from unsafe controllers used in possibly safety critical applications.

In Algorithm 3 we present the modified algorithm for Safe PILCO. In more detail, Step 5 employs the safe PILCO framework to synthesise a candidate policy (optimising its parameters $\boldsymbol{\theta}$) by maximising the composite objective function $J^{\pi}(\boldsymbol{\theta})$. Note that the objective function component related to safety, Q, can take various forms as long as it is correlated with the probability of violating the constraints (the surrogate losses tested in Chapter 5). In principle, the results of Corollary 1 could also be used in this step to replace moment matching. We do not do this as because Step 5 requires the policies to be evaluated multiple times, depending on the optimiser's budget (in current experiments 50-100 times). Hence, the computational burden required to recompute the bounds so many times may make policy optimisation very costly computationally. Furthermore, note that in safe PILCO moment matching allows one to obtain analytic gradients of the objective function with respect to the parameters $\boldsymbol{\theta}$ of the policy $\pi^{\boldsymbol{\theta}}$, which would not be available using Corollary 1. Due to these points, we propose using the new bounds only for Step 6, replacing the safety check.

Algorithm 3 High level algorithm description for modified Safe PILCO with safety guarantees.

Initialise policy parameters $\boldsymbol{\theta}$
Interact with the system, collect data
Train GP model on the data
repeat
    Policy optimisation, maximise $J^{\pi}(\boldsymbol{\theta}) = R^{\pi}(\boldsymbol{\theta}) + wQ^{\pi}(\boldsymbol{\theta})$
    Calculate bounds for chosen ϵ from Corollary 1
    if Bounded trajectory satisfies constraints with probability at least $1 - \epsilon$ **then**
        Interact with the system, collect data
        Retrain GP model on the new data set
    else
        Return to policy optimisation with an adapted objective function that has higher w
until task learned (satisfactory performance achieved or limited computational time/number of iterations reached.)

6.4 Experiments

In this section we apply the methods presented above to various GPs with SQE kernel trained from data. In all the experiments we use the bound from Theorem 1 with the L1 norm, that is with $d = 1$. First we explicitly compare our formal, guaranteed bounds with the probability estimation obtained by Moment Matching (MM) in two iterative prediction scenarios (with no control involved). We then investigate in the Mountain Car application [99] the behaviour of our methodology for certification of a given control policy. Finally we show how to compute bounds for the behaviour of closed-loop systems for a given controller. GPs are trained with the GPML package, using maximum marginal likelihood for hyperparameter selection. Candidate policies are either arbitrarily selected for the purpose of demonstration or obtained from PILCO. They are either linear or linear, squashed through a sine wave, to

constrain the input magnitude [34].

6.4.1 Iterative Prediction

We analyse the behaviour of our method in a one-dimensional synthetic dataset where the system dynamics are distributed as a Gaussian at each time step. Further, we assume that the initial state of the system is Gaussian, that is $\mathrm{x}_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$, with mean and variance given by $\mu_0 = 0$ and $\Sigma_0 = 0.01$. We compute predictions and bound the trajectory for an horizon of $T = 10$ time steps. We use $\epsilon = 0.05$, that is we require bounds holding with probability at least 95% and compare with the results obtained by MM. Namely, we compare our bounds with plus/minus two standard deviation estimated by MM. Notice that when MM is exact (i.e. when the system dynamics are effectively Gaussian at each time-step), then this would as well correspond to bounds at 95% probability. Results for this analysis are given in Figure 6.1, where our bound is depicted with a thick red solid line, and MM results are represented by the green shaded area. Further, we extract 100 trajectories from the GP, which are depicted with thin colored lines, in order to provide statistical validation for the results. Notice that the latter are almost entirely within the MM shaded area. In fact, since the system dynamics are fully Gaussian at each time step, that is $\mathbf{x}_t$ is Gaussian for each t, then the approximation made by MM is almost exact and well behaved. Notice that our method successfully bounds the sampled trajectories at each time step.

In the previous example, MM succeeds in bounding the GP trajectories because the Gaussian approximation performed by MM is well suited for that setting. However, as soon as this does not hold anymore, the results obtained with MM fail to bound the actual GP trajectories. As an example of this, con-

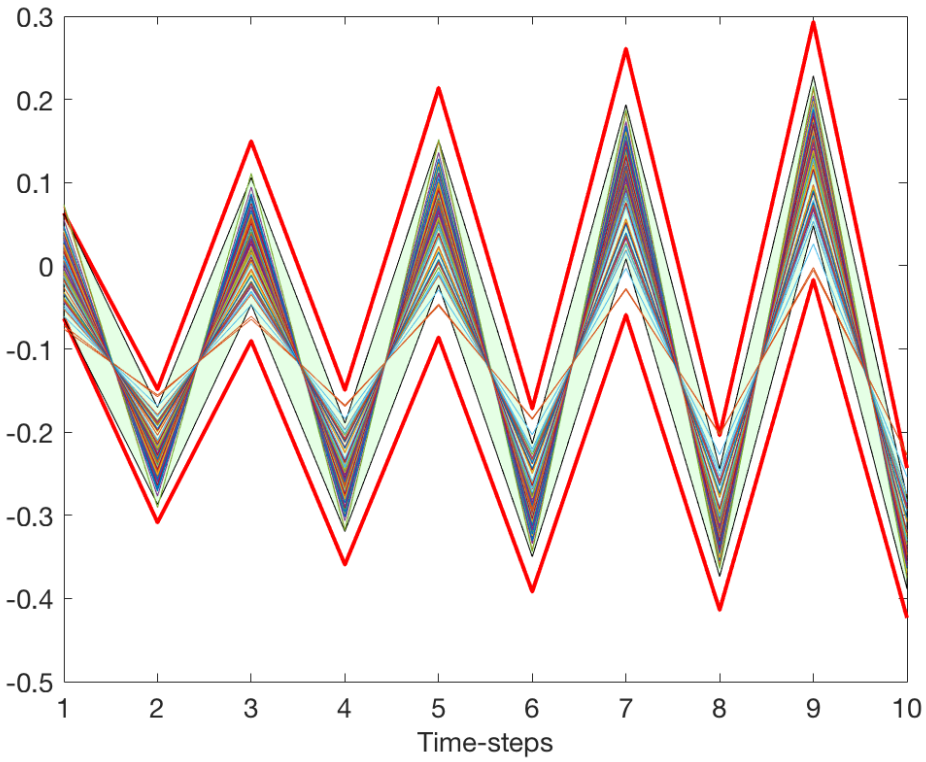


Figure 6.1: A set of 100 trajectories sampled from a GP (thin coloured line). The green shaded area corresponds to plus/minus two standard deviations of the moment matching prediction, and the thicker red lines delimit the area with 95% probability according to Theorem 1.

sider a system with dynamics given by:

$$h(x) = \begin{cases} \text{sign}(x)x^4, \text{ if } |x| < 1 \\ x, \text{ otherwise.} \end{cases} \tag{6.7}$$

We train a GP on data sampled from by this system. With the function being non-linear, we have that x_t is non-Gaussian for $t > 0$, which implies that application of MM will introduce unaccounted approximation errors, see Figure 6.2. Furthermore, the specific dynamics chosen are such that the MM variance prediction will inevitably shrink, leading to a systematic underestimation of the actual region in which GP trajectories are located. In fact when the initial

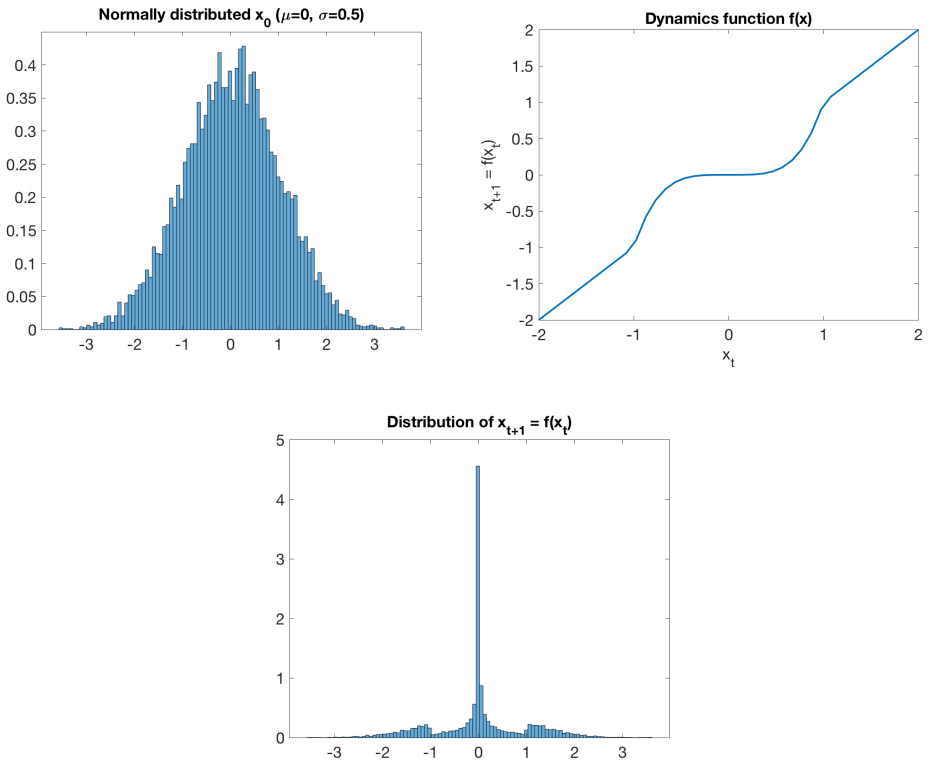


Figure 6.2: Initial state distribution, system dynamics and state distribution after a time-step for the system described by the set of Equations 6.7. Histograms show empirical results for 10000 trajectories. On the **top left** is the normally distributed initial state, which passes through the non-linear dynamics function in the **top right**, leading to the distribution at the **bottom**.

position of the trajectory, x_0, is greater than 1, then the trajectory will constantly be at x_0. As such, assuming $x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$, one can choose a Σ_0 big enough such that the GP trajectories will be outside any tube parallel to the x-axis with probability greater than ϵ. However after finitely-many time steps MM variance will wrongly shrink to values very close to zero, hence failing to account for the majority of the probability mass of the GP.

Empirical results for this system using $\epsilon = 0.05$ are plotted in Figure 6.3, for values of initial variance Σ_0 going from 0.1 to 0.6. The empirical results agree with the discussion above. If the initial variance is small enough, than the overwhelming majority of GP trajectories will converge to zero. However,

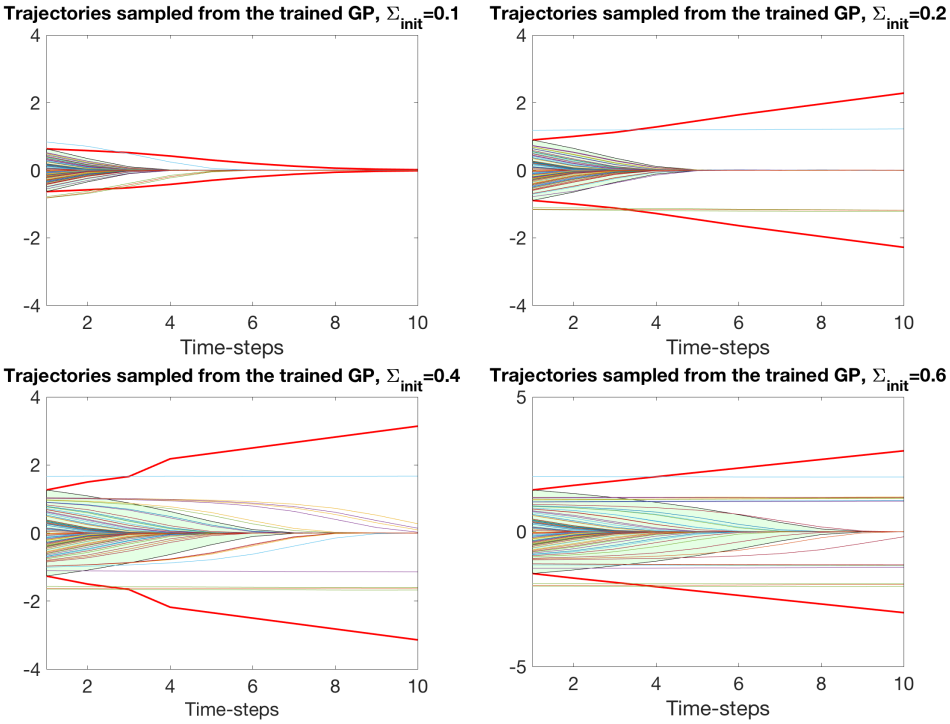


Figure 6.3: As the initial variance increases, more trajectories, having an initial state $|x_0| > 1$, do not converge to 0. Moment matching fails to account for this fact (green shaded area showing two standards deviations). Our bound (red line) grows appropriately. Thinner colored lines represent 100 sampled trajectories from the GP.

as the initial variance grows, more and more trajectories diverge. MM fails to account for this behavior, and the variance predicted by MM fails to mirror the actual dynamics of the GP under analysis. Note that, our method, being guaranteed to provide correct results, is able to successfully bound (up to probability $1 - \epsilon$) the actual trajectory of the GP, independently of the initial variance. In fact, our method does not rely on any particular assumption, and is able to provide worst-case scenario analysis independently of the general shape of the GP.

6.4.2 Open-loop control for Mountain Car

In this section we show how our method can be used to certify a control input for a dynamical system. The environment we are considering is a version of the continuous mountain car problem [99]. Briefly, a car is required to ascend up a hill to its right, with a goal state on top of the hill. Because it does not have enough power to climb the hill directly, it has to initially ascend a hill to the left first to gain potential. The state space has two dimensions (position and velocity of the car), and the control input is one dimensional and corresponds to a force applied to the car.

As previously, we train a GP on data generated from the environment, in this case, following a random policy. We assume we have access to an initial (normal) distribution for the starting state and our task is to to evaluate a proposed sequence of actions. Specifically, we want to perform predictions about the sequence of states (position and velocity) of the car, and to provide high probability bounds for these predictions. The trained GP model has a 3-dimensional input space, as it takes $(\boldsymbol{x}_t, u_t)$ pairs as inputs, corresponding to the two state-space variables and the control input, and 2-dimensional outputs, that correspond to $\boldsymbol{x}_{t+1}$. The two output dimensions correspond to two independent GPs, each one predicting a state variable. However, the predictions of each model are based on the previous predictions of *both* models. In more detail, we assume a state $\boldsymbol{x}_t \in \mathcal{X} \subset \mathbb{R}^2$, where both components of $\boldsymbol{x}_t$ are bounded. These form a tuple $[x_t^1, x_t^2, u]$, where $x_t^1 \in [lb_1, ub_1]$, and $x_t^2 \in [lb_2, ub_2]$, and the exact value of u is known (as we are verifying an arbitrary, fixed control policy). This tuple is the input to the two GP models, with one of them providing the predicted position x_{t+1}^1, with its new lower and upper bound, and the other one providing the same quantities for the velocity x_{t+1}^2.

t	Control u	x^1	x^2	Bound x^1	Bound x^2
1	1.85	-0.50	0.00	0.020	0.020
2	-0.97	-0.38	0.53	0.030	0.080
3	1.39	-0.37	-0.49	0.055	0.125
4	0.17	-0.53	-0.20	0.105	0.220
5	-1.95	-0.57	-0.02	0.130	0.405
6	-	-0.87	-0.05	0.225	0.595

Table 6.1: Predictions along with 90% probability bounds for a sequence of 5 actions applied to the mountain car. Columns x^1 and x^2 report the mean value of position and velocity of the car. Columns Bound x^1 and Bound x^2 report the computed the interval around x^1 and x^2 containing at least 90% of the trajectories.

We train the GP model on a dataset of 500 random actions applied to the mountain car. Now, for a proposed sequence of actions, we can bound the predicted trajectories, using our method with $\epsilon = 0.1$ (that is bound with 90% probability). Results from a typical run are presented in Table 6.1. Drawing 1000 trajectories from the mountain car system we verify that empirically more than 90% (91.6%) of them stay within the bounded area around the predictions obtained by our bound.

6.4.3 Closed-loop control of linear and quadratic systems

Here we use the proposed method to predict the closed-loop behaviour of several dynamical systems for a proposed feedback controller. The systems are either linear, or linear with an added quadratic term, of the general form:

$$\dot{x}^i = \boldsymbol{A}^i \boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{Q}^i \boldsymbol{x} + \boldsymbol{B}^i \boldsymbol{u}, \tag{6.8}$$

where x^i is the i-th component of the state vector $\boldsymbol{x}$. We assume a dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{y}_i\}$ of transitions is provided, where $\boldsymbol{y}_t = \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and

	System 1		System 2		System 3		System 4		System 5		
t/Bounds for:	x, W=0	x, W=-0.2	x^1	x^2	x^1	x^2	x^1	x^2	x^1	x^2	x^3
t=1	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165
t=2	0.170	0.165	0.161	0.161	0.162	0.161	0.143	0.159	0.165	0.161	0.165
t=3	0.174	0.164	0.157	0.158	0.160	0.158	0.042	0.153	0.165	0.155	0.165
t=4	0.178	0.164	0.153	0.154	0.158	0.155	0.009	0.147	0.162	0.153	0.165
t=5	0.182	0.163	0.149	0.151	0.157	0.152	0.005	0.141	0.159	0.152	0.165
t=6	0.186	0.163	0.145	0.148	0.154	0.150	0.005	0.134	0.157	0.147	0.165
Viol. ratio	0.073	0.090	0.0841		0.0957		0.0347		0.0659		

Table 6.2: Calculated bounds for different systems over an episode with 5 transitions. As "Viol. ratio", violations ration, we denote the fraction of transitions for which the bounds (calculated with a tolerance $\epsilon = 0.10$) were violated out of the 1000 sampled trajectories for each system.

a candidate controller $\boldsymbol{C}$. We train the GP model on 300 data points, and the bounds are calculated with $\epsilon = 0.1$ (90% probability bounds). The controller is either linear, or linear squashed by a sine function, as in PILCO [34] (see Equations (2.27), (2.28)). The reference point is the origin and the starting region is a hypercube around the origin with size 0.1650 for each dimension. In this setting the mean of the predicted states for the system is of secondary importance (in the linear case it is trivially zero) and our interest is focused on the width of the bounds on the prediction error. Shrinking bounds can be interpreted as similar to a probabilistic notion of stability for the GP model: shrinking bounds indicate that with the current controller and initial conditions, the model, with high probability, will stay in a (shrinking) region around the origin.

For each scenario, once the data and candidate controller is provided we:

- Train a GP model on the provided dataset.
- Assuming that the model is accurate, use the presented method to make bounded iterative predictions
- Statistically verify that the bounds are valid by sampling trajectories from the real system (verifying both that the learned model is accurate

enough, and that the predicted bounds quantify uncertainty correctly).

All results are presented in Table 6.2.

System 1, 1-dimensional state space, 1 control input, linear

In this simple case, we start with a linear, one-dimensional system with one control input. The parameters take the following values $A = 0.05, Q = 0, B = 1.0$. We use a linear controller for this case, so $u = Wx$. For the system to be asymptotically stable, we need $A + BW < 0 \Leftrightarrow W < -A$. We estimate the bounds with *no control*, $W = 0$, and for a controller that stabilises the system, $W = -0.2$. In the first case the bounds *must* be getting wider (since our bounds are conservative), while in the second, the bounds should be getting narrower around the origin but that's not guaranteed. Results show that without a controller the bounds indeed get wider, while with the controller the bounds get narrower.

System 2, 2-dimensional, 1 control input, linear

Here we make bounded predictions for a linear system with 2 dimensions and a single control input. This is only incrementally harder than the previous example, since the two dimensions have independent dynamics and the controller stabilises the first dimension only while the second dimension has inherently convergent dynamics. The system parameters:

$$\boldsymbol{A} = \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & -0.4 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix}, \boldsymbol{W} = \begin{bmatrix} -0.6 & 0.0 \end{bmatrix}.$$

The bounds on both dimensions contract with time.

System 3, 2-dimensional, 2 control inputs, linear

Next we work with a system that's still 2-dimensional with state variables that are not independent, but two control inputs available. The system parameters:

$$\boldsymbol{A} = \begin{bmatrix} 0.1 & 0.08 \\ -0.05 & 0.15 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 1.0 & 0 \\ 0.0 & 1.0 \end{bmatrix}, \boldsymbol{W} = \begin{bmatrix} -0.4 & 0.0 \\ 0.0 & -0.5 \end{bmatrix}.$$

As shown in Table 6.2 the bounds contract in this case too.

System 4, 2-dimensional, 1 control input, quadratic dynamics, controller from PILCO

Here we train a linear controller squashed by a sine function (effectively bounding the control inputs between -1 and 1) with PILCO [34] and then we calculate the bounds for the resulting system.

$$\boldsymbol{A} = \begin{bmatrix} -0.2 & 0.05 \\ -0.05 & -0.4 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix}, \boldsymbol{Q}^1 = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix},$$

$$\boldsymbol{Q}^2 = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 0.2 \end{bmatrix}, \boldsymbol{W} = \begin{bmatrix} -8.61 & -0.02 \end{bmatrix}.$$

The estimated bounds verify convergence.

System 5, 3-dimensional system, 2 control inputs, linear

In this example the system is linear and has 3 dimensions and 2 control inputs. Its parameters are:

$$\boldsymbol{A} = \begin{bmatrix} -0.2 & 0.0 & -0.0 \\ 0.0 & -0.3 & 0.0 \\ 0.0 & 0.0 & -0.6 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \\ 0.0 & 0.0 \end{bmatrix},$$

$$\boldsymbol{W} = \begin{bmatrix} -0.4 & 0.0 & 0.0 \\ 0.0 & -0.2 & 0 \end{bmatrix}.$$

Notice that for the third state variable, even though the system is contractive (by inspecting $\boldsymbol{A}$), the bound does not contract (it coincidentally stays constant).

Overall the results indicate that the bounds can correctly identify contractive behaviour due to the controller.

6.5 Conclusions

In this chapter, we derived a new formal probabilistic bound for iterated predictions with a GP model, without control, in open-loop and in closed-loop scenarios. Our approach does not make any further assumptions on the properties of the GP, other than knowledge of the kernel hyperparameters, learnt through maximum marginal likelihood, and every intermediate quantity used is calculated directly from the data. The experimental results show that our method is able to correctly propagate uncertainty even when existing heuristic approaches fail. Furthermore, they showcase how our method can be used to certify the safety of proposed controllers using GP models.

6.6 Proofs

Proof of Theorem 1 First we prove the following Lemma:

Lemma 1 *Let $\mathbf{f}(\boldsymbol{x})$ be a stochastic process. Consider measurable sets $\mathcal{A}$ and $\mathcal{B}$. Then, it holds that*

$$P(\mathbf{f}(\boldsymbol{y}) \in \mathcal{A} | \boldsymbol{y} \in \mathcal{B}) \leq P(\sup_{\boldsymbol{y} \in \mathcal{B}} \mathbf{f}(\boldsymbol{y}) \in \mathcal{A}).$$

Proof 2 *For the proof we use g as the joint probability distribution of $\mathbf{f}(y)$ and y.*

$$\begin{aligned}
&P(\mathbf{f}(y) = \mathcal{A} | y \in \mathcal{B}) = \\
&\frac{\int_{\mathcal{A}} \int_{\mathcal{B}} g(\mathbf{f}(y = x, y = \bar{y}) dx d\bar{y}}{P(y \in \mathcal{B})} = \\
&\frac{\int_{\mathcal{A}} \int_{\mathcal{B}} g(\mathbf{f}(\bar{y}) = x | y = \bar{y}) g(y = \bar{y}) dx d\bar{y}}{P(y \in \mathcal{B})} \leq \\
&\frac{\int_{\mathcal{A}} \int_{\mathcal{B}} \sup_{\hat{y} \in B} g(\mathbf{f}(\hat{y}) = x | y = \bar{y}) dx d\bar{y}}{P(y \in \mathcal{B})} = \\
&\int_{\mathcal{A}} \sup_{\hat{y} \in \mathcal{B}} g(\mathbf{f}(\hat{y}) = x) dx \frac{\int_{\mathcal{B}} g((y = \bar{y}))}{P(y \in \mathcal{B})} = \\
&\int_{\mathcal{A}} \sup_{\hat{y} \in \mathcal{B}} g(\mathbf{f}(\hat{y}) = x) dx = \\
&P(\sup_{y \in \mathcal{B}} \mathbf{f}(y) \in \mathcal{A})
\end{aligned}$$

Now the following calculations follow

$$
\begin{aligned}
& P(\mathrm{e}_{t+1} > K_{t+1}) \\
& \text{(By Definition of } \mathrm{e}_t\text{)} \\
=& P(|\boldsymbol{g}(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t) - \mathbf{f}(\mathbf{x}_t, \boldsymbol{u}_t)|_1 > K_{t+1}) \\
& \text{(By Marginalising with the events } \mathrm{e}_t > K_t\text{, } \mathrm{e}_t \leq K_t\text{)} \\
\leq& P(|\boldsymbol{g}(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t) - \mathbf{f}(\mathbf{x}_t, \boldsymbol{u}_t)|_1 > K_{t+1} \,|\, \mathrm{e}_t \leq K_t) P(\mathrm{e}_t \leq K_t) \\
& \quad + P(\mathrm{e}_t > K_t) \\
& \text{(By Lemma 1)} \\
\leq& P(\sup_{\boldsymbol{x} \in I_{\hat{\mathbf{x}}_t}^{K_t}} |\boldsymbol{g}(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t) - \mathbf{f}(\mathbf{x}, \boldsymbol{u}_t)|_1 > K_{t+1}) P(\mathrm{e}_t \leq K_t) \\
& \quad + P(\mathrm{e}_t > K_t) \\
& \text{(By the fact that } P(\mathrm{e}_t \leq K_t) = 1 - P(\mathrm{e}_t > K_t)\text{)} \\
=& P(\sup_{\boldsymbol{x} \in I_{\hat{\mathbf{x}}_t}^{K_t}} |\boldsymbol{g}(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t) - \mathbf{f}(\mathbf{x}, \boldsymbol{u}_t)|_1 > K_{t+1})(1 - P(\mathrm{e}_t > K_t)) \\
& + P(\mathrm{e}_t > K_t).
\end{aligned}
$$

7

Discussion

7.1 Summary

The sharp increase in interest in reinforcement learning in recent years has led to an ever increasing number of algorithms and approaches, with significant quantitative performance gains in comparison with the previous state of the art. The majority of these algorithms operate in simulated domains, where massive datasets can be amassed at very low cost, the only restriction being the available computational resources. Even the methods that have physical components (such as robotic agents) are trained and evaluated, for the most part, in controlled laboratory environments. Moving these algorithms from their current environments closer to the real world requires direct addressing of the issues of safety and the associated implications of data scarcity.

Furthermore, many of the newer reinforcement learning approaches, especially based upon deep-RL, although capable of achieving impressive performance, have been shown to be surprisingly brittle [64, 66, 38]. As a growing

body of literature indicates, performance is largely dependent upon hyperparameter selection, can vary significantly from run to run (changing exclusively the random seed used) and is sensitive to seemingly unimportant implementation details. These issues, beyond simply rendering conclusive empirical evaluation harder, as exhaustive hyperparameter searches are necessary and evaluation of computationally taxing experiments have to be repeated a large number of times to reduce the uncertainty of the observed performance, have led to questioning commonly accepted interpretations of recent results [66, 38].

The community has addressed this challenge, in part, by favouring more comprehensive experiments [35, 153]. This approach, although definitely valuable, is only empirical and, to achieve the required reliability for safety critical applications [100], extremely large (perhaps unrealistically large) datasets would be needed. Thus, if we are to accept the ever-increasing role of intelligent algorithms in society, industry and commerce, algorithm safety, trust and verifiability are prerequisites and need to be addressed directly.

Throughout this thesis we have focused on providing degrees of guaranteed safety in model-based policy search. We trained models to capture the dynamics of unknown systems and used the subsequent model predictions to evaluate, improve and verify the safety of proposed parametric policies, defining safety in terms of the probability of satisfying predefined constraints. We developed an open source software tool for this purpose, and made it available for the wider research community. Finally, we provided an alternative approach for an integral component of the process that verifies the safety of a candidate policy: uncertainty tracking, over multiple time steps, for Gaussian process models.

We hypothesised and verified experimentally that a policy-search method (or indeed a reinforcement learning method in general) that is exclusively

focused on performance cannot be naively assumed to discover safe policies based on a generic reward function. We consider such a requirement to be satisfied only by tackling the problem of meticulous reward function design, or by encoding the relevant safety specifications in an other manner. Our methodology allows for such state-space safety constraints to be clearly stated and enforced. We consider this an intuitive encoding of prior knowledge, that covers a wide selection of probable scenarios, and showcase its use in several problem domains.

In Chapter 4, we presented our new implementation of the PILCO framework. We outlined its main components and their functionalities, and highlighted the advantages of the new Python implementation in comparison with the existing Matlab version. We showed experimentally that it has the flexibility necessary to solve popular RL benchmark problems, not designed specifically with PILCO in mind, while maintaining data-efficiency.

In Chapter 5, we augmented the PILCO framework with safety constraints. The constraint formulation preserves desired properties of the framework, such as efficient evaluation of the proposed policies. By adaptively tuning the objective function, we effectively replace the need for hand tuned reward functions, and by verifying probabilistically that a policy complies with the imposed constraints before deployment, we reduce the number of constraint violations, with minimal effect on performance.

In Chapter 6, we provided formal probabilistic bounds for the predictive uncertainty of a Gaussian process model, over multiple iterative predictions. While moment matching can be sufficient for systems without strong nonlinearities, or in scenarios where some underestimation of uncertainty is acceptable, other problems call for a more conservative approach. For such scenarios, our new method provides stronger theoretical results and formal guar-

antees.

The main challenges we set out to tackle with our work are related to the impediments of more widespread use of RL algorithms: the need for extensive datasets, and the high risks associated with training and deploying RL agents without taking safety into account. Overall we developed methods and proposed solutions that make clear contributions towards this direction. Our software tool provides a straightforward and accessible way for experimentation with model based RL methods, without the need for the resources neural network-based methods often require. By embedding safety constraints in the PILCO framework, we demonstrated how probabilistic models, with very limited prior knowledge, can naturally address safety considerations. With our final contribution, we addressed the often overlooked issue of possibly inaccurate uncertainty propagation, for a popular class of models. Of course, the applicability of our methods is not universal, as we made significant assumptions on the characteristics of the tasks we take on. Additionally, the emphasis we put on data efficiency and safety can potentially put our work at a disadvantage, in cases where data are abundant, or high dimensional environments (which typically require large datasets, independent of the algorithm used). We outline research directions with the potential of addressing these limitations in the next and final section of this dissertation.

7.2 Future Work

We conclude the thesis by highlighting some possible extensions to the work we presented thus far.

Firstly, our work was based exclusively on Gaussian processes as the model-building engine. While the probabilistic nature of GPs offers important ad-

vantages, ensembles of neural networks [27] as well as neural networks using dropout [45, 46] have been shown to be able to approximate these probabilistic properties. These models can combine uncertainty quantification with scalability, which is of crucial importance for complex domains. Furthermore, investigating alternatives to the kernel, namely a global, RBF-kernel, might allow the modelling of systems without the smooth, differentiable dynamics we have mostly focused on.

A crucial issue for the wider adoption of the proposed methods is tackling problems with higher dimensional state and action spaces. There are multiple research directions that could offer valuable results in this effort. Incorporating prior knowledge in the modelling process, with an approximate model used in conjunction with a learnt one, is a viable option. Informative priors on the hyperparameters of a GP is another option, although intuition about a system's properties is often hard to encode in such priors. Using multiple local models, instead of a global one, has also been explored in the literature with success ([127, 84]). Local models are commonly employed in online frameworks similar to MPC, as they are typically simpler than global models, which in turn allows for efficient online computation.

Moreover, adapting our methods to partially observable settings represents an important challenge with high practical significance, as in a large part of relevant problems (in robotics for example), we cannot directly observe all relevant state variables. The closest work to ours in the partially observable domain has been presented by McAllister and Rasmussen [94], and a unifying framework that takes into account both safety concerns and partial observability, although challenging, should be possible. Another work in this area, Eleftheriadis et al. [37], increases the modelling power of PILCO, but does not tackle policy optimisation. Further research in this direction, investigating

which existing policy optimisation schemes are suitable for such models, and what adaptations or expansions would be needed, is clearly promising.

Finally, an important future challenge would be applying our method to real-world systems, with all the additional complexity that comes with moving away from simulation. Moving to robotic applications for example, would require dealing with noisy sensors and unpredictable disturbances. This raises practical difficulties (e.g. communicating with on-board sensors), which, despite not insurmountable, are non-trivial indeed. Nevertheless, the combination of data efficiency and safety awareness renders the method well-suited for physical systems.

Bibliography

[1] Alessandro Abate. "Formal verification of complex systems: model-based and data-driven methods". In: *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*. ACM. 2017, pp. 91–93.

[2] Alessandro Abate, Henk Blom, Nathalie Cauchi, Sofie Haesaert, Arnd Hartmanns, Kendra Lesser, Meeko Oishi, Vignesh Sivaramakrishnan, Sadegh Soudjani, Cristian Ioan Vasile, et al. "ARCH-COMP18 Category Report: Stochastic Modelling." In: *ARCH@ ADHS*. 2018, pp. 71–103.

[3] David Abel, John Salvatier, Andreas Stuhlmüller, and Owain Evans. "Agent-Agnostic Human-in-the-Loop Reinforcement Learning". In: *CoRR* abs/1701.04079 (2017). URL: http://arxiv.org/abs/1701.04079.

[4] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. "Constrained Policy Optimization". In: *International Conference on Machine Learning*. 2017, pp. 22–31.

[5] Robert J Adler and Jonathan E Taylor. *Random fields and geometry*. Springer Science & Business Media, 2009.

[6] Anayo K Akametalu, Shahab Kaynama, Jaime F Fisac, Melanie N Zeilinger, Jeremy H Gillula, and Claire J Tomlin. "Reachability-based safe learning with Gaussian processes". In: *IEEE 53rd Annual Conference on Decision and Control (CDC), 2014: 15-17 Dec. 2014, Los Angeles, California, USA*. IEEE. 2014, pp. 1424–1431.

[7] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. "Concrete problems in AI safety". In: *arXiv preprint arXiv:1606.06565* (2016).

[8] Dario Amodei et al. "Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin". In: ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 173–182. URL: http://proceedings.mlr.press/v48/amodei16.html.

[9] Olov Andersson, Fredrik Heintz, and Patrick Doherty. "Model-Based Reinforcement Learning in Continuous Environments Using Real-Time Constrained Optimization." In: *AAAI*. 2015, pp. 2497–2503.

[10] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[11] Karl J Astrom. "Optimal control of Markov processes with incomplete state information". In: *Journal of mathematical analysis and applications* 10.1 (1965), pp. 174–205.

[12] Brian Axelrod, Leslie Pack Kaelbling, and Tomas Lozano-Perez. "Provably Safe Robot Navigation with Obstacle Uncertainty". In: *Robotics: Science and Systems (RSS)*. 2017. URL: http://lis.csail.mit.edu/pubs/axelrod-rss-17.pdf.

[13] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. 2008.

[14] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. "Unifying count-based exploration and intrinsic motivation". In: *Advances in neural information processing systems*. 2016, pp. 1471–1479.

[15] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[16] Felix Berkenkamp, Andreas Krause, and Angela P. Schoellig. "Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics". In: *CoRR* abs/1602.04450 (2016). URL: `http://arxiv.org/abs/1602.04450`.

[17] Felix Berkenkamp and Angela P. Schoellig. "Learning-based robust control: guaranteeing stability while improving performance". In: *Proc. of the Workshop on Machine Learning in Planning and Control of Robot Motion, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2014.

[18] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[19] Arno Blaas, Luca Laurenti, Andrea Patane, Luca Cardelli, Marta Kwiatkowska, and Stephen Roberts. "Robustness Quantification for Classification with Gaussian Processes". In: *arXiv preprint arXiv:1905.11876* (2019).

[20] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[21] Jan-P. Calliess, Michael A. Osborne, and Stephen J. Roberts. "Conservative collision prediction and avoidance for stochastic trajectories in continuous time and space". In: *AAMAS*. 2014.

[22] Joaquin Quiñonero Candela, Agathe Girard, Jan Larsen, and Carl Edward Rasmussen. "Propagation of uncertainty in Bayesian kernel models-application to multiple-step ahead forecasting". In: *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03)*. Vol. 2. IEEE. 2003, pp. II–701.

[23] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, and Andrea Patane. "Robustness guarantees for Bayesian inference with Gaussian processes". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 7759–7768.

[24] Nathalie Cauchi and Alessandro Abate. "Benchmarks for cyber-physical systems: A modular model library for building automation systems (Extended version)". In: *arXiv preprint arXiv:1803.06315* (2018).

[25] Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, and Jean-Baptiste Mouret. "Black-box data-efficient policy search for robotics". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 51–58.

[26] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. "End-to-end safe reinforcement learning through barrier functions for

safety-critical continuous control tasks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 3387–3395.

[27] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Advances in Neural Information Processing Systems*. 2018, pp. 4754–4765.

[28] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. "Model-based reinforcement learning via meta-policy optimization". In: *arXiv preprint arXiv:1809.05214* (2018).

[29] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. "Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 1. 2019, pp. 128–136.

[30] Marc Peter Deisenroth. "Efficient reinforcement learning using Gaussian processes". PhD thesis. Karlsruhe Institute of Technology, 2010.

[31] Marc Peter Deisenroth, Peter Englert, Jan Peters, and Dieter Fox. "Multi-task policy search for robotics". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3876–3881.

[32] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. "A survey on policy search for robotics". In: *Foundations and Trends® in Robotics* 2.1–2 (2013), pp. 1–142.

[33] Marc Peter Deisenroth, Carl E. Rasmussen, and Dieter Fox. "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning". In: *Robotics: Science and Systems*. 2011.

[34] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *In Proceedings of the International Conference on Machine Learning*. 2011.

[35] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. "Benchmarking deep reinforcement learning for continuous control". In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.

[36] David Duvenaud. "Automatic model construction with Gaussian processes". PhD thesis. University of Cambridge, 2014.

[37] Stefanos Eleftheriadis, Tom Nicholson, Marc Deisenroth, and James Hensman. "Identification of Gaussian process state space models". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5309–5319.

[38] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. "Implementation Matters in Deep RL: A Case Study on PPO and TRPO". In: *International Conference on Learning Representations*. 2019.

[39] Tom Everitt. "Towards safe artificial general intelligence". PhD thesis. The Australian National University, 2018.

[40] G Farquhar, T Rocktäschel, M Igl, and S Whiteson. "TreeqN and ATreEC: Differentiable tree-structured models for deep reinforcement learning". In: *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*. Vol. 6. ICLR. 2018.

[41] AA Feldbaum. "Dual control theory. I". In: *Avtomatika i Telemekhanika* 21.9 (1960), pp. 1240–1249.

[42] Jaime F Fisac, Anayo K Akametalu, Melanie N Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J Tomlin. "A general safety framework for learning-based control in uncertain robotic systems". In: *IEEE Transactions on Automatic Control* 64.7 (2018), pp. 2737–2752.

[43] Vincent François-Lavet, Yoshua Bengio, Doina Precup, and Joelle Pineau. "Combined reinforcement learning via abstract representations". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 3582–3589.

[44] Roger Frigola, Yutian Chen, and Carl E. Rasmussen. "Variational Gaussian Process State-Space Models". In: *CoRR* abs/1406.4905 (2014).

[45] Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. 2016, pp. 1050–1059.

[46] Yarin Gal, Rowan Thomas McAllister, and Carl Edward Rasmussen. "Improving PILCO with Bayesian Neural Network Dynamics Models". In: *Data-Efficient Machine Learning workshop*. Vol. 951. 2016, p. 2016.

[47] Chris Gamble and Jim Gao. *Safety-first AI for autonomous data centre cooling and industrial control*. `https://deepmind.com/blog/safety-first-ai-autonomous-data-centre-cooling-and-industrial-control/`. Accessed: 2019-07-4. 2018.

[48] Javier García and Fernando Fernández. "A Comprehensive Survey on Safe Reinforcement Learning". In: *Journal of Machine Learning Research* 16 (2015), pp. 1437–1480. URL: `http://jmlr.org/papers/v16/garcia15a.html`.

[49] Alan Genz. "Numerical computation of multivariate normal probabilities". In: *Journal of computational and graphical statistics* 1.2 (1992), pp. 141–149.

[50] Jeremy H Gillula and Claire J Tomlin. "Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 2723–2730.

[51] Jeremy H Gillula and Claire J Tomlin. "Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 2723–2730.

[52] Agathe Girard, Carl Edward Rasmussen, Joaquin Quiñonero Candela, and Roderick Murray-Smith. "Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting". In: *Advances in neural information processing systems*. 2003, pp. 545–552.

[53] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[54] Abhijit Gosavi. "Reinforcement learning for model building and variance-penalized control". In: *Proceedings of the 2009 Winter Simulation Conference (WSC)*. IEEE. 2009, pp. 373–379.

[55] Michael Green and David JN Limebeer. *Linear robust control*. Courier Corporation, 2012.

[56] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. "Continuous deep q-learning with model-based acceleration". In: *International Conference on Machine Learning*. 2016, pp. 2829–2838.

[57] David Ha and Jürgen Schmidhuber. "World models". In: *arXiv preprint arXiv:1803.10122* (2018).

[58] Sofie Haesaert, Paul M. J. Van den Hof, and Alessandro Abate. "Data-driven and Model-based Verification via Bayesian Identification and Reachability Analysis". In: *CoRR* (2015).

[59] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. "Logically-constrained reinforcement learning". In: *arXiv preprint arXiv:1801.08099* (2018).

[60] Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J Pappas, and Insup Lee. "Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE. 2019, pp. 5338–5343.

[61] Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. "Deep Reinforcement Learning with Temporal Logics". In: *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer. 2020, pp. 1–22.

[62] Verena Heidrich-Meisner and Christian Igel. "Evolution strategies for direct policy search". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2008, pp. 428–437.

[63] David Held, Zoe McCarthy, Michael Zhang, Fred Shentu, and Pieter Abbeel. "Probabilistically Safe Policy Transfer". In: *arXiv preprint arXiv:1705.05394* (2017).

[64] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. "Deep reinforcement learning that matters". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[65] Nathan Hunt, Nathan Fulton, Sara Magliacane, Nghia Hoang, Subhro Das, and Armando Solar-Lezama. "Verifiably safe exploration for end-to-end reinforcement learning". In: *arXiv preprint arXiv:2007.01223* (2020).

[66] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. "A Closer Look at Deep Policy Gradients". In: *International Conference on Learning Representations*. 2019.

[67] Craig Innes and Subramanian Ramamoorthy. "Elaborating on learned demonstrations with temporal logic specifications". In: *arXiv preprint arXiv:2002.00784* (2020).

[68] Lucas Janson, Edward Schmerling, and Marco Pavone. "Monte Carlo motion planning for robot trajectory optimization under uncertainty". In: *Robotics Research*. Springer, 2018, pp. 343–361.

[69] Sebastian Junges, Nils Jansen, Christian Dehnert, Ufuk Topcu, and Joost-Pieter Katoen. "Safety-Constrained Reinforcement Learning for MDPs". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Marsha Chechik and Jean-François Raskin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 130–146. ISBN: 978-3-662-49674-9.

[70] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[71] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. "Uncertainty-Aware Reinforcement Learning for Collision Avoidance". In: abs/1702.01182 (2017). arXiv: 1702.01182. URL: http://arxiv.org/abs/1702.01182.

[72] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. "Model Based Reinforcement Learning for Atari". In: *International Conference on Learning Representations*. 2019.

[73] Sham Kakade. "A natural policy gradient". In: *Advances in neural information processing systems* 14 (2001), pp. 1531–1538.

[74] Sanket Kamthe and Marc Peter Deisenroth. "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control". In: *arXiv preprint arXiv:1706.06491* (2017).

[75] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1* (*Long and Short Papers*). 2019, pp. 4171–4186.

[76] Ivan Kiskin and Alessandro Abate. *Verification of Gaussian Process Dynamical Systems*. online. 2016.

[77] Jens Kober and Jan R Peters. "Policy search for motor primitives in robotics". In: *Advances in neural information processing systems*. 2009, pp. 849–856.

[78] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. "Learning-based Model Predictive Control for Safe Exploration and Reinforcement Learning". In: *CoRR* abs/1803.08287 (2018). arXiv: 1803.08287. URL: http://arxiv.org/abs/1803.08287.

[79] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[80] Andras Gabor Kupcsik, Marc Peter Deisenroth, Jan Peters, and Gerhard Neumann. "Data-efficient generalization of robot skills with contextual policy search". In: *Twenty-Seventh AAAI Conference on Artificial Intelligence*. 2013.

[81] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. *Building machines that learn and think like people*. Tech. rep. Center for Brains, Minds and Machines (CBMM), arXiv, 2016.

[82] Neil D Lawrence. "Gaussian process latent variable models for visualisation of high dimensional data". In: *Advances in neural information processing systems*. 2004, pp. 329–336.

[83] Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. "Sparse spectrum Gaussian process regression". In: *The Journal of Machine Learning Research* 11 (2010), pp. 1865–1881.

[84] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[85] Pu Li, Harvey Arellano-Garcia, and Günter Wozny. "Chance constrained programming approach to process optimization under uncertainty". In: *Computers & Chemical Engineering* 32.1-2 (2008), pp. 25–45.

[86] Zhaojian Li, Uroš Kalabić, and Tianshu Chu. "Safe reinforcement learning: Learning with supervision using a constraint-admissible set". In: *2018 Annual American Control Conference (ACC)*. IEEE. 2018, pp. 6390–6395.

[87] Dong C Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* 45.1-3 (1989), pp. 503–528.

[88] Jan Marian Maciejowski. *Predictive control: with constraints*. 2002.

[89] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[90] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[91] Maja J Mataric. "Reward functions for accelerated learning". In: *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 181–189.

[92] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. "Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning". In: *International Conference on Artificial Neural Networks*. Springer. 2006, pp. 840–849.

[93] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. "GPflow: A Gaussian process library using TensorFlow". In: *Journal of Machine Learning Research* 18.40 (Apr. 2017), pp. 1–6. URL: http://jmlr.org/papers/v18/16-537.html.

[94] Rowan McAllister and Carl Edward Rasmussen. "Data-Efficient Reinforcement Learning in Continuous State-Action Gaussian-POMDPs". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 2040–2049.

[95] Rowan McAllister and Carl Edward Rasmussen. "Data-efficient reinforcement learning in continuous-state POMDPs". In: *arXiv preprint arXiv:1602.02523* (2016).

[96] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*. 2016, pp. 1928–1937.

[97] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[98] Teodor Mihai Moldovan and Pieter Abbeel. "Safe exploration in Markov decision processes". In: *Proceedings of the 29th International*

Coference on International Conference on Machine Learning. 2012, pp. 1451–1458.

[99] Andrew William Moore. "Efficient Memory-based Learning for Robot Control". PhD thesis. University of Cambridge, 1990.

[100] Richard Murray. *Can We Really Use Machine Learning in Safety Critical Systems*. Intersections between Control, Learning and Optimization 2020. 2020. URL: https://www.youtube.com/watch?v=Wi8Y---ce28.

[101] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7559–7566.

[102] Andrew Y Ng and Michael I Jordan. "Shaping and policy search in reinforcement learning". PhD thesis. University of California, Berkeley Berkeley, 2003.

[103] P. Ngatchou, A. Zarei, and A. El-Sharkawi. "Pareto Multi Objective Optimization". In: *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*. 2005, pp. 84–91. DOI: 10.1109/ISAP.2005.1599245.

[104] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[105] Junhyuk Oh, Satinder Singh, and Honglak Lee. "Value prediction network". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6118–6128.

[106] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. "An algorithmic perspective on imitation learning". In: *arXiv preprint arXiv:1811.06711* (2018).

[107] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvari, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado Van Hasselt. "Behaviour Suite for Reinforcement Learning". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rygf-kSYwH.

[108] Michael A Osborne, Roman Garnett, and Stephen J Roberts. "Gaussian processes for global optimization". In: vol. 2009. 2009.

[109] Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. "Learning model-based planning from scratch". In: *arXiv preprint arXiv:1707.06170* (2017).

[110] Jan Peters, Katharina Mülling, and Yasemin Altun. "Relative entropy policy search." In: *AAAI*. Vol. 10. Atlanta. 2010, pp. 1607–1612.

[111] Jan Peters and Stefan Schaal. "Policy gradient methods for robotics". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2006, pp. 2219–2225.

[112] KB Petersen, MS Pedersen, et al. "The Matrix Cookbook, vol. 7". In: *Technical University of Denmark* 15 (2008).

[113] Kyriakos Polymenakos, Alessandro Abate, and Stephen Roberts. "Safe Policy Search Using Gaussian Process Models". In: *Proceedings of the*

18th International Conference on Autonomous Agents and Multi Agent Systems. IFAAMS. 2019, pp. 1565–1573.

[114] Kyriakos Polymenakos, Alessandro Abate, and Stephen Roberts. *Safe Policy Search with Gaussian Process Models*. 2019. arXiv: 1712.05556.

[115] Kyriakos Polymenakos, Luca Laurenti, Andrea Patane, Luca Cardelli, Marta Kwiatkowska, Alessandro Abate, and Stephen Roberts. "Safety guarantees for iterative predictions with Gaussian Processes". In: *Proceedings of the IEEE Conference on Decision and Control*. IEEE Xplore. 2020.

[116] Kyriakos Polymenakos, Nikitas Rontsis, Alessandro Abate, and Stephen Roberts. "SafePILCO: a software tool for safe and data-efficient policy synthesis". In: *International Conference on Quantitative Evaluation of Systems*. Springer. 2020.

[117] Joaquin Quiñonero-Candela, Agathe Girard, Jan Larsen, and Carl Edward Rasmussen. "Propagation of uncertainty in Bayesian kernel models-application to multiple-step ahead forecasting". In: *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*. Vol. 2. IEEE. 2003, pp. II–701.

[118] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. 2006.

[119] Stephen Roberts, Michael Osborne, Mark Ebden, Steven Reece, Neale Gibson, and Suzanne Aigrain. "Gaussian processes for time-series modelling". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 371.1984 (2013), p. 20110550.

[120] Diederik M Roijers and Shimon Whiteson. "Multi-objective decision making". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 11.1 (2017), pp. 1–129.

[121] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. "Online planning algorithms for POMDPs". In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 663–704.

[122] Sam Roweis and Zoubin Ghahramani. "A unifying review of linear Gaussian models". In: *Neural computation* 11.2 (1999), pp. 305–345.

[123] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (2017).

[124] Makoto Sato, Hajime Kimura, and Shibenobu Kobayashi. "TD algorithm for the variance of return and mean-variance reinforcement learning". In: *Transactions of the Japanese Society for Artificial Intelligence* 16.3 (2001), pp. 353–362.

[125] William Saunders, Girish Sastry, Andreas Stuhlmueller, and Owain Evans. "Trial without Error: Towards Safe Reinforcement Learning via Human Intervention". In: *arXiv preprint arXiv:1707.05173* (2017).

[126] Stefan Schaal. "Is imitation learning the route to humanoid robots?" In: *Trends in cognitive sciences* 3.6 (1999), pp. 233–242.

[127] Stefan Schaal and Christopher G Atkeson. "Assessing the quality of learned local models". In: *Advances in neural information processing systems* (1994), pp. 160–160.

[128] Jeff G Schneider. "Exploiting model uncertainty estimates for safe dynamic control learning". In: *Advances in neural information processing systems*. 1997, pp. 1047–1053.

[129] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. "Mastering atari, go, chess and shogi by planning with a learned model". In: *arXiv preprint arXiv:1911.08265* (2019).

[130] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015, pp. 1889–1897.

[131] Guy Shani, Joelle Pineau, and Robert Kaplow. "A survey of point-based POMDP solvers". In: *Autonomous Agents and Multi-Agent Systems* 27.1 (2013), pp. 1–51.

[132] Olivier Sigaud and Freek Stulp. "Policy search in continuous action domains: an overview". In: *Neural Networks* 113 (2019), pp. 28–40.

[133] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[134] Edward Snelson and Zoubin Ghahramani. "Sparse Gaussian processes using pseudo-inputs". In: *Advances in neural information processing systems*. 2006, pp. 1257–1264.

[135] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[136] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias W Seeger. "Information-theoretic regret bounds for gaussian process optimization in the bandit setting". In: *IEEE Transactions on Information Theory* 58.5 (2012), pp. 3250–3265.

[137] Yanan Sui, Alkis Gotovos, Joel Burdick, and Andreas Krause. "Safe Exploration for Optimization with Gaussian Processes". In: *Proceedings of The 32nd International Conference on Machine Learning*. 2015, pp. 997–1005.

[138] Wen Sun, Luis G Torres, Jur Van Den Berg, and Ron Alterovitz. "Safe motion planning for imprecise robotic manipulators by minimizing probability of collision". In: *Robotics Research*. Springer, 2016, pp. 685–701.

[139] Richard S Sutton. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Machine learning proceedings 1990*. Elsevier, 1990, pp. 216–224.

[140] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 1998.

[141] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[142] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. "# exploration: A study of count-based exploration for deep reinforcement learning". In: *Advances in neural information processing systems*. 2017, pp. 2753–2762.

[143] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. *DeepMind Control Suite*. Tech. rep. DeepMind, Jan. 2018. URL: https://arxiv.org/abs/1801.00690.

[144] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[145] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. "Safe exploration in finite Markov decision processes with Gaussian processes". In: *Advances in Neural Information Processing Systems*. 2016, pp. 4312–4320.

[146] Benjamin Van Niekerk, Andreas Damianou, and Benjamin S Rosman. "Online constrained model-based reinforcement learning". In: *Association for Uncertainty in Artificial Intelligence (AUAI)* (2017).

[147] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[148] J. Vinogradska, B. Bischoff, J. Achterhold, T. Koller, and J. Peters. "Numerical Quadrature for Probabilistic Policy Search". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* (2018).

[149] Julia Vinogradska, Bastian Bischoff, Duy Nguyen-Tuong, Anne Romer, Henner Schmidt, and Jan Peters. "Stability of controllers for Gaussian process forward models". In: *International Conference on Machine Learning*. 2016, pp. 545–554.

[150] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: https://doi.org/10.1038/s41592-019-0686-2.

[151] Tung-Long Vuong and Kenneth Tran. "Uncertainty-aware Model-based Policy Optimization". In: *arXiv preprint arXiv:1906.10717* (2019).

[152] J. M. Wang, D. J. Fleet, and A. Hertzmann. "Gaussian process dynamical models". In: *Proc. NIPS 2005*. 2005, pp. 1441–1448.

[153] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. *Benchmarking Model-Based Reinforcement Learning*. 2019. arXiv: 1907.02057 [cs.LG].

[154] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[155] Théophane Weber, Sébastien Racaniére, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdoménech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. "Imagination-Augmented Agents for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1707.06203* (2017).

[156] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Machine Learning* 8 (1992), pp. 229–256.

[157] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. “Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search”. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 528–535.