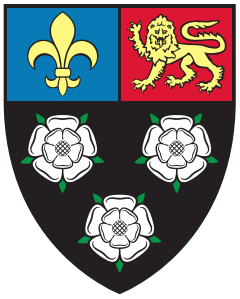
Bayesian Learning for Data-Efficient Control



Rowan McAllister

Supervisor: **Prof. C.E. Rasmussen**

Advisor: Prof. Z. Ghahramani

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

King's College

September 2016

I would like to dedicate this thesis to my loving family, Julie, Ian, Marion, and Emily.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Rowan McAllister
September 2016

Acknowledgements

Thank you to my supervisor Carl Rasmussen, who spent countless hours with me, patiently teaching me machine learning theory, exploring research ideas, asking the right questions, editing papers, and coding. I have enjoyed our discussions immensely and have learned so much from your supervision, in particular your enthusiasm to seek the 'right' objective or model. A source of constant amusement has been when you claim to not understand something, even though the rest of the lab treats you as the authority on the subject, simply because there is often no limit to how deep one can understand something. As a result, I've learned to think much deeper about why something works well (or poorly), to build richer intuitions behind observed phenomena in a model, rather than to claim I understand anything in its entirety. Carl was always there if I needed help and I couldn't of asked for better supervisor.

I also wish to thank my advisor Zoubin Ghahramani who helped me begin the PhD, get a great internship with autonomous vehicles at UberATC, and develop my writing style. My internship was made extra enjoyable by working closely with Jeff Schneider who was a fantastic leader and now I am hooked on working on autonomous vehicles after I hand in this thesis.

Thanks to my collaborators Yarin Gal and Mark van der Wilk. Mark van der Wilk helped derive estimates of Gaussian process variance reduction in § 4.5.1 including producing Fig. 4.4. Yarin Gal collaborated with me with equal effort on Chapter 5. Thank you to the older PhD students who have since moved on, Andrew McHutchon and Roger Frigola, for all the formative chats helping me hone my research ideas. I was lucky enough to work some bright masters students Aleksi Tukiainen, Martin Kukla, and Jonas Umlauft, who often helped with hardware and pair programming. To David MacKay, even though I only met you a few times, and wish I had known you better, your book 'Information Theory, Inference, and Learning Algorithms' was wonderful to read and shaped my understanding and love of Bayesian theory. Thank you to my examiners Marc Deisenroth and Richard Turner for your helpful feedback and improvements.

To Terrey and Anne Arcus, a big thank you for helping me financially through the University of Sydney's Arcus Travelling Scholarship. I couldn't of been here without your help, and I intend to pass the favour to another University of Sydney student in 20 years! Also thanks to the University of Sydney for the Eleanor Sophia Wood Postgraduate Research Travelling Scholarship and King's College Cambridge for the King's Studentship which completed my funding (which I will also pass forwards). I also wish to thank my previous supervisors Robert Fitch and Thierry Peynot who took me on as a masters student at the Australian Centre for Field Robotics at Sydney in preparation for the PhD, with whom I learned much about outdoor robotics research and first started using Bayesian techniques. Your help, along with Irena Koprinska helped prepare me for this PhD.

Thanks to my wonderful partner Charlotte who critiqued my boring practice talks and for her general support throughout the past 2.5 years. Finally, I wish to thank my family: mum, dad, Marion, and Emily; without your support none of this would have been possible.

ABSTRACT

Applications to learn control of unfamiliar dynamical systems with increasing autonomy are ubiquitous. From robotics, to finance, to industrial processing, autonomous learning helps obviate a heavy reliance on experts for system identification and controller design. Often real world systems are nonlinear, stochastic, and expensive to operate (e.g. slow, energy intensive, prone to wear and tear). Ideally therefore, nonlinear systems can be identified with minimal system interaction. This thesis considers *data efficient* autonomous learning of control of nonlinear, stochastic systems. Data efficient learning critically requires probabilistic modelling of dynamics. Traditional control approaches use deterministic models, which easily overfit data, especially small datasets. We use probabilistic Bayesian modelling to learn systems from scratch, similar to the PILCO algorithm, which achieved unprecedented data efficiency in learning control of several benchmarks. We extend PILCO in three principle ways. First, we learn control under significant observation noise by simulating a filtered control process using a tractably analytic framework of Gaussian distributions. In addition, we develop the 'latent variable belief Markov decision process' when filters must predict under real-time constraints. Second, we improve PILCO's data efficiency by directing exploration with predictive loss uncertainty and Bayesian optimisation, including a novel approximation to the Gittins index. Third, we take a step towards data efficient learning of high-dimensional control using Bayesian neural networks (BNN). Experimentally we show although filtering mitigates adverse effects of observation noise, much greater performance is achieved when optimising controllers with evaluations faithful to reality: by simulating closed-loop filtered control if executing closed-loop filtered control. Thus, controllers are optimised w.r.t. how they are used, outperforming filters applied to systems optimised by unfiltered simulations. We show directed exploration improves data efficiency. Lastly, we show BNN dynamics models are almost as data efficient as Gaussian process models. Results show data efficient learning of high-dimensional control is possible as BNNs scale to high-dimensional state inputs.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 CONTROL OF DYNAMICAL SYSTEMS

The world contains all sorts of dynamical systems, from vehicles, to financial markets, to electrical systems, many of which we wish to control in some way. Control is about continually deciding appropriate responses to a system's state, as that state changes, to influence the system towards desirable outcomes. Good decisions are not always obvious, having both short-term and long-term consequences. Sequential decisions making tasks are particularly complex, since decisions now affect the availability and consequence of decisions later.

A block diagram depicting the process of control is given Fig. 1.1. For concreteness we use the term 'robot' to collectively describe the system, controller, sensors and subjective beliefs (state estimation) modules seen in Fig. 1.1 (although control is by no means limited to physical, robotic systems). We start with the robot's estimate of current system state $\hat{x}$ (e.g. configuration of a robotic arm), from which a controller π decides on a control output u (e.g. the amount of voltage to apply the arm's actuators). The control decision u influences the system dynamics f, which changes the system's state x. As part of closed loop control, a sensor g makes a (possibly noisy) observation y (e.g. potentiometer readings of the arm's configuration) of the state x, with which the robot updates its subjective state estimate $\hat{x}$. The robot's goal is minimising cumulative cost, each cost representing the objective 'undesirability' of state x. For example, the cost could be the distance between arm's welding nozzle and the location to weld. Designing a good controller π for this task requires understanding the effects that voltages have on the robotic arm, and physical constraints on x or u. In this thesis, we consider control of nonlinear dynamical systems f with Gaussian sensor noise from g.

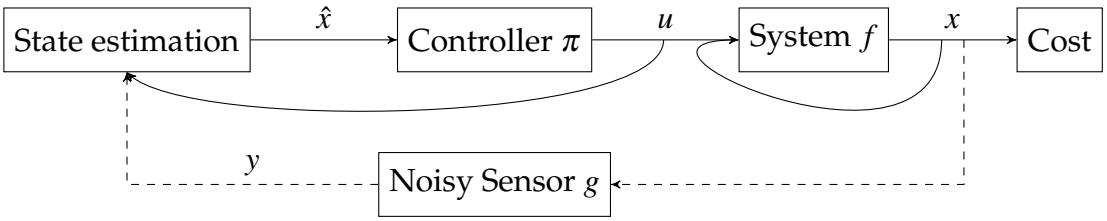


Fig. 1.1 ***A block diagram example of closed loop control of a dynamical system using feedback.*** *Open loop control corresponds to removing the feedback loop (dashed arrows). Without sensors, state estimation is dead-reckoned, with increasing error as time progresses.*

1.2 Data Efficient Learning of Control

'Big data' is a common technology buzzword to describe datasets so large that previously simple tasks of transfer, storage, query, and analysis of data pose challenging problems. Company executives, journalists, and consultants often use the phrase 'big data', no doubt to impress each other with their grand visions of detailed consumer datasets in 21st century commerce, and to highlight challenges associated in learning from large distributed datasets. In this thesis, we are not impressed with *big* data, but the very opposite: we are impressed with *small* data. For many real-world problems, data is slow or expensive to acquire. If so, *data efficient* learning helps to make better use of existing data, and judiciously select new data. *Bayesian* learning makes efficient use of existing data, by seamlessly combining data with our prior knowledge and by yielding probabilistic predictions – more informative than point-predictions. In addition, *active* learning helps judiciously choose what data to collect next, by seeking specific data that is most likely to improve performance.

The specific context of small data learning we are interested in is control. We are interested in autonomously learning how to control initially-unfamiliar systems with minimal system interaction (i.e. data collection). To do so, we consider two aspects of control, system identification (learning the input-output mapping of an unfamiliar system) and controller design (optimisation of a controller function to minimise an objective loss function). *Autonomous* learning disqualifies many traditional approaches to system identification that require human experts, including specification of a system's differential equations governing the dynamics, Lyapunov analysis, or other types of graphical analysis. Traditional control approaches also treat system identification and controller design as separate, sequential phases of learning to control a system. Only after a system is identified, is a controller designed. Learning a system in this way is called *passive* learning (opposed to *active*), and can

be extremely data-inefficient. Without giving the robot a chance seek data likely to assist dynamics modelling and controller design, much of the data collected during a system identification phase could be redundant. Yet for systems that are slow, expensive to operate, or prone to wear and tear, such superfluous data is an unnecessary cost.

A solution to autonomous, data efficient learning of control is *dual control*: simultaneous system identification and controller optimisation. By contrast, traditional control theory assumes known dynamics whereas dual control assumes unknown dynamics a priori (Wittenmark, 1995). Reinforcement Learning (RL) addresses the same problem as dual control, sometimes associated with discrete state spaces, but applicable to continuous states also (Sutton and Barto, 1998). RL learns to control unfamiliar systems through a process of trial and error, similar to a child learning to ride a bicycle. In this thesis, we concentrate on episodic tasks. The robot has repeated trials to interact with a system over a finite time horizon before 'reset' to an initial state. Our goal is to minimise the total expected cost accumulated over a fixed number trials (episodes).

Many RL algorithms can provably find near-optimal controllers eventually, but they can take many trials to do so. In this thesis we develop data efficient RL algorithms by 1) using Bayesian modelling of the system's stochastic nonlinear dynamics, 2) evaluate controllers according to how they are used (which we call being 'faithful to reality'), and 3) balancing exploration and exploitation (defined later). As we shall see in this thesis, Bayesian modelling is particularly critical for data efficient learning and *small* datasets. Bayesian modelling's probabilistic foundation avoids overfitting models, unlike many control methods that conduct system identification using least squares estimation on polynomials or neural networks. Such probabilistic system identification is important during controller design, offering informative probabilistic-predictions of state trajectories (opposed to point-predictions) used to optimise a controller (Deisenroth, 2009).

1.3 Contributions

This thesis achieves unprecedented data-efficiency in learning to control some variations of the cart-pole and cart-double-pole swing-up tasks. Doing so, this thesis contributes to the field of data efficient learning of control is several ways:

1. We introduce a novel probabilistic framework to evaluate controllers that make decisions from filtering observations in § 3.3. Filtering helps mitigate the effects

of sensor-noise on state estimation, and integrating a filter into a system is often straightforward. However, *simulating* such a controller (for controller improvement) is much more challenging and one of our novel contributions. Simulating filtered control required analytically capturing the plausible distributions over the robot's future belief states. Accurate simulations must be faithful to reality. Thus, each possible belief within the analytic distribution updates and decide controls in simulation in the exact same way as happens for a single belief during system execution.

2. We derive Gaussian process predictions for inputs of a hierarchical uncertainty. Specifically, inputs which are Gaussian random variables, whose mean parameters are themselves Gaussian random variables. Such mathematics were required in § 3.3 since belief states represent subjective probability distributions, thus distributions over belief states formed hierarchical distributions.

3. We introduce a new type of Markov Decision Process (MDP) called latent variable belief MDP in § 3.4. A belief MDP is an alternate interpretation to the well known Partially Observable MDP (POMDP). We generalise POMDPs to *latent variable* belief MDPs which model *both* belief variables and latent state variables. Two benefits are an ability for the robot to understand the consequences of incorrect prior assumptions, and an ability to use different dynamics models offline during system simulation and online during system execution. During offline simulation, slow-yet-accurate models can be used, whereas during online execution, fast-yet-inaccurate dynamics models may be required instead, the performance of which can be evaluated by the accurate dynamics model in advance.

4. We derive an approximate mean and variance of a controller's predictive cumulative cost distribution. Both moments are useful for directed exploration using Bayesian Optimisation (BO) in § 4.4. This improves upon pure-exploitation PILCO algorithm.

5. Of our three BO algorithm, one is novel: an approximate lower-bound to the famous Gittins index, introduced § 4.3.4.

6. We introduce the a data efficient framework for *deep*-RL algorithm in Chapter 5, able to scale to high dimensional observations.

1.4 OUTLINE

The rest of this thesis proceeds with a summary of background material in Chapter 2, three novel extensions to PILCO in Chapter 3 – Chapter 5, and conclusions in Chapter 6. Chapters may contain optional sections, marked with a star (*) symbol in their title, which the reader may safely skip over. Optional sections comprise tangential work, overly technical work, or incomplete work. To keep track of terminology, we list the meanings of all symbols and acronyms on page 195. A summary of each chapter follows:

CHAPTER 2, BACKGROUND: We discuss relevant background, in the control theory and RL literature, including methods for system identification and controller design. We introduce PILCO, a model-based RL algorithm for continuous state and action spaces, which has already shown unprecedented data efficiency for a variety of tasks such as the cart double-pole swing-up.

CHAPTER 3, FILTERED PILCO: We present a data efficient RL method for continuous state-action systems under significant observation noise. Data efficient solutions under small noise exist, such as PILCO, which learns the cartpole swing-up task in 30 seconds. PILCO evaluates controllers by planning state-trajectories using a dynamics model. However, PILCO applies controllers to the observed state, therefore planning in *observation*-space. We extend PILCO with filtering to instead plan in *belief*-space, consistent with POMDP planning. This enables data efficient learning under significant observation noise, outperforming more naive methods such as *post-hoc* application of a filter to controllers optimised by the original (unfiltered) PILCO algorithm. We test our method on the cartpole swing-up task, which involves nonlinear dynamics and requires nonlinear control.

CHAPTER 4, EXPLORING PILCO: PILCO's extreme data efficiency was largely due a principled treatment of dynamics uncertainty using a probabilistic model during controller evaluation to compute the expected cumulative cost of controllers. Using the same probabilistic model, we additionally compute cumulative cost variance, to direct exploration towards testing controllers of uncertain cumulative cost. By doing so, we show we can achieve even greater data efficiency than the unprecedented performance of PILCO. We compare PILCO against three extensions that intentionally explore using Bayesian optimisation, testing using PILCO's original cart double-pole swing-up task.

CHAPTER 5, DEEP-PILCO: Deep RL is an emerging force in the RL community, able to learn control in high dimensional state-space tasks. However, a common shortcoming of deep RL is data inefficiency, with current approaches requiring thousands of trials to learn even the simplest of tasks. In this chapter we extend PILCO's framework to use deep dynamics models, and look at the probabilistic equivalent of deep RL: Bayesian deep RL. We lay foundations towards data efficient model-based Bayesian deep RL and demonstrate our approach on the cart-pole swing-up problem.

CHAPTER 6, CONCLUSIONS: Finally, we provide some conclusions of this thesis and an outlook of interesting avenues for future research.

Chapter 2

Learning to Control Dynamical Systems

The purpose of this chapter is to introduce existing literature pertaining to learning to control dynamical systems. We shall progressively focus our discussion towards our goal of data efficient learning of control. Doing so, we intend to aid the reader's understanding of – and give context to – our novel contributions explained in Chapters 3–5.

Learning control of dynamical systems is a broad subject. Applications range form industrial (refining, manufacturing, power), transportation, logistics, electronics, robotics, computer science, to economics. With such breadth of application, the subject unsurprisingly draws interest from a wide array of academic disciplines. There is the control theory community itself. Other academic communities also study aspects of control under various names, parenthesised below. There is the mathematics community (operations research, optimisation), machine learning (reinforcement learning), economics (decision theory), statistics (bandits), computer science (dynamic programming), and robotics (path planning).

The control problems studied by various academic disciplines often overlap, yet communication between disciplines is surprisingly limited. Differing notation and terminology often obscures the similarity of solutions developed by each community, restricting their potential impact. Nevertheless, our intention in this chapter is to summarise relevant research from any community and to appeal to readers from either a control theory, machine learning, statistics, or computer science background. Most literature we will discuss is from control theory and reinforcement learning (RL). We will refer the reader to other control theory resources as we discuss them. For a more detailed summary of RL, we refer the reader to either Sutton and Barto

(1998); Szepesvári (2010). Sutton and Barto (1998) offers a comprehensive introduction to RL, whilst Szepesvári (2010) summarises some core ideas of RL in a more mathematically rigorous way.

We shall present common ideas across disciplines using a common notation. Whilst our opinion is that RL's notation is intuitive, we mostly borrow notation from control theory, being a larger community. Where less standardised notation exists we adhere to a principle of 'minimal notation clutter', using related symbols for related quantities. For example, a state x and control u will have dimensions X and U, not n and m as often used. Similarly, process noise associated with state x and observation noise associated with observation y at time t will be expressed ε_t^x and ε_t^y, not $v(t)$ and $w(t)$. As the chapter progresses, a fair amount of notation will be introduced. To avoid inundating our reader, a nomenclature section is provided on page 195 which lists all symbols and their meanings. Another reason we use control theory notation is related to our desire to introduce control theorists to certain machine learning tools. We wish to re-frame control in a way that is hopefully still recognisable to control theorists. In particular, we take the unconventional approach of modelling the process of control probabilistically using *Probabilistic Graphical Models* (PGMs). PGMs help show how random (or *uncertain*) control variables interrelate by specifying their conditional dependency structure. Such structure is important for system identification, i.e. fitting a dynamics model to data using Bayesian inference, and to forwards-simulate stochastic systems.

Our discussion of background literature begins by first describing the *process* of control given a controller and knowledge of a system's dynamics in § 2.1. Second, we discuss how to design a 'good' controller in § 2.2. Third, we no longer assume *a priori* knowledge of the system's dynamics, discussing how to learn dynamics *automatically* by collecting data and using machine learning in § 2.3. Fourth, we discuss data efficient (or *sample efficient*) learning in § 2.4. Finally, we discuss a particular algorithm called PILCO in § 2.5, a gold standard for data efficient learning of control given black-box, continuous-state, low-dimensional systems.

2.1 Control of Dynamical Systems

Previously we saw closed-loop control depicted using a block diagram (Fig. 1.1). Block diagrams, whilst intuitive, can be ambiguous. Here, we instead discuss control in more mathematically-precise terms. We first introduce each variable and function

of interest (with RL notation in parentheses), itemised below. Afterwards we discuss a general set of equations that define their relationship.

State Variables x : The first control variable we introduce is the Markov *system state* vector $x \in \mathbb{R}^{\mathrm{X}}$ (RL: *state* $s \in \mathcal{S}$), e.g. the joint configuration of a robot. The Markov state contains all variables relevant to predicting the evolution of the system to future states.

Control Variables u : We are free to affect the system in a constrained manner, by manipulating a vector of control variables $u \in \mathbb{R}^{\mathrm{U}}$ (RL: *action* $a \in \mathcal{A}$). for example, u could represent the set of voltages we choose to input into various motors attached to a robot.

Controller Function π : We specify how we will manipulate the control u in response to an estimated system state $\hat{x}$ using a *controller* function (RL: *policy* $\pi : \hat{x}_t \to u_t$), referred to as closed loop control. Similar to most discrete-time reinforcement learning we only consider zero-order hold controllers, meaning the chosen values of control signal vector u_t are held fixed between the discrete timesteps t and $t+1$.

Dynamics Function f : Changes in the state of the system x are governed by the *system dynamics* (or *plant*) function $f : x_t \times u_t \to x_{t+1}$ (RL: *transition function* T), which we may have complete, partial, or zero knowledge of. We assume the future state at time $t+1$ – x_{t+1} – only depends on the current state x_t, control u_t, possibly time t, and the dynamics function f. Given x_t, if the next state x_{t+1} is conditionally independent of all previous states $x_{0:t-1}$, the system is said to be Markov. We assume the Markov property throughout this thesis (a common assumption), which considerably simplifies controller design through factorisation. The setting of control decisions manipulating Markov systems is termed a Markov Decision Process (MDP).

Time t : An important aspect of specifying the control task is whether time t is discrete or continuous. Some analytic techniques are available for continuous time, otherwise the time may be artificially discretised to apply simpler control techniques. Our main focus is discrete time in this thesis. Even for discrete-time problems, real physical systems operate in continuous time, so one must specify what a controller should do in between discrete timesteps. For example, by modelling with discrete-time, we will only consider zero-order hold controllers. Another point is time-horizon: how long does the control

task last for? Since we use discrete-time, we will define time horizon T as an *integer* number of discrete time-steps (opposed to measuring time in seconds). The time horizon T can be infinite, but we will work with finite horizons, typically the order of several seconds in our simulated robotic experiments. We repeat a control task over E-many trials (RL: *episodes*), each trial lasting T timesteps. In this chapter, we will sometimes interchange between continuous and discrete, relying on context to make sense, rather than using new variables. All subsequent chapters assume discrete time.

Observation Variables y : A robot often lacks the ability to observe the state of the system x exactly, or perhaps only observes some of the state vector's variables, but not all. If so, state x is called a *hidden* or *latent* variable and said to be 'partially observed' opposed to 'fully observed'. In such cases, we instead reason about what is directly observed, i.e. readings from the robot's sensors: the observed variable $y \in \mathbb{R}^{\mathrm{Y}}$ (RL: *observation* $o \in \Omega$). Observed variables help us reason about the *plausibility* of possible system states x.

Observation Function g : Observations y are read as the output of a *sensor* or *measurement* function g (RL: *observation function* $O : o \times x \to \mathbb{R} \in [0, 1]$, a probability). For example, the observation y could be what is read from multiple potentiometer sensors attached to a robotic arm which relate to (but do not necessarily fully determine) the state of arm's current configuration.

Cost Variable and Function c : A user specifies *what* they want a robot to achieve (not *how* an outcome should be achieved) by defining a 'cost' incurred by the robot for being in a particular state. The user supplies a $\mathrm{cost}(\cdot)$ function (RL: *reward function* R) which outputs a cost variable c_t (RL: *reward* r_t) incurred at each timestep t for being in a particular state x_t. In control theory, the cost function is usually a function of both the current state x_t and current control chosen u_t. If the cost function is stochastic, usually the expected cost is all that is required to compute the overall loss (discussed below). In RL, the cost can also depend on the control u_t following state x_{t+1}, since this additional generality still preserves the state's Markov property. A distinct 'terminal cost' is sometimes used, a once-off cost incurred at time T, which we ignore.

Loss Variable and Function J : The overall objective is usually to minimise the sum of expected costs: $J_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} \mathbb{E}\left[\mathrm{cost}(x_\tau)\right]$ (control theory: minimise the expected *cost-to-go* J, RL: maximise *value* V, bandits: minimise *cumulative*

regret R), which we will refer to as the *utility* or *loss* function. An optional discount factor $\gamma \in [0,1]$ represents an 'interest rate', encoding 'costs incurred sooner matter more than costs incurred later'. Often γ is ignored, by setting $\gamma = 1$. A trivial difference exists between control theorists who *minimise* a sum of costs, and the RL community who *maximise* a sum of rewards. The triviality is: $\text{reward}(x) = -\text{cost}(x) = -\text{cumulativeRegret}(x) + \text{constant}$.

Belief Function b **:** A belief function b or $b(x)$ is a probability distribution over the state space $\mathbb{R}^X$, representing the robot's subjective uncertainty concerned with being in any particular state x. When x is partially observed via noisy observations y, a robot cannot be certain of which state x it is in, but nevertheless can construct a set of plausible states it might be in. The belief adheres to the Bayesian interpretation that rational actors should update subjective beliefs according to the rules of probability. For instance, a prior belief $b_{t|t-1} = b_{t|t-1}(x_t) = p(x_t|y_{0:t-1}, u_{0:t-1})$ may combine with the likelihood of an observation y_t to update a robot's belief to $b_{t|t} = b_{t|t}(x_t) = p(x_t|y_{0:t}, u_{0:t-1})$. The first subscript of $b_{t|t-1}$ refers to the belief function of state x at time 't', i.e. x_t, whilst the second subscript refers to conditioning on all observations y up until time '$t-1$' inclusive (notation used by Kalman filtering, described next chapter). The belief acts as a sufficient statistic that summarises all past observations $y_{0:t}$ into a single probability distribution. Robot beliefs are not always *explicitly* mentioned in control theory as a belief function, yet considered implicitly when filtering, discussed § 3.2. In computer science beliefs are explicitly considered in Partially Observable MDPs (POMDPs), discussed § 2.2.2. We find explicit handling of a belief function helpful to distinguish between different types of probabilities we later discuss, e.g. distinguishing between the probability of the next state of the system $p(x_{t+1})$ due to objective system stochasisties, versus subjective beliefs that the robot might have in the next timestep $b_{t+1|t} = b_{t+1|t}(x_{t+1})$.

Now that we have introduced the principle variables and functions for control, we will discuss their relationship with each other (except for the belief, discussed later). A set of equations, which describe control in sufficient generality to apply to the majority of continuous-state, continuous-control, continuous-observation, discrete-time control tasks, follows:

General equations of discrete-time control:

$$\begin{aligned}
\text{initial state} &: x_0 = \varepsilon^{x_0} && \in \mathbb{R}^{\mathrm{X}}, && (2.1)\\
\text{dynamics} &: x_{t+1} = f(x_t, u_t, t, \varepsilon_t^x) && \in \mathbb{R}^{\mathrm{X}}, && (2.2)\\
\text{observation} &: y_t = g(x_t, u_t, t, \varepsilon_t^y) && \in \mathbb{R}^{\mathrm{Y}}, && (2.3)\\
\text{policy} &: u_t = \pi(y_{0:t}, u_{0:t-1}, t, \varepsilon_t^u) && \in \mathbb{R}^{\mathrm{U}}, && (2.4)\\
\text{cost} &: c_t = \mathrm{cost}(x_t, u_t, x_{t+1}, t, \varepsilon_t^c) && \in \mathbb{R}, && (2.5)\\
\text{loss} &: J_t = \textstyle\sum_{\tau=t}^{T} \gamma^{\tau-t} \mathbb{E}\left[c_\tau\right] && \in \mathbb{R}. && (2.6)
\end{aligned}$$

Several sources of random noise (from generic distributions) are present in (2.1)–(2.6) inducing different stochasticities in the system. Noise variables include: random initialisation of the system state ε^{x_0}, random process noise ε_t^x in the dynamics f, random observation noise ε_t^y modelling an imperfect sensor g, random controller noise ε_t^u (either inherent due to imperfect motors or deliberately injected) resulting in a stochastic controller, and random cost noise ε_t^c.

RL offers an alternative to modelling functions of noise variables, by instead using probabilistic models to encapsulate system stochasticities. RL models the probabilities of events directly. For example, a transition model $T(x_t, u_t, x_{t+1}) = p(x_{t+1}|x_t, u_t; f)$. Likewise RL's observation model models $p(y_{t+1}|x_t, u_t, x_{t+1}; g)$, a policy models $p(u_t|x; \pi)$, and a reward model models $p(c_t|x_t, u_t, x_{t+1}; \mathrm{cost})$. RL implicitly assumes noise variables are independent and identically distributed (i.i.d.).

2.2 Designing a Controller

So far we have described the *process* of control. For instance, we can understand how the system state x might evolve given a controller π, dynamics f, and sensor g. In this section, we no longer assume we are given a controller π, but must design a controller ourselves. Furthermore, our interest is in *automatic* controller design. As such we ignore methods that require human experts.

2.2.1 Criteria for a Good Controller

Before getting into controller design, we should first address: what defines a 'good' controller? Some common criteria (not necessarily mutually exclusive) a user may be concerned with are:

- an ability to stabilise the system state x close to goal state x^*,
- energy expenditure, e.g. resulting from large voltages $|u| \gg 1$,
- transient response time, i.e. time to $||x - x^*|| < \varepsilon$,
- restricted rate of change of control $||u_{t+1} - u_t||$ (to minimise motor-wear),
- robustness to an inaccurately measured dynamics model f,
- simplicity, e.g. a linear controller function π, since the properties of linear controllers are well understood.

Whilst each criterion above poses interesting quantities to measure, predict, and trade off, we would rather the user specify their desiderata and trade-offs between each desideratum within a single white-box cost function for us to optimise. By optimising a sum of costs, we view control as an optimisation problem only, a very general treatment of control (more general than common goals of stabilising a system state nearby a reference point), which leads us onto the topic of optimal control.

2.2.2 Optimal Control

An optimal controller is one, which minimises the loss function J. We use the star symbol ($*$) superscript to denote optimality. For instance, the minimum loss J^* is attained by following an optimal controller π^*. During the 1960s modern control theory had established two main interpretations and approaches for 'optimal control' (Fernández Cara and Zuazua Iriondo, 2003). This first was Richard Bellman's Dynamic Programming (DP) approach to compute *globally* optimal controllers (Bellman and Kalaba, 1965). DP is well known to most communities. Second was Lev Pontryagin's Maximum (or Minimum) Principle (PMP) to compute *locally* optimal controllers, well known to the control community (Pontryagin, 1987). We will talk briefly about DP, and then PMP.

DYNAMIC PROGRAMMING

Central to DP is the Bellman equation. The Bellman equation defines the loss function $J^*(x)$ in a recursive relationship. When the state x is fully observed, the system is a Markov Decision Process (MDP). For discrete-time MDPs, the Bellman equation is:

$$J_t^*(x_t) = \min_{u_t} \int p(x_{t+1}|x_t,u_t;f)\left[\text{cost}(x_t,u_t,x_{t+1}) + \gamma J_{t+1}^*(x_{t+1})\right] \mathrm{d}x_{t+1}, \tag{2.7}$$

which defines the expected cumulative cost (loss) from state x following optimal control decisions (as given by the min operator) as a recursive relationship. Another interpretation of the Bellman equation above is as a satisfactory condition that a given loss function $\hat{J}$ is optimal. If we test a given function $\hat{J}(x)$, and the function $\hat{J}(x)$ satisfies the Bellman equation $\forall x_t \in \mathbb{R}^{\mathrm{X}}$, then $\hat{J} = J^*$ is optimal. Even loss estimates $\hat{J}(x)$, which are not yet optimal, can converge towards optimality by iteratively bootstrapping from other loss estimates 'further down the line' at x_{t+1}. Given the optimal loss function J_t^* (RL: *state-value* function $V_t : x_t \in \mathbb{R}^{\mathrm{X}} \to \mathbb{R}$), another important function is the optimal loss conditioned on a first control (RL: *action-value* function $Q_t : x_t \times u_t \in \mathbb{R}^{\mathrm{X}+\mathrm{U}} \to \mathbb{R}$), which we refer to later:

$$J_t^*(x_t,u_t) = \int p(x_{t+1}|x_t,u_t;f)\left[\text{cost}(x_t,u_t,x_{t+1}) + \gamma J_{t+1}^*(x_{t+1})\right] \mathrm{d}x_{t+1}, \tag{2.8}$$

defining the optimal controller:

$$\pi^*(x_t) = \operatorname*{argmin}_{u_t} J_t^*(x_t,u_t). \tag{2.9}$$

To understand why the Bellman equation (2.7) is recursive, note the loss (defined (2.6)) is minimised by

$$\begin{aligned} J_t^*(x_t) &= \min_{\pi} \mathbb{E}\left[\sum_{\tau=t}^{T} \gamma^{\tau-t}\text{cost}(x_\tau,\pi(x_\tau),x_{\tau+1})\right] && (2.10)\\ &= \min_{\pi} \mathbb{E}\left[\text{cost}(x_\tau,\pi(x_\tau),x_{\tau+1}) + \gamma\sum_{\tau=t+1}^{T} \gamma^{\tau-(t+1)}\text{cost}(x_\tau,\pi(x_\tau),x_{\tau+1})\right] && (2.11)\\ &= \min_{\pi} \mathbb{E}\left[\text{cost}(x_\tau,\pi(x_\tau),x_{\tau+1}) + \gamma J_{t+1}^*(x_{t+1})\right] && (2.12) \end{aligned}$$

(Sutton and Barto, 1998, chapter 4). The loss is the least expected cost-to-go over all trajectories from that state. Both (2.10) and (2.11) show that the global minimisation of the accumulative cost must equal the minimisation of the next cost and all costs

after that (whose definition is that $J^*_{t+1}(x_{t+1})$, resulting in (2.12)). The benefit of such bootstrapping is solutions to the Bellman equation do not require a search over each possible state trajectory $\xi_t = \{x_t, u_t, ..., x_T\}$, exponential in the time horizon T, but rather polynomial since each state can summarise their cost-to-go independent on the paths that reach them.

Variants of the Bellman equation exist. For discrete states, the integrals in (2.7) and (2.8) become sums. The continuous time variant is the Hamilton–Jacobi–Bellman equation (HJB) discussed below. A third Bellman variant is when the state x is partially observed through observation y, called a Partially Observable Markov Decision Process (POMDP), also discussed below.

*DYNAMIC PROGRAMMING IN CONTINUOUS TIME: In continuous-time we aim to optimise

$$J^*(x_t, t) = \min_{\pi} \int_t^T e^{-\frac{\tau - t}{\rho}} \text{cost}(x_\tau, \pi(x_\tau, \tau), \tau) \mathrm{d}\tau, \tag{2.13}$$

subject to the continuous-time system dynamics constraints $\mathrm{d}x/\mathrm{d}t = f(x_t, u_t, t)$, and where $e^{-\frac{\tau - t}{\rho}}$ replaces γ as the continuous-time discounting of future costs, where ρ is a discounting-rate parameter. Since time is continuous, the loss J^* is not defined by a recursive relationship (as (2.7) is), but rather a Partial Differential Equation (PDE) called the Hamilton–Jacobi–Bellman equation (HJB) (Bellman, 1965; Doya, 2000):

$$-\frac{1}{\rho} \frac{\partial J^*(x_t, t)}{\partial t} = \min_{u_t} \left(\text{cost}(x_t, u_t, t) + \frac{\partial J^*(x_t, t)}{\partial x_t} f(x_t, u_t, t) \right). \tag{2.14}$$

As a PDE, the continuous-time loss function J^* is solved analytically. However, only a small number of circumstances exist where (2.14) is analytically solvable. One example is systems with Linear dynamics functions f and a Quadratic cost functions (LQ). Another example is the continuous-time underactuated pendulum task (which has nonlinear dynamics) with quadratic cost function. Unfortunately, solving such differential equations is too difficult for more complex problems in general. Thus the discrete-time models are more common.

DYNAMIC PROGRAMMING WITH LATENT STATES: A robot will be uncertain about its current state if unable to fully observe the outcomes of its own transitions through a state space. 'Partial observability' describes the situation where a robot receives noisy observations which provide some information but not conclusive information as to which state the robot transitioned into. Such a system is called a Partially Ob-

servable Markov Decision Process (POMDP) Astrom (1965); Sondik (1971). POMDPs use a belief function $b(x)$. As part of optimal control under uncertainty, a robot must maintain (continually update) its belief. Starting with:

$$b_{t|t}(x_t) \quad = \quad p(x_t|y_{0:t}, u_{0:t-1}), \tag{2.15}$$

then executing control u_t and making observation y_{t+1} from prior belief $b_{t|t}$ the robot updates its belief using Bayes rule:

$$\begin{aligned} b_{t+1|t+1}(x_{t+1}) &\doteq p(x_{t+1}|y_{0:t+1}, u_{0:t}) && (2.16)\\ &= p(x_{t+1}|b_{t|t}, y_{t+1}, u_t) && (2.17)\\ &= \frac{\int b_{t|t}(x_t)p(x_{t+1}, y_{t+1}|x_t, u_t; f, g)\mathrm{d}x_t}{\iint b_{t|t}(x_t)p(x_{t+1}, y_{t+1}|x_t, u_t; f, g)\mathrm{d}x_t\mathrm{d}x_{t+1}}. && (2.18) \end{aligned}$$

In RL, the probability weighting $p(x_{t+1}, y_{t+1}|x_t, u_t)$ is decomposed into two meaningful terms, each of which might be known otherwise in the process of being learned from data:

$$p(x_{t+1}, y_{t+1}|x_t, u_t; f, g) \quad = \quad \underbrace{p(x_{t+1}|x_t, u_t; f)}_{\text{transition model}} \cdot \underbrace{p(y_{t+1}|x_t, u_t, x_{t+1}; g)}_{\text{observation model}}. \tag{2.19}$$

The belief is a distribution yet still considered as a *belief-state* or *information-state* that the robot is currently in (opposed to a physical state x). The POMDP Bellman equation for optimal loss as a function of the belief is:

$$\begin{aligned} J^*_t(b_{t|t}) \quad = \quad & \min_{u_t} \int b_{t|t}(x_t) \iint p(x_{t+1}, y_{t+1}|x_t, u_t; f, g) \\ & \times \big(\text{cost}(x_t, u_t, x_{t+1}) + \gamma J^*_{t+1}(b_{t+1|t+1})\big)\mathrm{d}y_{t+1}\,\mathrm{d}x_{t+1}\,\mathrm{d}x_t, \end{aligned} \tag{2.20}$$

noting $b_{t+1|t+1}$ is a function of u_t and y_{t+1} as per (2.17). An alternate interpretation of POMDPs is that they are in fact belief-MDPs in (MDP over *belief* space, not state space) which has its own 'Belief-MDP' Bellman equation (Kaelbling et al., 1998):

$$\begin{aligned} J^*_t(b_{t|t}) \quad = \quad & \min_{u_t} \int p(y_{t+1}|b_{t|t}, u_t; f, g) \\ & \times \big(\text{cost}(b_{t|t}, u_t, y_{t+1}) + \gamma J^*_{t+1}(b_{t+1|t+1})\big)\mathrm{d}y_{t+1}, \end{aligned} \tag{2.21}$$

where

$$\begin{aligned} \text{cost}(b_{t|t}, u_t, y_{t+1}) \quad \doteq \quad & \iint b_{t|t}(x_t)p(x_{t+1}, y_{t+1}|x_t, u_t; f, g) \\ & \times \text{cost}(x_t, u_t, x_{t+1})\mathrm{d}x_{t+1}\,\mathrm{d}x_t. \end{aligned} \tag{2.22}$$

Notice the close similarity between the belief-MDP Bellman equation (2.21) and the original MDP Bellman equation in (2.7). The principle difference is the physical state x is exchanged for a belief b. This analogy is why POMDPs are often referred to as 'belief-MDPs'.

PONTRYAGIN'S MINIMUM PRINCIPLE

Pontryagin's Minimum Principle (PMP) is a sufficient condition to a control path $\xi = \{x_0, u_0, .., x_T\}$ being locally optimal subject to the dynamics constraints $\mathrm{d}x_t/\mathrm{d}t - f(x_t, u_t) = 0$ (Pontryagin, 1987). Approaches that aim to satisfy the PMP are termed *variational approaches*, *multiplier approaches*, or *trajectory optimisation*. Often Lagrange multipliers are used to handle an optimisation under constraints. Even though the PMP concerns local optimality, it is often used to find candidate trajectories that are possibly globally-optimal, serving as a necessary condition of global optimality. The PMP minimises: $\min_{u_{0:T}} J(x_0) = \min_{u_{0:T}} \int_0^T \mathrm{cost}(x_t, u_t)\mathrm{d}t$ subject to the constraints $\dot{x} = f(x,u)$, $x_0 = \varepsilon^{x_0}$, and any constraints on u.

The PMP only solves the control signal from the current state $J_t(x_t)$, not all possible states $J_t(x) \forall x \in \mathbb{R}^{\mathrm{X}}$. As such, a PMP solution is often much cheaper to compute than DP solutions. Additionally, the PMP does not suffer from DP's 'curse of dimensionality' problem. PMP can easily scale to problems of high state-dimensionality, $\mathrm{X} \gg 1$. PMP methods start with a nominal trajectory, generated from random controls for the first optimisation step, otherwise the trajectory from the previous optimisation step. Then the trajectory is updated, usually with gradient methods. Once converged, the trajectory ξ^* is locally-optimal.

2.2.3 CONTROLLING LINEAR-DYNAMICAL SYSTEMS

A vast amount of mature literature exists when the dynamics and observation functions are known to be linear equations (or linear differential equation in the continuous-time case), i.e. $f(x,u,t) = A_t x + B_t u + \varepsilon_t^x$ and $g(x,u,t) = C_t x + D_t u + \varepsilon_t^y$, where A_t, B_t, C_t, D_t are known matrices at each timestep. Many textbooks have been written on linear control theory. Some include Åström and Wittenmark (2013); Nise (2007); Ogata and Yang (1970). Perhaps a more clearly written textbook is Aström and Murray (2010). For a more mathematical treatment see Sontag (2013). The advantage of linear dynamics, quadratic cost, and Gaussian noise is how well-understood properties of the system are. Question of frequency-response, transient response times, and steady-state errors of a system are easily answered, questions

a practitioner often cares about when designing a controller. In addition, it is well understood how to tune a controller to optimise each desideratum above.

For example, the state of many classical mechanics setups have dynamics f often expressible with second-order Ordinary Differential Equations (ODE). For example, the state of an object-positions in a mechanical spring-dampener-inertia system with force control inputs, or the state of currents in an electrical inductor-resistor-capacitor network with voltage control inputs. In such settings, PID (Proportional-Integral-Derivative) controllers are suitable (Nise, 2007):

$$\pi(y_{0:t}) \quad = \quad K_p e_t + K_i \int_0^t e_\tau \mathrm{d}\tau + K_d \frac{\mathrm{d}e_t}{\mathrm{d}t}, \qquad \text{where error} \quad e_t = x^* - y_t \tag{2.23}$$

is the difference between the present state x_t (perhaps noisily observed as y_t) to a desired goal state x^*. Controller parameters are K_p, K_i, and K_d must be tuned. The name PID comes from three correcting terms: Proportional, concerns present error, applying a corrective control signal in proportion with the present error signal; Integral, concerns past error, by being proportional with longterm steady-state errors; and Derivative, concerns future error, by being proportional to the rate of change of error in time, and thus 'predicting' future error to minimise overshooting etc. Tools to design PID controllers (i.e. optimise parameters K_p, K_i and K_d) include the Laplace transform and Root-Locus methods for stability analysis, and the Fourier transform and Bode plots for frequency analysis (Nise, 2007).

If in addition to Linear dynamics the cost is Quadratic (LQ), then the global-optimisation problem is convex. For example, $\text{cost}(x,u,t) = x^\intercal Q_t x + u^\intercal R_t u + 2x^\intercal N_t u$. Globally optimal controllers for LQ systems have closed-form solutions both in discrete and continuous time (the HJB equation (2.14) has an analytic solution in such case). If the state x is fully observed and no process noise ε_t^x exists, then the optimal controller is the Linear Quadratic Regulator (LQR) (Stengel, 1986). Otherwise, if the state x is only partially observed through observation y, then as long as the observation noise ε_t^y is additive and Gaussian distributed (and the system satisfies a condition known as observability, see Kalman (1959)), an analytic solution is also possible. The optimal controller is the application of the LQR controller to the state estimation by a Kalman filter, termed a Linear Quadratic Gaussian controller (LQG). An LQG controller is a function of the least-squares estimate of the state given previous observations and controls: $\pi^*(b_{t|t}) = \pi^*(\hat{x}_t) = -L\hat{x}_t$ where $\hat{x}_t = \mathbb{E}\left[b_{t|t}\right] = \mathbb{E}\left[p(x_t|y_{0:t}, u_{0:t-1})\right]$. Linear dynamical systems have the desirable property of *certainty equivalence*, where optimal control is only dependent on the expected

belief, not a function of the full belief-distribution $b_{t|t}(x_t)$ which nonlinear optimal control must consider.

2.2.4 Controlling Nonlinear-Dynamical Systems

Although dealing with linear dynamical systems is well understood, the real world often involves nonlinear dynamics. To complicate matters further, the range of possible control outputs u will have physical limitations. As such, there is usually no such thing as linear controller functions π in the real world. Some major difficulties in controlling nonlinear dynamics are:

- The optimisation task of finding globally-optimal controllers is now non-convex in general. Solving for globally-optimal controllers of nonlinear dynamics is intractable.
- Open loop control, which is undesirable even for linear systems, can be completely unusable given nonlinear systems can be chaotic.
- Certainty equivalence, whilst true in linear systems, is generally not true for nonlinear systems. Consequently, belief variance is critical to compute and cannot be pre-computed (opposed to Kalman filters which can pre-compute future variances, discussed next chapter).
- Inference is no longer tractable.

Like linear systems, much of the control literature of nonlinear systems is still concerned with various stability guarantees, some of which involve an expert present. For example, Lyapunov or asymptotic stability analysis, Poincaré maps, limit cycles, bifurcation analysis (i.e. how the stable fixed points appear/disappear as a function of system parameter values) are some common tools to address control of nonlinear systems (Khalil, 1996; Slotine et al., 1991). As before we will instead only concern ourselves with 'control as optimisation', and find controllers that optimise the loss function J.

Given linear systems were so well understood, and nonlinear systems entail such difficulties, a first approach to control a particular nonlinear system involves checking whether a system's dynamics can be safely *linearised*. Even though a system may not be linear, it is often safe to assume linear dynamics within a small operating range about some operating point (x', u'). Linearising a system around (x', u') opens the door to linear control tools and methods. If multiple operating points exist, a set

of linear controllers can be trained, one controller per operating point, termed *gain scheduling*. Alternatively, we can linearise a system around a different point (x', u') at each timestep $t = \{0, 1, .., T\}$. Doing so, we can still conduct local analysis to provide linear-control guarantees (under our assumptions of linearity at each time point) but lose global properties. An example is iterative-LQR (iLQR) control. For instance, from $x_{t+1} = f(x_t, u_t)$ to $x_{t+1} \approx x_t + \dot{x}_t \Delta t$, where

$$\dot{x}_t \approx f(x', u') + \underbrace{\frac{\partial f(x_t = x', u_t = u')}{\partial x_t}}_{A_t}(x_t - x') + \underbrace{\frac{\partial f(x_t = x', u_t = u')}{\partial u_t}}_{B_t}(u_t - u'). \tag{2.24}$$

By a change of variables, this can be rewritten in linear form: $\dot{\bar{x}}_t = A_t \bar{x}_t + B_t \bar{u}_t$. Notice that iterative linearisation at each time point (2.24) is equivalent to controlling a time-varying linear dynamical system. Incorporating a time dependence acts as a substitute for not modelling global nonlinearities directly. Even though the dynamics may be known to be nonlinear time-invariant, they are treated as if they were linear time-variant, with different linearisations per timestep.

A special class of nonlinear systems is 'control affine' systems, e.g. used by Xie et al. (2015). A system is control affine if $\dot{x} = f_1(x) + f_2(x)u$, meaning a system possibly nonlinear w.r.t. x yet linear w.r.t. u. If matrix $f_2(x)$ has rank $\geq$ X (the dimension of state x) then the system is '*fully actuated*' (at point x in the state space at least). With fully actuated systems we can apply a control technique called '*feedback linearisation*', a way to algebraically linearise a system by cancellation of nonlinear terms distinct from Jacobian-linearisation (2.24). The motivation is again that application of well-understood linear control methods can then be applied. The matrix $f_2(x)$ is rectangular in general, but for simplicity let us assume $f_2(x)$ is square. If $\text{rank}(f_2(x)) \geq$ X then $f_2(x)$ is invertible and feedback linearisation uses a controller of the form: $u = f_2^{-1}(x)[v - f_1(x)]$, where v is our choice to manipulate. By substituting the expression for u above into a control affine system, the nonlinearities cancel, leaving a *linear* equation $\dot{x} = v$. By focusing on controls v instead of u, the dynamics become a linear function. We are free to inject any forces into v as long as our actuators can provide the torques we request. Feedback linearisation is a powerful tool and should be considered if its conditions are satisfied. Exceptions to applicability include systems with uncertainties, state constraints, control constraints, or *underactuated* systems whose $\text{rank}(f_2(x)) <$ X prevents $f_2(x)$ inversion, in which case feedback linearisation is inapplicable.

If the system is not control-affine, a popular closed-loop control method is Model Predictive Control (MPC) (Garcia et al., 1989). MPC satisfies (or attempts to satisfy) Pontryagin's Minimum Principle (PMP) discussed previously on page 17. At each timestep, MPC takes the current state x_t and uses a dynamics model to simulate forwards a single trajectory $\xi_t = \{x_t, u_t, ..., x_\tau\}$ multiple timesteps where $\tau \leq T$. The loss $J(\xi_t)$ is computed, and improved by gradient decent, until a locally optimal path ξ_t^* is found. Then the first control u_t is executed, resulting in the system moving forward a single timestep, to new state x_{t+1}. The process then repeats. At time $t = 0$, the path ξ_0 is usually randomly initialised. Otherwise usually MPC initialises ξ_t with shifted elements from the previously-optimised path ξ_{t-1}^*. Note MPC's simulation time horizon τ is not necessarily as long as the task's time horizon T. τ is usually less if running online and computationally-limited. Since the simulation-horizon gets shifted forward one timestep too, MPC is also called *receding horizon control*, as MPC then optimises $\xi_{t+1} = \{x_{t+1}, u_{t+1}, ..., x_{\tau+1}\}$. MPC has different variants including direct shooting, direct transcription, and direct collocation (Benson et al., 2006; Enright and Conway, 1992). Direct shooting uses a set of controls $x_0, u_0, u_1, ..., u_T$ to simulate a system with known dynamics $\dot{x} = f(x, u)$ to evaluate all $x_{0:T}$ then computing $J(x_0)$. Direct transcription instead optimises a nominal trajectory ξ_t, not by simulation, but by specifying dynamical constraints on x for the optimiser, often easier given optimising software since the simulation and associated chain rule of loss gradients does not need to be implemented. Direct collocation instead uses 'collocation points' between x_t and x_{t+1} for greater accuracy over Euler integration.

Advantages of MPC include being a general method for nonlinear control, since MPC does not make many assumptions about the nature of the system. MPC is even useful in some linear systems in the place of PID controllers, which have difficulties with large time delays and high-order dynamics. Disadvantages of MPC applied to stochastic systems are subtle. Often MPC is implemented such that *simulation* is open loop, whilst the *execution* of MPC is closed loop. For instance, MPC optimises trajectories under the assumption of determinism, but inevitably transitions stochastically to a new state x_{t+1}, a fact not anticipated when optimising the control path ξ_t. Nevertheless MPC still compensates for such prediction errors with feedback during execution and re-optimising a new control path ξ_{t+1} beginning at the corrected state x_{t+1}. We will show later in § 3.3.6 that such mismatches between open loop simulation and closed loop execution result in biased predictions of losses (predicted losses < executed losses) yielding suboptimal controllers. By contrast, other methods such as iLQR and iLQG have simulators that are 'faithful to reality',

simulating closed-loop feedback control along trajectories since execution involves closed-loop control. Even though PMP nonlinear control methods are locally optimal, and globally optimal methods (e.g. using DP) are intractable, locally optimality is often 'good enough', and often the only choice for large state dimensionality X tasks.

For the rest of this thesis we focus on local-optimality for tractability reasons. Some alternatives to single trajectory optimisation are deterministic and stochastic path-planning algorithms. Path planning algorithms use tree or graphs, e.g. the A^*, rapidly-exploring random trees (Lavalle, 1998), or probabilistic roadmaps (Kavraki et al., 1996) algorithms well known in robotics. For an overview see LaValle (2006).

2.2.5 Functional Forms of Controllers

Before concluding this section, we should discuss options for specifying the function form of the controller function π. From (2.4) we had the generalised controller $u_t = \pi(y_{0:t}, u_{0:t-1}, t, \varepsilon_t^u; \psi)$ where we now introduce optional controller parameters ψ, and assume visible states x_t instead of noisy observation y_t for ease of explanation.

Linear

Linear controllers have the form:

$$\pi(x;\psi) = \psi x, \tag{2.25}$$
$$\psi \in \mathbb{R}^{\mathrm{U}\times\mathrm{X}}. \tag{2.26}$$

Such a form is ideal for stabilising a system state x with small variations around a goal state x^*. LQR and LQG controllers for the infinite-time horizon use linear controllers.

Iterative-Linear

In § 2.2.4 we also discussed how using iterative linearisation treats nonlinear time-invariant dynamics functions as if they were linear time-variant dynamics functions. A similar trick can be used with the functional form of a controller function as well. Iterative-linear controllers have the form:

$$\pi(x,t;\psi) = \psi_t x, \tag{2.27}$$
$$\psi_t \in \mathbb{R}^{\mathrm{U}\times\mathrm{X}}, \tag{2.28}$$
$$\psi = \{\psi_1, ..., \psi_T\}. \tag{2.29}$$

There exists no linear controller that can swing up and stabilise the cart double-pole system (seen Fig. D.2), e.g. PID controllers or linear controllers using fixed matrices multiplied by the state estimate. Yet a *sequence* of linear controllers indexed by time can successfully do so (e.g. Pan and Theodorou (2014)). Advantages of sequential linear controllers is speed and easier optimisation without numeric instabilities (opposed to RBFs, discussed next). A disadvantage is the system is rendered partially open-loop control. Whilst the system is not technically open-loop (the control output at time t is a linear function of the state), the linear function itself is open-loop, and not a function of the state, opposed to gain scheduling approaches. This is problematic since no linear controller can control the cart-double pole swing up task. If the controller gets 'out of sync' with the state of the system, there is little hope of recovering. This occurs with imprecise starting times of the system, or if the controller simply fails to swing-up the pendulum the first time, the controller's linear function will change to be appropriate for stabilising an inverted pendulum, not swinging up the pendulum, even though a second attempt at swinging-up is required.

Such an issue would not be a problem if the dynamics were linear. For example, optimal control using iLQR or iLQG controllers of finite-time horizons use iterative-linear controllers. Such time dependence only exists because of an approaching time-horizon, not because the system requires different linear controllers to control different regions of the state space.

Radial Basis Function Network

A very flexible controller function is Radial Basis Function (RBF) networks with Gaussian basis functions. RBF controllers, like neural networks, can approximate any continuous function if the network is large enough. Hence, RBFs are attractive for learning nonlinear control autonomously, without the need of a human to specify task-dependent controller functional forms. A Gaussian RBF network per control output $\mathrm{u} = [1, ..., \mathrm{U}]$ is possibly defined:

$$u^{\mathrm{u}} = \pi^{\mathrm{u}}(x; \psi) = \sum_{\mathrm{p}=1}^{\mathrm{R}} w_{\mathrm{p}}^{\mathrm{u}} \exp\left[-\tfrac{1}{2} \sum_{\mathrm{x}=1}^{\mathrm{X}} \left(\frac{x_{\mathrm{x}} - \mu_{\mathrm{x}}^{\mathrm{p}}}{\sigma_{\mathrm{x}}^{\mathrm{u}}} \right)^2 \right], \tag{2.30}$$

$$\psi = \{w_1^1, ..., w_{\mathrm{R}}^{\mathrm{U}}, \mu_1^1, ..., \mu_{\mathrm{X}}^{\mathrm{R}}, \sigma_1^1, ..., \sigma_{\mathrm{X}}^{\mathrm{U}}\}. \tag{2.31}$$

The idea is each RBF will only 'activate' if the state x is close enough to the centroid μ in some (e.g. Euclidean) space, in this case measured if within length-scale σ,

contributing to the output with weighting w. A drawback of using RBFs is numerical instability: optimising the controller with gradient decent can result in pairs or centroids very close to each other $\mu_i \approx \mu_j$ with large opposing weights $w_i \approx -w_j$, $|w_i| \gg 1$. Such weights can grow without bound, resulting in the two large yet similar numbers subtracting from each other, resulting in a loss in precision.

2.3 Modelling the System's Dynamics

So far we have discussed how to design a controller under the assumption that we knew the dynamics f and observation function g. A term for such fully-known functions is 'white-boxes', where all the internal structure of the box is known, e.g. the Newtonian ODEs that describe the physics of how a system changes. Knowledge of both the dynamics and observation function has been a critical component of all control methods we have discussed so far, allowing anticipation of future events that can *evaluate* a controller, giving us the chance to then optimise the controller. Indeed trajectory based approaches such as MPC or iLQR largely depend on accurate knowledge of dynamics models, with negligible dynamics uncertainty. In this section we no longer assume we know the systems dynamics f *a priori* (we will, however, assume g is still known). We instead discuss how to model a system's responses to previously observed control inputs using a dynamics model $p(f)$. We will discuss: 1) the benefits of using a model for controller design, 2) how to learn a dynamics model from data (online or offline) termed *system identification*, and 3) choices of models we can use and their associated assumptions. System identification is about learning the input-output mapping of both functions only as a mathematical relation. An explanation of the internal representation that gives rise to such a mathematical function is not required.

2.3.1 No Model

First we should discuss whether models are necessarily for controller design. The short answer is 'no' — even globally-optimal controllers are discoverable without retaining any dynamics knowledge. Control theory techniques include *extremum seeking control* or *iterative learning control* discussed § 2.3.2. Reasons to avoid models include online executability with insufficient time to train a model or high dimensional state spaces (e.g. pixel spaces), which many models cannot scale to.

Several such *model-free* methods exist in RL. For small numbers of discrete states, the estimates for the optimal loss of each state can be tabulated, and eventually the optimal losses can be learned using Q-learning (Watkins, 1989). If the number of states is too great (or continuous), locally-optimal controllers can be designed using *value-function approximation*. Such approaches choose a functional form of the value function, whose parameters are fitted by regression of (2.7), i.e. attempting to satisfy the Bellman equation, using batches of logged dynamics data, e.g. sequences of input-output tuples $\{x_t, u_t, x_{t+1}\}$. However, such approaches can take 100-1000s episodes to learn simple tasks (Ernst et al., 2005; Heess et al., 2015; Lagoudakis and Parr, 2003). A third approach which neither requires a model nor the logging of any data is *policy-gradient* methods (Sutton et al., 1999), e.g. the REINFORCE algorithm (Williams, 1992). The REINFORCE algorithm estimates the *expected* cumulative cost gradient (gradients w.r.t. policy parameters) using a sample-average and a trick to cancel our dynamics function gradients (since they do not explicitly depend on policy parameters). However, this estimator is high variance. Like value-function approximation, policy gradients are applicable in high dimensions, optimise locally-optimal controllers, but unfortunately they are data intensive.

2.3.2 Grey-Box, Parametric System Identification

A grey-box system is one where some aspects of the dynamics are known *a priori*, but not all. For instance, we have some important insights into the system, but more can be learned. A common example is where the system dynamics functional form is known (e.g. the ODEs), yet parameter values are unknown. This is often the case when the physical process is well understood by a human expert yet some parameters are difficult to measure, e.g. moments of inertia and coefficients of friction. Such cases warrant parametric system identification, where our model of the dynamics matches the functional form of the known dynamics, with the parameters fitted to data (Xie et al., 2015). Some methods fit a system's parameters using maximum likelihood estimation or least squares estimation which can lead to overfitting (Durbin and Koopman, 2012, chapter 7); other methods using Bayesian inference to learn posteriors over parameters are preferable to avoid overfitting (Durbin and Koopman, 2012, chapter 13). An aesthetic solution is incorporating unknown dynamics parameters into the state representation x, transforming a learning task into a POMDP planning task (Duff, 2002; Ross et al., 2008; Webb et al., 2014). Tasks with finite numbers of states and controls can be similarly solved. For example, one may use two Dirichlet parameters to model the probability of each of the finitely-

many state-control-state transitions (Poupart et al., 2006). Note, such solutions are inapplicable to more general continuous-state 'black-box' systems.

Robust Control

If the grey-box system parameters can only be learned or measured beyond a certain confidence level, then *robust control* techniques are warranted. A robust controller's performance does not change greatly if the system's parameters are altered slightly (Ioannou and Sun, 2012). For example, a set of ODEs may only present an idealised version of the system, or parameters were measured within some tolerance. Robust control tries to deal explicitly with these modelling errors, when uncertain parameters and disturbances occur within tight sets. The aim is to give good performance (usually stability) in presence of errors. For example, high gain feedback control helps make a system robust to small errors in dynamics-parameters (Ioannou and Sun, 2012, section 1.2.1). A well known robust control methods is H-infinity loop-shaping, with trajectories robust to disturbances (McFarlane and Glover, 1992) (the name deriving from the use of the infinity norm).

From a Bayesian perspective, design of robust controllers is only well defined given a prior distribution of possible dynamics $p(f)$. Without explicit expressions of dynamic's uncertainties, treated according to the rules of probability, then 'robust control' methods appear *ad hoc*, and a measure of a controller's 'robustness' can be unclear. Vague definitions, e.g. 'robust controller tolerates a range of different dynamics without much change in performance' or 'controller designed for one system still works well under other systems' do not help determining whether one controller is 'more robust' than another nor 'optimally robust'. By ignoring prior information $p(f)$ and instead design controllers with intentional insensitivity to an unspecified range of possible dynamics, or simply a set of bounds, a controller resists fully exploiting relevant dynamics knowledge to otherwise act optimally under expectation of the prior belief. This brings into question how much the property of robustness contributes to an expected increase in the loss (Jaynes, 2003, section 21.2).

Adaptive Control

Whilst robust controller functions are fixed and applied to fixed partially-unknown systems, *adaptive controllers* adapt to time-varying systems (usually defined by time-varying system parameters), i.e. where $x_{t+1} = f(x_t, u_t, t)$. Alternatively, the adaptive

controller can be applied to systems which are initially unknown. Even though the system itself may be time-invariant, the information we have about the system is time-varying. Adaptive controllers must continually re-estimate system parameters online during system execution to adapt to the changing system. Often the context involved a slow but steady drift of system parameters. As an example, an aircraft whose weight changes as it burns fuel continually alters the plane's dynamics. Different from robust control, adaptive controllers do not require a prior on bounds of uncertain time-variant parameters. Controllers can be very simple, e.g. 'iterative learning control' used for stabilisation, as an auto-tuning proportional-controller:

$$u_{t+1} \;=\; \pi(y_t) \;=\; u_t + K_p e_t, \qquad \text{where error } e_t = x^* - y_t. \tag{2.32}$$

Full auto-tuning PID controller gains (K_p, K_i, K_d) are implemented under adaptive pole placement techniques. However, by adapting PID controller gains with no model to summarise the data seen the controller can adapt in ways that destabilise the system without a clear way to restore the system. Without a dynamics model $p(f)$, such undesirable events can not be predicted in advance of them happening. It remained unclear how to prove longterm stability of adaptive controllers. Such controllers could initially adapt well to a changing system and later fail (Anderson, 1985; Hespanha et al., 2003).

2.3.3 Black-Box, Nonparametric System Identification

A black-box dynamical system f is one we know little about *a priori*. Neither ODEs nor parameters are known, nor whether f is even expressible as an analytic function. Application of a grey-box solution by assuming a particular functional forms (e.g. polynomial) risk inability to model (or at best underfit) the arbitrarily-complex function. Instead, the modelling black-box systems can follow a cycle of data collection, model selection, model fitting (inference), then validation. If a model does not pass the validation stage, a different function may be required. With unknown dynamical functional forms, highly flexible and expressive approximating functions, such as NNs and RBFs, are attractive choices of models, both able to approximate any continuous function with enough nodes/centroids. A network structure is chosen in advance of model training, yet it is not always clear what structure a NN might need. Too few nodes, and the model will underfit the data. Too many, and the computational complexities rises, as does the ability of the model to overfit the data.

Nonparametric models, such as K-nearest neighbours, instead have model complexities, which grow in proportion to the size and complexity of the growing dataset, mitigating effects of a model underfitting the data. Nonparametric models and universal approximating functions avoid requiring prior dynamics knowledge from human experts specifying the dynamics' function form, instead modelling dynamics directly from data. In addition, nonparametric models do not require knowledge of *how complex* the unknown dynamics might be since the model's complexity grows with the available data. However, even though nonparametrics avoid underfitting, not all avoid overfitting.

Bayesian Nonparametrics

Bayesian NonParametric models (BNP) are particularly suited for dynamics modelling given their resistance to both underfitting (being Bayesian) and overfitting (being nonparametric). Overfitting otherwise leads to issues of *model bias*, and underfitting limits the complexity of the system this method can learn to control. BNP regression avoids overfitting by considering (integrating-over) *all* plausible dynamics functions that can explain the data according to the prior, opposed to optimising a *single* function which best explains the data. By considering all plausible functions, BNP models can quantify their predictive confidence, by returning full probabilistic distributions as predictions, instead of point-estimates as other nonparametric methods do. Probabilistic prediction is important when accurate simulation of plausible system trajectories $p(\xi_t)$ is required during controller design.

A popular BNP model for regression is the Gaussian Process (GP). GPs are widely used in control for systems identification due to their uncertainty estimates, which quantify predictive confidence, especially important for robust and data efficient control (Deisenroth and Rasmussen, 2011; Hemakumara and Sukkarieh, 2013; Kocijan et al., 2004; McAllister et al., 2012; Murray-Smith and Sbarbaro, 2002). We discuss the basics of GPs here, in the context of being a dynamics models. For further reading on GPs we recommend Rasmussen and Williams (2006).

Formally, a GP is an infinite set of random variables, e.g. $\{f(\tilde{x}) \in \mathbb{R} : \tilde{x} \in \mathbb{R}^{X+U}\}$, any finite subset of which is jointly Gaussian distributed. Here, a dynamics input point $\tilde{x}^\intercal \doteq [x^\intercal, u^\intercal]$ in the continuous space of $\mathbb{R}^{X+U}$ indexes a single real-valued Gaussian scalar random variable $f(\tilde{x})$. Being an uncountably *infinite* set of random variables, a sample from a GP defines a random *function*. Thus, GPs are also considered as a prior over the space of possible dynamics functions. When combined with data, a GP becomes a posterior over functions. A GP prior is fully

described by a mean function $\mu(\tilde{x}) = \mathbb{E}_f[f(\tilde{x})]$ and a covariance function (or *kernel*) $k(\tilde{x},\tilde{x}') = \mathbb{C}_f[f(\tilde{x}),\ f(\tilde{x}')]$, expressed $f(\tilde{x}) \sim \mathcal{GP}(\mu(\tilde{x}), k(\tilde{x},\tilde{x}'))$. The functional form of k is restricted to those producing positive definite matrices (Rasmussen and Williams, 2006, section 4.3). The most common covariance function is the squared exponential: $k(\tilde{x},\tilde{x}') = s^2 \exp\left(-\frac{1}{2}(\tilde{x}-\tilde{x}')^\intercal \Lambda^{-1}(\tilde{x}-\tilde{x}')\right)$, where s^2 is called the signal variance and Λ is a diagonal matrix of squared length scales defining how smooth we expect the functions are, and noise variance is Σ^ε. With N-many training data point in training input matrix $X \in \mathbb{R}^{N\times(X+U)}$ and training output vector $y \in \mathbb{R}^N$, the GP predictive distribution is:

$$f(\tilde{x})|X,y \sim \mathcal{N}\left(\mu(\tilde{x}) + k(\tilde{x},X)\beta,\ k(\tilde{x},\tilde{x}) - k(\tilde{x},X)(K+\Sigma^\varepsilon)^{-1}k(X,\tilde{x})\right), \tag{2.33}$$

$$\beta \doteq (K+\Sigma^\varepsilon)^{-1}(y-\mu(X)), \tag{2.34}$$

$$K \doteq k(X,X), \qquad \text{called the Gram matrix.} \tag{2.35}$$

The main benefit of GP dynamics models in control is an ability to model arbitrary continuous functions – including uncertainty estimates – with fairly general and high level assumptions about the system being modelled. Prior assumptions include only: 1) function smoothness (which is often the case for dynamical systems), and 2) time-invariant dynamics (most control systems falling into this category). GPs can model arbitrary continuous functions due to their nonparametric nature. For example, a GP's expressiveness is not limited, but generally increases with more data, generally avoiding underfitting. In addition, a GP's Bayesian nature avoids overfitting, critical for data efficient learning (explained § 2.4). By contrast, polynomials, NNs, and RBFs easily overfit to small datasets and do not provide uncertainty estimates. Nevertheless, polynomials and NNs are already familiar and perhaps analysed more by the control community, adept with different tricks to avoid overfitting such models, such as regularisation and weight-decay. Such problems do not concern Bayesian methods, which naturally regularise models with a prior. GPs are also fast gaining popularity, especially in the robotics literature, resulting in increased theoretical analysis helpful for designing robust controllers. Recent work by Vinogradska et al. (2016) analyses stability with GP dynamics models. In particular, the work determines state space regions where a system using GP dynamics models for finite-time horizon closed-loop control is provably stable.

A downside of GPs is their increased computational complexity (discussed § 2.3.4) compared to NNs and RBFs. *Sparse* GPs instead give approximate predictions with less computational burden. Sparse GPs use a set of M-many *inducing points* to capture most of the GP posterior structure. A sparse GP approximates the true

GP posterior (which uses all N data points) with far fewer inducing points M than data points N. The result is a GP that still learns from entire dataset, but is able to circumvent a costly $\mathcal{O}(N^3)$ training complexity when the approximated function f is simple in some way, generating much redundancy in the N data points. Two common sparse GP methods we use are the Fully Independent Training Conditional (FITC) (Csató and Opper, 2002; Snelson and Ghahramani, 2006) and Variational Free Energy (VFE) (Titsias, 2009). Theoretical and empirical comparisons of FITC and VFE are given by Bui et al. (2016) and Bauer et al. (2016) respectively.

2.3.4 Time Complexity

Our focus on data efficient algorithms naturally leads to training models with small amounts of data. Nevertheless, model scalability is still a concern since 'small' is relative, and other scaling factors such as state dimensionality X are still a concern. Below in Table 2.1 we list of time complexity of each type model previously discussed. Five catagories of model's time complexity are considered. The first is training the model, fitting it to data, which might happen online during an episode or offline between episodes. Second is the complexity of point-predictions given a trained model. Computing the prediction's input-output gradients can incur additional complexity, but is a necessity for trajectory gradient-based optimisation approaches, third column. Forth (continued on the next line of the table), considering input-uncertainty and output-uncertainty when making probabilistic predictions often incurs further computation to handle how a distribution of inputs and outputs. Similarly, probabilistic predictions with gradient information is even more complex, in the fifth column.

Each cell within Table 2.1 is expressed in average-case complexity $\mathcal{O}$ notation. For training data we have points $X \in \mathbb{R}^{N\times(X+U)}$ and targets $y \in \mathbb{R}^{N\times 1}$ where we use N to denote the number of training datum, X is the state-dimensionality, and U is the control dimensionality. The time-complexity of GPs can be traded with accuracy by using a reduced number of inducing points instead M. In deriving GP complexities for Table 2.1 we assume $N \geq M \geq X + U$. For NN predictions given input and out distributions, we use P-many 'particles' to represent distributions (used in Chapter 5). We assume a small, fixed number of NN layers. We use R-many RBF centroids. Note both RBF and NN prediction is highly parallelisable, a property heavy exploited with modern GPUs, but not considered in Table 2.1.

Table 2.1 Dynamic Models' Time Complexity

Model	Training	Point-Prediction	Point-Prediction Gradients
GP	$\mathcal{O}(N^3X)$	$\mathcal{O}(NX(X+U))$	$\mathcal{O}(NX(X+U))$
GP-sparse	$\mathcal{O}(NM^2X)$	$\mathcal{O}(MX(X+U))$	$\mathcal{O}(MX(X+U))$
NN	$\mathcal{O}(N(X+U)^2)$	$\mathcal{O}((X+U)^2)$	$\mathcal{O}((X+U)^3)$
RBF	$\mathcal{O}(NRX(X+U))$	$\mathcal{O}(RX(X+U))$	$\mathcal{O}(RX(X+U))$
		Prob. Prediction	Prob. Prediction Gradients
GP		$\mathcal{O}(N^2X^2(X+U))$	$\mathcal{O}(N^2X^2(X+U)^2)$
GP-sparse		$\mathcal{O}(M^2X^2(X+U))$	$\mathcal{O}(M^2X^2(X+U)^2)$
NN		$\mathcal{O}(P(X+U)^2)$	$\mathcal{O}(P(X+U)^3)$
RBF		$\mathcal{O}(R^2X^2(X+U))$	$\mathcal{O}(R^2X^2(X+U)^2)$

2.4 Data Efficient Learning of Control

Earlier in § 1.2 we mentioned our goal is learning control of dynamical systems in a *data efficient* way. Most control and RL methods are data-intensive, requiring much system interaction before learning good controllers. For systems prone to wear and tear, or expensive to operate, data efficiency is critical. In this section we will describe 1) a definition of data efficiency used throughout this thesis, 2) how to improve data efficiency, and 3) some data efficient algorithms from the literature. Our intention is to help understand the following section, concerning the PILCO algorithm (Deisenroth and Rasmussen, 2011), and why PILCO is a gold-standard for data efficient learning of control. For these reasons, we choose to extend the PILCO algorithm in the following chapters.

In this thesis, we consider learning control over E-many of episodes, each of which executes from timestep $t = 0$ to time horizon $t = T$. The loss of the eth episode is measured as the cost-to-go from the first timestep at $t = 0$, i.e. J_0^e. We judge algorithms based on minimising the 'total loss' $\mathbf{J}$, the summed losses over all episodes:

$$\mathbf{J} \doteq \sum_{e=1}^{E} J_0^e = \sum_{e=1}^{E} \sum_{t=0}^{T} c_t^e. \tag{2.36}$$

The idea is the robot has repeated episodes to interact with a system, learning by trial and error. Better controllers naturally reduce a particular episode's loss. But importantly, the *earlier* a good controller is discovered, the more subsequent episodes there are in which to capitalise from that information, to help reduce the total loss $\mathbf{J}$ in (2.36). We assume online execution of episodes that execute too fast to effectively

learn during an episode. Instead, we assume all learning happens offline *between* episodes.

Two methods to improve data efficiency are: 1) modelling the system's dynamics, and 2) testing different controllers which *might* be optimal (termed the *exploration-exploitation* dilemma in RL (Sutton and Barto, 1998), or *dual-control* in control theory (Wittenmark, 1995)). We will focus on the benefits of modelling only here. Balancing exploration and exploitation is discussed § 4.2.

Using dynamics models to design controllers is generally more data efficient than model-free control due to a model's ability to generalise from limited experience. A model helps to discover and then exploit structure inherent to the control task. In addition, models help *backup* (propagate) loss-estimates globally throughout state-action space given local updates of dynamics knowledge (Atkeson and Santamaria, 1997; Boone, 1997). Model-free methods can require millions of datum to solve even low-dimensional tasks such as the cartpole swing-up (Gu et al., 2016; Lillicrap et al., 2015). Model-based methods by contrast solve similar tasks with only several hundred datum (Deisenroth and Rasmussen, 2011; Pan and Theodorou, 2014). The major cost associated with modelling – especially accurate modelling – is the computational demand. When learning any control tasks, a trade off exists between computational efficiency and data efficiency. Continually retraining a model as each execution generates additional data is a computational burden, which can be especially great for flexible Bayesian models such as the GP, as discussed § 2.3.4. Using the GP model as an example, we can trade off computational efficiency with data efficiency by changing the number of inducing points used in sparse GP approximations. An increase in inducing points, decreases computational efficiency, but increases data efficiency (through more accurate modelling for better informed controller optimisation). As computation efficiency strongly determines temporal efficiency, a model-free or fast-inaccurate-model approach may be the only options when a robot must act constantly in real-time (e.g. infinite time horizon problems, with no chance for offline fitting of computationally heavy models between episodes). As always, incorporating expert domain knowledge helps to increase data efficiency, by reducing the set of plausible dynamics models. Yet we continue to focus on fully autonomous learning: learning from scratch without prior task-specific knowledge of the system dynamics.

Models do not only increase data efficiency for the current task, but also for new tasks. For instance, if one changes a robot's goal by changing the cost function, the dynamics knowledge (which is task-independent) learned from previous tasks

is directly transferable to new tasks, speeding up controller design of new tasks. Model-free approaches by contrast cannot adapt to changing tasks since they only retain knowledge of the loss function which is specific to a particular task.

Assuming we do have the computational resources and time to model dynamics, a common problem we must still address is *model bias*. Model bias typically arises when predictions are based on only a single model selected from a large plausible set of models. By assuming a single model is the true latent dynamics function, controller optimisation becomes susceptible to model errors. This is because selecting any single model – even the Maximum *A Posteriori* (MAP) model – from a large plausible set is quite possibly the wrong model, being just one of the many plausible explanations of what generated the observed data. And the less data observed, the greater the number of plausible dynamics models exist that can generate the observed data. Model-bias is especially problematic when optimising data efficiency, since the robot constantly learns and acts in the low-data regime where the set of plausible models is vast (Deisenroth et al., 2015). As discussed, low-data regimes completely undermine traditional trajectory-based control approaches which rest heavily on an assumption of model-correctness, such as MPC. Unless model-based algorithms consider the complete set of plausible dynamics, they will succumb to model-bias, counteracting the data efficiency benefits of using a model.

Avoiding model bias is made possible with *probabilistic* dynamics models. Probabilistic models make probabilistic predictions by marginalising over the complete set of plausible dynamics functions given the data seen so far. To understand how probabilistic models avoid model bias let us consider how the data constrains the space of plausible models. Since the set of plausible functions $p(f)$ is only constrained to be those which could have generated the current data $\{X, y\}$, the set of plausible functions is unconstrained in state-space regions far from where data has been collected. Thus, in state-space regions far from data, we expect large variation in predictions between models. Therefore, the act of marginalising over all model predictions at points far-from-data naturally results in high-variance (highly uncertain) predictions. Such a property is widely endorsed by the Bayesian community, reflecting 'honest' models whose prediction confidences are commensurate with the amount of data relevant to such a prediction. When the controller optimisation process then use such a probabilistic predictive model, any expected cost of being in states far from data thus has little dependence on the controller. In effect, controller optimisation focuses towards what is known to (probably) improve a controller given data, not on what *might* improve a controller based on unsubstantiated predictions from a

non-probabilistic model far from data. Point-predictions can easily lead controller optimisation astray, especially in unconstrained regions of the state space where the function values are arbitrary.

A data efficient algorithm that does propagate dynamics uncertainty throughout simulation to avoid model-bias is PILCO (Deisenroth and Rasmussen, 2011). As a result, not only does PILCO avoid being led astray by point-predictions from a single model, but by concentrating on policies agreeable to most plausible models, PILCO is more likely to collect data in promising areas of the state space, to help with future controller optimisation, yielding data efficient learning. We maintain PILCO is the unbeaten, gold standard of data efficiency of black-box continuous-state systems and real-time episodes. Although PILCO cannot scale to high-dimensional state-spaces, it does handle arbitrary cost functions unlike LQ based methods, which only use quadratic cost functions to simplify optimisation.

2.5 THE PILCO FRAMEWORK

The Probabilistic Inference and Learning for COntrol (PILCO) algorithm is a model-based policy-search RL algorithm, which achieved unprecedented data efficiency in learning to control the cartpole swing-up problem and others (Deisenroth and Rasmussen, 2011). PILCO applies to continuous-state, continuous-action, and discrete-time control tasks of nonlinear, stochastic and time-invariant dynamical systems. The key to PILCO's success is its probabilistic dynamics model. A probabilistic dynamics model is used to predict single-step system dynamics, from one timestep to the next. This allows PILCO to probabilistically predict multi-step system trajectories over arbitrary time horizon T, by repeatedly using the predictive dynamics model's output at one timestep, as the (uncertain) input in the following

Algorithm 1 PILCO

1: *Define* controller's functional form: $\pi : x_t \times \psi \to u_t$.
2: *Execute* system with random controls one episode to generate initial data.
3: **for** episode $e = 1$ to E **do**
4: *Learn* dynamics model $p(f)$.
5: *Simulate* state trajectories from $p(X_0)$ to $p(X_T)$ using π.
6: *Evaluate* controller: $J(X_{0:T}, \psi) = \sum_{t=0}^{T} \gamma^t \bar{c}_t, \qquad \bar{c}_t = \mathbb{E}_X[\text{cost}(X_t)|\psi]$.
7: *Improve* controller: $\psi \leftarrow \text{argmin}_{\psi \in \Psi} J(\psi)$.
8: *Execute* system, record data: $X_e = [x_{0:T-1}, u_{0:T-1}], \; y_e = x_{1:T}$.
9: **end for**

timestep. For tractability purposes, PILCO uses moment-matching to keep the simulated state distribution $p(x)$ Gaussian. The result is an analytic distribution of state-trajectories, approximated as a marginal set of Gaussian distribution over T states, $p(x_i) \sim \mathcal{N} \; \forall \; i \in [0, T]$. The controller is evaluated using the expected cumulative cost of the trajectories. Next, the controller is improved using local gradient-based optimisation, searching over the controller-parameter space. In RL this is termed policy improvement using policy search. A distinct advantage of moment-matched prediction for policy search instead of particle methods is smoother gradients and fewer local optima (McHutchon, 2014). Finally, the controller is executed, generating additional data to re-train the dynamics model. The whole process then repeats for E-many episodes. For the remainder of this section we discuss, step by step, PILCO summarised by Algorithm 1. The user first defines a parametric controller π function (Algorithm 1, line 1) and then executes an episodes with random controls (line 2) to generate some initial data.

2.5.1 System Execution Phase

With the controller now defined, PILCO is ready to *execute* the system (Algorithm 1, lines 2 and 8). As the systems executes (either as a black-box physics-simulator or a real robot) new data is generated. This training points X_e and training targets y_e during the eth episode are added to the total training data $\{X_{1:e}, y_{1:e}\}$, abbreviated $\{X, y\}$.

Execution begins from a random initial state $x_0 \overset{sample}{\sim} \mathcal{N}\left(\mu_0^x, \Sigma_0^x\right) \in \mathbb{R}^{\mathrm{X}}$. Then, as Fig. 2.1 depicts, the controller π, parameterised by ψ, takes the observed state x_0 as input, outputting control signal $u_0 = \pi(x_0, \psi) \in \mathbb{R}^{\mathrm{U}}$. Applying control u_0 to the dynamical system in state x_0, results in a new system state x_1. Repeating until horizon T results in a new single state-trajectory of data $\{X_e, y_e\}$.

2.5.2 Learning Dynamics

To learn the unknown dynamics (Algorithm 1, line 4), a GP is used. As discussed, a GP does not require much prior dynamics knowledge, only assumptions that dynamics are time-independent and are smooth on some (unknown) scale. Importantly, no task-specific prior knowledge is required. Nevertheless prior knowledge can be included if available. Some methods train models on 'batch' datasets. To maximise data efficiency, PILCO uses all available data $\{X, y\}$.

The latent dynamics function is assumed to have the form $f : \tilde{x}_t \to x_{t+1}$, where $\tilde{x}_t^\intercal \doteq [x_t^\intercal, u_t^\intercal] \in \mathbb{R}^{\mathrm{X}+\mathrm{U}}$. PILCO models the dynamics with X independent GP priors, one for each dynamics output variable: $f^a : \tilde{x}_t \to x_{t+1}^a$, where $a = \{1, ..., \mathrm{X}\}$ is the ath dynamics output, and $f^a \sim \mathcal{GP}(\phi_a^\intercal \tilde{x}, k(\tilde{x}_i, \tilde{x}_j))$. Note we implement PILCO with a linear[1] mean function $\phi_a^\intercal \tilde{x}$. The covariance function k is squared exponential, with length scales $\Lambda_a = \mathrm{diag}([\lambda_{a,1}^2, ..., \lambda_{a,\mathrm{X}+\mathrm{U}}^2])$, and signal variance s_a^2: $k^a(\tilde{x}_i, \tilde{x}_j) = s_a^2 \exp\left(-\frac{1}{2}(\tilde{x}_i - \tilde{x}_j)^\intercal \Lambda_a^{-1}(\tilde{x}_i - \tilde{x}_j)\right)$.

Note the notation of '$p(f)$' is often used to represent a GP prior distribution. However, in this thesis we use '$p(f) = p(f|\mathcal{D})$' interchangeably as the posterior distribution over dynamics functions for succinctness. We have no need to represent the prior distribution in this thesis.

2.5.3 System Simulation Phase

In contrast to executions, PILCO also *simulates* analytic distributions of state trajectories (Algorithm 1, line 5) to evaluate a controller. PILCO simulates offline, between the real online system executions. Simulated control using the dynamics model is identical to real executed control except each control variable is randomised, due to the subjective uncertainty of future state transitions. To distinguish random variables from nonrandom we use capitals: X_t, Y_t, U_t, $\tilde{X}_t$ and X_{t+1}, all of which we approximate as jointly Gaussian. These variables interact both in execution and prediction according to Fig. 2.1. To predict X_{t+1} now that $\tilde{X}_t$ is uncertain PILCO uses the iterated laws of expectation and variance:

$$p(X_{t+1}|\tilde{X}_t) = \mathcal{N}\left(\mu_{t+1}^x, \Sigma_{t+1}^x\right), \tag{2.37}$$

$$\mu_{t+1}^x = \mathbb{E}_{\tilde{X}}\left[\mathbb{E}_f\left[f(\tilde{X}_t)\right]\right], \tag{2.38}$$

$$\Sigma_{t+1}^x = \mathbb{V}_{\tilde{X}}\left[\mathbb{E}_f\left[f(\tilde{X}_t)\right]\right] + \mathbb{E}_{\tilde{X}}\left[\mathbb{V}_f\left[f(\tilde{X}_t)\right]\right]. \tag{2.39}$$

After a one-step prediction from X_0 to X_1, PILCO repeats the process from X_1 to X_2, and so on up to X_T, resulting in a multi-step prediction whose joint we refer to as a distribution over state-trajectories. An advantage of forwards simulating a single Gaussian distribution through time, is that system simulation only scales linearly with horizon. More precise simulation of possible futures a system might have is generally exponential in the horizon given the different true distributions that occur across each branch (conditioned on action and outcome) at each time step, a

[1] The original PILCO instead uses a zero mean function, and instead predicts relative changes in state. We change PILCO here for easier comparison with our extensions of PILCO in later chapters.

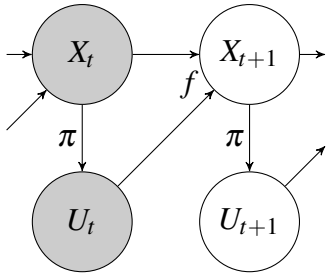


Fig. 2.1 ***PILCO's modelling of control as a probabilistic graphical model (PGM).*** *At each timestep, the observed system state X_t is inputted into the controller function π to decide on a control U_t. Finally, the system evolves to a new state X_{t+1} according to the unknown, nonlinear dynamics function f whose inputs comprise the previous state X_t and control decision U_t. The time series process then repeats. Both the current state X_t and current control U_t are coloured grey at time t, a convention to indicate a node's variable is observed and known to the robot. However, the future state X_{t+1} and future control U_{t+1} are yet to be observed and decided respectively, and thus currently unknown (white) at the present point in time t.*

branching which causes an exponential number of distinct future distributions the state X could be in.

Using identities (B.28) – (B.33) derived in Appendix B.3, both the mean and variance can be computed. The scalar expectation of the ath element of predictive output vector X_{t+1} w.r.t. random input vector $\tilde{X}_t \sim \mathcal{N}\left(\mu_t^{\tilde{x}}, \Sigma_t^{\tilde{x}}\right)$ is:

$$\mu_{t+1}^{x,a} = s_a^2 \beta_a^\intercal q^a + \phi_a^\intercal \mu_t^{\tilde{x}}, \tag{2.40}$$

$$\beta_a \doteq (K_a + \Sigma_a^\varepsilon)^{-1}(y_a - \phi_a^\intercal X), \tag{2.41}$$

$$q_i^a \doteq q(X_i, \mu_t^{\tilde{x}}, \Lambda_a, \Sigma_t^{\tilde{x}}). \tag{2.42}$$

The scalar covariance of the ath and bth elements of predictive output X_{t+1} is (derived Appendix B.3):

$$\begin{aligned}\Sigma_{t+1}^{x,ab} = {} & s_a^2 s_b^2 \left[\beta_a^\intercal (Q^{ab} - q^a q^{b\intercal}) \beta_b \delta_{ab} \left(s_a^{-2} - \mathrm{tr}((K_a + \Sigma_a^\varepsilon)^{-1} Q^{aa})\right)\right] + \\ & + C_{\tilde{x}x'}^{a\intercal} \Sigma_t^{\tilde{x}} \phi_b + \phi_a^\intercal \Sigma_t^{\tilde{x}} C_{\tilde{x}x'}^b + \phi_a^\intercal \Sigma_t^{\tilde{x}} \phi_b,\end{aligned} \tag{2.43}$$

$$Q_{ij}^{ab} \doteq Q\left(X_i, X_j, \Lambda_a, \Lambda_b, 0, \mu_t^{\tilde{x}}, \Sigma_t^{\tilde{x}}\right), \tag{2.44}$$

$$C_{\tilde{x}x'}^a = s_a^2 (\Lambda_a + \Sigma_t^{\tilde{x}})^{-1} (X - \mu_t^{\tilde{x}}) \beta_a^\intercal q^a. \tag{2.45}$$

For data efficient learning, a simulator needs to be as *faithful* to reality as possible. Because the system executes closed loop control, the simulator should simulate closed loop control. Methods, such as MPC, are mostly inconsistent here, simulating open loop control (by anticipating a sequence of state-control sequence over

several time steps, optimising the control at each time step, whilst ignoring dynamics uncertainty from one step to the next, thus simulating an open-loop control), whilst executing closed loop control. Data inefficiencies arise in such cases due to a controller being optimised for a different setup than what is used.

2.5.4 Controller Evaluation

To evaluate the controller π (more specifically the controller parameters ψ) (see Algorithm 1, line 6), PILCO computes the loss as a function of the simulated set of state distributions $X_{0:T}$ and policy parameters ψ:

$$J(X_{0:T}, \psi) \quad = \quad \sum_{t=0}^{T} \gamma^t \bar{c}_t, \quad \bar{c}_t \; = \; \mathbb{E}_X \left[\text{cost}(X_t) | \psi \right]. \tag{2.46}$$

As PILCO's loss function is only a function of *expected* cumulative costs, (2.46) is simplified as the accumulation of each timestep's expected cost $\bar{c}_t$. The choice of cost function is arbitrary, but often chosen by PILCO's original authors to be a saturating function, bounded in range $[0, 1]$. Since optimal controllers are invariant to affine transformations of the cost function (scaling and translation), the choice of bounds 0 and 1 is without loss of generality to other bounded intervals. Advantages of saturating functions are 1) indirectly promoting exploration, and 2) avoiding 'unnecessarily large' costs (Deisenroth et al., 2015). Large unbounded costs that LQ control methods use can be counterproductive to learning good controllers since the optimisation procedure constantly avoids (and may never explore) controller parameterisations predicted to incur arbitrarily high cost. Rarely does it make sense that a system could perform 'arbitrarily bad' without bound. For many control tasks, a system either performs satisfactory, or poorly, but not 'poorly without bound'. Such unbounded costs can needlessly become the sole focus of the controller optimisation, instead of simply discovering a satisfactory controller in a large space mostly unsatisfactory controllers.

2.5.5 Controller Improvement

Given controller evaluation $J(X_{0:T}, \psi)$, which depends on controller parameters ψ, PILCO optimises the controller parameters using the analytic gradients of the loss function (Algorithm 1, line 7). A BFGS optimisation method searches over the continuous space of controller parameters $\psi \in \Psi$ that minimise the total cost $J(X_{0:T}, \psi)$ using gradient $\mathrm{d}J/\mathrm{d}\psi$. Computing $\mathrm{d}J/\mathrm{d}\psi$ requires derivatives $\mathrm{d}\bar{c}_t/\mathrm{d}\psi$ at

each time t to chain together (Deisenroth and Rasmussen, 2011):

$$\frac{\mathrm{d}J(X_{0:T},\psi)}{\mathrm{d}\psi} = \sum_{t=0}^{T}\gamma^t\frac{\mathrm{d}\bar{c}_t}{\mathrm{d}\psi}, \qquad \bar{c}_t = \mathbb{E}_X\left[\mathrm{cost}(X_t)|\psi\right] \tag{2.47}$$

$$= \sum_{t=0}^{T}\gamma^t\left(\frac{\partial\bar{c}_t}{\partial\mu_t^x}\frac{\mathrm{d}\mu_t^x}{\mathrm{d}\psi}+\frac{\partial\bar{c}_t}{\partial\Sigma_t^x}\frac{\mathrm{d}\Sigma_t^x}{\mathrm{d}\psi}\right), \quad \text{where} \tag{2.48}$$

$$\frac{\mathrm{d}\mu_t^x}{\mathrm{d}\psi} = \frac{\partial\mu_t^x}{\partial\mu_{t-1}^x}\frac{\mathrm{d}\mu_{t-1}^x}{\mathrm{d}\psi}+\frac{\partial\mu_t^x}{\partial\Sigma_{t-1}^x}\frac{\mathrm{d}\Sigma_{t-1}^x}{\mathrm{d}\psi}+\frac{\partial\mu_t^x}{\partial\mu_{t-1}^u}\frac{\partial\mu_{t-1}^u}{\partial\psi}+\frac{\partial\mu_t^x}{\partial\Sigma_{t-1}^u}\frac{\partial\Sigma_{t-1}^u}{\partial\psi} \tag{2.49}$$

$$\frac{\mathrm{d}\Sigma_t^x}{\mathrm{d}\psi} = \frac{\partial\Sigma_t^x}{\partial\mu_{t-1}^x}\frac{\mathrm{d}\mu_{t-1}^x}{\mathrm{d}\psi}+\frac{\partial\Sigma_t^x}{\partial\Sigma_{t-1}^x}\frac{\mathrm{d}\Sigma_{t-1}^x}{\mathrm{d}\psi}+\frac{\partial\Sigma_t^x}{\partial\mu_{t-1}^u}\frac{\partial\mu_{t-1}^u}{\partial\psi}+\frac{\partial\Sigma_t^x}{\partial\Sigma_{t-1}^u}\frac{\partial\Sigma_{t-1}^u}{\partial\psi} \tag{2.50}$$

and where $p(X_t)\sim\mathcal{N}(\mu_t^x,\Sigma_t^x)$.

By optimising the controller w.r.t. trajectory distribution $p(X_{0:T})$ generated by a distribution of possible dynamics $p(f)$, the controller's performance next episode is locally optimal w.r.t. all the dynamics information which is summarised by the trained dynamics model $p(f)$. Opposed to much of the literature on robust control (§ 2.3.2) PILCO, does not sacrifice data efficiency, and can quantify its uncertainty in performance if we take the variance of the cumulative cost (discussed Chapter 4). Unlike much adaptive control techniques (§ 2.3.2), PILCO is able to anticipate whether a new controller might destabilise a system due to its probabilistic model.

2.5.6 Related Algorithms

The PILCO algorithm builds on, and has inspired, multiple other similar algorithms. Work by Pan and Theodorou (2014) on Probabilistic Differential Dynamic Programming (PDDP) is based on PILCO, and deals with unknown dynamics using a GP model. PDDP makes iterative linearisations of the dynamics model outputs at each time point, specifically linearisation of the output-mean and output-variance w.r.t. the input-mean and input-variance. Using such linearisations, PDDP makes a second order Taylor expansion of the loss function, combined with a quadratic cost function, to approximately compute a locally optimal sequence of controls for the system. Such iterative-linear approximation of the state-distribution together quadratic cost function, PDDP resembles iLQG without observation noise. PDDP slightly underperforms PILCO in data efficiency by 15%-25% on tasks such as the double-cart pole (Pan and Theodorou, 2014). Yet PDDP boasts much greater computational efficiency than PILCO, with a 20+ fold speedup in offline computation between episodes, even though PDDP still requires computationally expensive gradient information (briefly

discussed § 2.5.5). Presumably this is because of the LQ formulation, in which the optimiser makes fewer optimisation steps than PILCO before convergence. Disadvantages of such an approach is 1) an unbounded cost (discussed § 2.5.4) which restricts exploration, and 2) restriction to iterative-linear functional forms of the controller which are partially open-loop controllers as explained in § 2.2.5.

Further work by Pan et al. (2015) is not fully black-box, and requires an expert to supply some dynamics information. Although Pan et al. (2015) do not learn a controller that runs in real time, Pan et al. (2015) find the control signal at each 0.02s timestep requiring re-optimising the GP, and forward-backward sweeps until convergence. This certainly improves data efficiency by updating the model more often (*during* episodes in addition to between episodes), at the expense of not being able to execute the system in real time. PILCO by contrast handles arbitrary differentiable cost functions (importantly saturating costs functions) unlike many control papers which rely on quadratic cost functions to simplify the optimisation procedure (even becoming convex under linear dynamics).

2.6 DISCUSSION

In this chapter, we discussed control methods relevant to understanding novel contributions made in subsequent chapters. We used a common notation to help compare ideas originating from different academic fields, particular control and reinforcement learning. Before concluding this chapter we revisit the general equations of control, and the restricted form used for the remainder of this thesis. In § 2.1 we discussed the process of control, and a general set of control equations covering most of the literature (2.1)–(2.6), copied below for the reader's convenience (2.51)–(2.56):

General equations of discrete-time control:

$$x_0 = \varepsilon^{x_0} \in \mathbb{R}^{\mathrm{X}}, \tag{2.51}$$

$$x_{t+1} = f(x_t, u_t, t, \varepsilon_t^x) \in \mathbb{R}^{\mathrm{X}}, \tag{2.52}$$

$$y_t = g(x_t, u_t, t, \varepsilon_t^y) \in \mathbb{R}^{\mathrm{Y}}, \tag{2.53}$$

$$u_t = \pi(y_{0:t}, u_{0:t-1}, t, \varepsilon_t^u) \in \mathbb{R}^{\mathrm{U}}, \tag{2.54}$$

$$c_t = \mathrm{cost}(x_t, u_t, x_{t+1}, t, \varepsilon_t^c) \in \mathbb{R}, \tag{2.55}$$

$$J_t = \textstyle\sum_{\tau=t}^{T} \gamma^{\tau-t} \mathbb{E}[c_\tau] \in \mathbb{R}. \tag{2.56}$$

In the following chapters, we make additional assumptions on the system to be controlled. Like PILCO we only consider continuous-state, continuous-control, discrete-time, finite-time-horizon control tasks. We additionally consider continuous-observations y. We also use a *belief state* b in (2.60), as found in the POMDP literature, a sufficient statistic of the probability of the system being in some state x (from the robot's or an observers point of view) given all previous observations and control outputs. Beliefs thus exist in the space of Gaussian probability distributions over X dimensional spaces: $\mathcal{N}^{\mathrm{X}}$. The belief's compact representation is only possible given the Markov property of the state x. Thus we use this sufficient statistic as input to a controller. However, we simplify by only inputting the expected value of the belief distribution, an approximation which we discuss next chapter.

Compared to the general set of control equations above, we only consider a restricted set of equations to describe our system and controller:

Restricted equations of control we consider in Chapters 3–5:

$$
\begin{aligned}
x_0 &= \varepsilon^{x_0} && \in \mathbb{R}^{\mathrm{X}} && , \ \varepsilon^{x_0} \overset{iid}{\sim} \mathcal{N}(\mu_0^x, \Sigma_0^x), && (2.57)\\
x_{t+1} &= f(x_t, u_t) + \varepsilon_t^x && \in \mathbb{R}^{\mathrm{X}} && , \ \varepsilon_t^x \overset{iid}{\sim} \mathcal{N}(0, \Sigma_x^\varepsilon), && (2.58)\\
y_t &= x_t + \varepsilon_t^y && \in \mathbb{R}^{\mathrm{X}} && , \ \varepsilon_t^y \overset{iid}{\sim} \mathcal{N}\left(0, \Sigma_y^\varepsilon\right), && (2.59)\\
b_{t|t} &= p(x_t | y_{0:t}, u_{0:t-1}) && \in \mathcal{N}^{\mathrm{X}} && , && (2.60)\\
u_t &= \pi(\mathbb{E}_{b_{t|t}}\left[b_{t|t}\right]) && \in \mathbb{R}^{\mathrm{U}} && , && (2.61)\\
c_t &= \mathrm{cost}(x_t) && \in [0,1] && , && (2.62)\\
J_t &= \textstyle\sum_{\tau=t}^{T} \gamma^{\tau-t} \mathbb{E}\left[c_\tau\right] && \in [0, T-t+1] && . && (2.63)
\end{aligned}
$$

We shall now discuss the assumptions behind the restricted set of control equations (2.57)–(2.63), compared to the general control equations (2.51)–(2.56). We list the assumption in order by decreasing severity below.

1. Our most severe and limiting assumption is perhaps our observation function, reflected in restrictive form of our observation function in (2.59), simply a function of the latent state x with additive Gaussian observation noise ε_t^y, compared to (2.53). The only unknown is the observation noise variance Σ_y^ε, which must be learned from data. By assuming this simple and known structure in the observation function, the burden is placed on the user to 1) define the state variables within vector x the robot needs to track, 2) preprocess sensory data to

measure each state variable. Also, observation noise ε_t^y being independent of state x and control u is not always the case. For example, a camera's precision in measuring an object's position could be largely dependent on the object's velocity, due to blurring effects.

2. A medium-level limitation is the controller being a function of the belief-mean only, (2.61). It could condition on the belief variance also, or other statistics of the belief distribution. Adapting a controller to condition on variance is not difficult, but not trivial either. It involves additional chain terms in the controller optimisation process. The consequences of how detrimental such a restriction is unclear to the authors, however, as seen later in § 3.3.6, the optimal control of an inverted pendulum does depend on how certain the robot is about the state. Lower state uncertainty allows the robot to be aggressive, applying high controller gains to stabilise the system quickly, whilst higher state uncertainty warrant more cautious controllers slower to react.

3. A weaker-level assumption concerns our dynamics model in (2.58). We focus on the fairly general setting of unknown and nonlinear dynamics f w.r.t. to both input x and u as does (2.52), except with additive process noise ε_t^x. As discussed, nonlinear dynamics makes controller design more difficult, and we make no prior assumptions on our modelling of the latent dynamics f except for function smoothness (on some unknown scale) and time-invariance. Additional Gaussian process noise ε_t^x being independent of the state and control, whilst not general, will satisfy any weakly-stochastic systems and will arguably approximate well most stochastic systems.

4. We also make some weak assumptions for sake of simplicity, which are not difficult to avoid. For example, our time-invariant controller (2.61) is easily changed into a time-variant controller (2.54) since 'time is known in advance'. Such a controller would be implemented with a set of time-dependent controller parameters to optimise, which only requires a slightly more complex chain rule expression than (2.50)–(2.50).

5. Another weak assumptions, for example, the cost function we consider (2.62) is only a function of the state x, but it is trivial to generalise to (2.55) being a function of the control u and time t and new state x_{t+1} also. Note, as previously discussed, optimal control is invariant to affine transformations to the cost function, so a bounded cost function $[0, 1]$ is still very general.

CHAPTER 3

LEARNING CONTROL WITH A FILTER

PILCO is an RL algorithm (discussed § 2.5) which uses GPs to learn a model of the system dynamics of continuous states. The method has shown to be highly data-*efficient* in the sense that it can learn with only very few interactions with the real system. However, a serious limitation of PILCO is that it assumes that the observation noise level is small. There are two main reasons, which make this assumption necessary. Firstly, the dynamics are learnt from the noisy observations (i.e. incorrectly modelling a non-Markov process $f : y_t \times u_t \to y_{t+1}$ as if it were Markov). Learning the dynamics model in this way does not correctly account for the noise in the observations (the true dynamics is $f : x_t \times u_t \to x_{t+1}$). Only if the observation noise ε_t^y is small, then observations $y_t = x_t + \varepsilon_t^y \approx x_t$ would be good approximations for input to the real dynamics function. Secondly, PILCO uses the noisy observation directly to calculate the control, $u_t = \pi(y_t) = \pi(x_t + \varepsilon_t^y)$, which is problematic if the observation noise ε_t^y is substantial. Imagine a controller π controlling an unstable system, where high gain feed-back is necessary for good performance. Observation noise is *amplified* when the noisy input is fed directly to the high gain controller, which in turn injects noise back into the state, creating cycles of increasing variance and instability.

In this chapter we extend PILCO to address these two shortcomings, enabling PILCO to be used in situations with substantial observation noise. The first issue is addressed using the so-called *direct* method for training the dynamics model, explained § 3.3.2. The second problem can be tackled by *filtering* the observations. One way to look at this is that PILCO does planning in observation space, rather than in belief space. In this chapter we extend PILCO to allow filtering of the observations, by combining the previous belief-state distribution with the dynamics model and the observation using Bayes rule to plan in belief space. Note, that this is easily done

when the controller is being applied, but to gain the full benefit of a filter, we have to also take the filter into account when simulating and evaluating the controller.

PILCO trains its controller through minimising the predicted loss when *simulating* the system and controller. Since the dynamics are not known exactly, the simulation in PILCO had to simulate *distributions* of possible trajectories of the physical state of the system. This was achieved using an analytical approximation based on moment-matching and Gaussian state distributions. In this chapter we thus augment the simulation over physical states to also include the state of the filter, an *information state* or *belief state*. This is complicated by the fact that our belief state itself is a probability distribution, we will now have to simulate distributions over distributions. This will allow the algorithm both to apply filtering during control but also to anticipate the effect of filtering during training, thereby learning a better controller.

We will first explore the undesirable effects of noisy observations in § 3.1 before discussing how filtering helps mitigate such effects in § 3.2. In § 3.3 we discuss both how we extend the PILCO framework to apply to a restricted form of POMDPs to instead plan in belief space to include filtering. We show experimental results that the proposed algorithm handles observation noise better than competing algorithms. An assumption is we observe noisy versions of the state variables. We do not handle more general cases where other unobserved states are also learnt nor learn any other mapping from the state space to observations other than additive Gaussian noise. We also introduce the Direct method § 3.3.2 to train a dynamics model from noisy observations. Thereafter we discuss a more generalised version of learning in the presence of observation noise in § 3.4, closing with some additional topics and conclusions.

3.1 Consequences of Unfiltered Observations

Previously, we saw PILCO model a noiseless control process using a Probabilistic Graphical Model (PGM), Fig. 2.1 on page 37. A PGM depicts conditional-dependency relationships between random control variables (distinguished as capitals), useful for simulating the systems forwards in time during controller evaluation, since the value of future control variables is currently uncertain (due to system stochasticities and subjective uncertainty about the dynamics). A PGM also makes clear which variables are observed by the robot (highlighted grey) and which variables are unobserved (termed *hidden* or *latent*) by the robot (highlighted white).

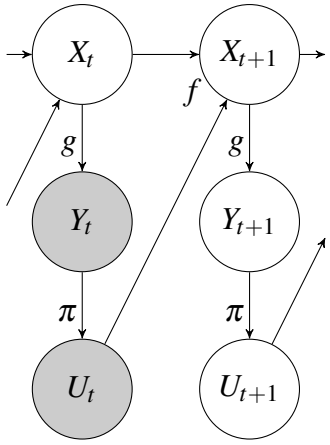


Fig. 3.1 ***Unfiltered control****, the PILCO framework as a probabilistic graphical model extended to model sensor noise. At each timestep, the latent system X_t is observed noisily as Y_t which is inputted directly into controller function π to decide control U_t. Finally, the latent system will evolve to X_{t+1}, according to the unknown, nonlinear dynamics function f of the previous state X_t and control U_t.*

Here, we adapt PILCO's PGM from Fig. 2.1 to explicitly account for noisy observations shown Fig. 3.1. Doing so, we can analyse and predict the effects that sensor noise has on the control process. The key difference between both PGMs is the system states X_t were fully observed (highlighted grey) in Fig. 2.1, whereas now the system states are latent (highlighted white) in Fig. 3.1. Since the robot in Fig. 3.1 now cannot access X_t directly (which would be preferable, being the uncorrupted state information), the unfiltered control process instead uses the observation Y_t (a corrupted version of X_t) as input into controller π to decide control U_t. The system dynamics f is unchanged from Fig. 2.1. The new state X_{t+1} is still a function of the previous state X_t (now unknown to the robot, white) and control U_t (still known to the robot, grey).

Modelling unknown (white) variables in PGMs can be a source of confusion. For example, is it legitimate for the simulator to 'know' latent variables $X_{0:T}$ for the purposes of simulating a system forwards given that the robot cannot know $X_{0:T}$ in reality? In addition, is it legitimate that the simulator passes such 'unknowable information' of $X_{0:T}$ to the cost function for the purpose of controller evaluation (Algorithm 1, line 6)? The answer to both questions is 'yes' – provided that the *simulated* robot does not base decisions on information that the *real* robot cannot know. Simulation of Fig. 3.1 takes the place of reality, generating sequential latent variables $X_{0:T}$, each of which generates an observation Y_t according to the sensor model g. Note in Fig. 3.1, the controller's input is grey (i.e. knowable by the real robot). Fig. 3.1 is thus consistent with reality, and legitimate for simulation. Such

consistency is important for accurate controller evaluation, $J(X_{0:T}, \psi)$. The same restriction does not apply to the dynamics function f, which can have inputs that are latent (X) or not (U). In simulation, we are concerned with how the robot would react to what it can know.

Imperfect noisy sensors impairs learning of control in two important ways. First, training a dynamics model is made more complex with noise on the inputs and outputs of the training data, discussed in § 3.1.1. Second, McHutchon (2014, section 4.5,4.8) noticed that any observation noise is injecting directly into a controller, and can quickly destabilise a system, discussed § 3.1.2.

3.1.1 Effect of Sensor Noise on Modelling Dynamics

The original PILCO algorithm ignored sensor noise when training each GP by assuming each observation y_t to be the latent state x_t. However, this approximation breaks down under significant noise. With only output noise, the GP training is straightforward, and hyperparameter likelihood maximisation will discover the correct noise variance with enough data. However, *input* noise in training data is a problem less studied. Although several recent works by Frigola (2015); McHutchon (2014); McHutchon and Rasmussen (2011) investigate the problem of noisy input data when training GP models in depth.

3.1.2 Effect of Sensor Noise on the Controller

Using noisy observations directly to calculate control outputs is problematic if the observation noise is substantial. Imagine a controller controlling an unstable system, where high gain feed-back is necessary for good performance. Observation noise is *amplified* when the noisy input is fed directly to the high gain controller, which in turn injects noise back into the state, creating a cycle of increasing variance and instability.

Let's be more precise. Consider the task of maintaining the balance of a currently inverted pendulum using the system in Fig. 3.1. Assume $X_t \sim \mathcal{N}(0, \Sigma_t^x)$. Recall the observation noise is modelled as $\varepsilon_t^y \overset{iid}{\sim} \mathcal{N}\left(0, \Sigma_y^\varepsilon\right)$ so the observations themselves are $Y_t = X_t + \varepsilon_t^y \sim \mathcal{N}\left(0, \Sigma_t^x + \Sigma_y^\varepsilon\right)$. We can locally analyse our system using linearisations. A locally approximate gain matrix Π is computed by linearising the controller about the mean input giving a Jacobian matrix $\Pi = \frac{\mathrm{d}\pi(y)}{\mathrm{d}y}|_{y=\mathbf{0}} \in \mathbb{R}^{\mathrm{U}\times\mathrm{X}}$. The variance injected

into dynamics f would now be

$$\mathbb{V}_{X_t}\left[\tilde{X}_t\right] \doteq \mathbb{V}_{X_t}\begin{bmatrix}X_t\\U_t\end{bmatrix} = \begin{bmatrix}\Sigma_t^x & \Sigma_t^x\Pi^\intercal\\ \Pi\Sigma_t^x & \Pi(\Sigma_t^x+\Sigma_y^\varepsilon)\Pi^\intercal\end{bmatrix}. \tag{3.1}$$

Next, we linearise the dynamics about this mean input, with gradient matrices $[A,B] = \frac{\mathrm{d}f(\tilde{x})}{\mathrm{d}\tilde{x}}|_{\tilde{x}=\mathbf{0}} \in \mathbb{R}^{\mathrm{X}\times(\mathrm{X}+\mathrm{U})}$, forming a local linearisation of the unfiltered system Fig. 3.1:

$$x_{t+1} \approx Ax_t + Bu_t + \varepsilon_t^x, \quad \varepsilon_t^x \overset{iid}{\sim} \mathcal{N}(0,\Sigma_x^\varepsilon), \tag{3.2}$$

$$y_t = x_t + \varepsilon_t^y, \quad \varepsilon_t^y \overset{iid}{\sim} \mathcal{N}\left(0,\Sigma_y^\varepsilon\right). \tag{3.3}$$

Finally, the subsequent latent state variance is

$$\begin{aligned}\mathbb{V}_{X_t}[X_{t+1}] &= [A,B]\mathbb{V}_{X_t}\left[\tilde{X}_t\right][A,B]^\intercal + \Sigma_x^\varepsilon\\ &= [A,B]\begin{bmatrix}\Sigma_t^x & \Sigma_t^x\Pi^\intercal\\ \Pi\Sigma_t^x & \Pi(\Sigma_t^x+\Sigma_y^\varepsilon)\Pi^\intercal\end{bmatrix}\begin{bmatrix}A^\intercal\\B^\intercal\end{bmatrix} + \Sigma_x^\varepsilon\\ &= \underbrace{(A+B\Pi)\Sigma_t^x(A+B\Pi)^\intercal}_{\text{system response}} + \underbrace{(B\Pi)\Sigma_y^\varepsilon(B\Pi)^\intercal}_{\text{amplified obs. noise}} + \underbrace{\Sigma_x^\varepsilon}_{\text{process noise}}.\end{aligned} \tag{3.4}$$

Any eigenvalues in (3.4) greater than eigenvalues of Σ_t^x indicate a growth in state-variance. Note (3.4) is a sum of PSD matrices, so all eigenvalues are non-negative.

How does PILCO's controller optimisation stabilise the pendulum upright in light of (3.4)? PILCO finds a balance for the local gain matrix Π, such that Π is 'small enough in magnitude' to avoid injecting too much noise into the system (second term in (3.4)), yet 'negative enough' to respond sufficiently quickly to deviations of the pendulum's upright position (first term in (3.4)). An optimal balance is found by completing the square (which PILCO does not explicitly do, since PILCO optimises cost, not stability), re-expressing (3.4) as:

$$\begin{aligned}\mathbb{V}_{X_t}[X_{t+1}] &= [B\Pi + A\Sigma_t^x(\Sigma_t^x+\Sigma_y^\varepsilon)^{-1}](\Sigma_t^x+\Sigma_y^\varepsilon)[B\Pi + A\Sigma_t^x(\Sigma_t^x+\Sigma_y^\varepsilon)^{-1}]^\intercal\\ &\quad + A\Sigma_t^x(\Sigma_t^x+\Sigma_y^\varepsilon)^{-1}\Sigma_y^\varepsilon A^\intercal + \Sigma_x^\varepsilon,\end{aligned} \tag{3.5}$$

with minimum at $B\Pi = -A\Sigma_t^x(\Sigma_t^x+\Sigma_y^\varepsilon)^{-1}$ if the system is fully actuated ($\mathrm{rank}(\Pi) \geq \mathrm{X}$). Several intuitions towards stabilising an unfiltered system can be gleaned from minimising (3.5):

- Larger observation noises Σ_y^ε warrant smaller gains Π to avoid injecting noise.

- The more 'sensitive' a system is (larger B), the less gain Π is required.
- The more rapid a system naturally moves (larger A), the more gain Π is warranted to control the system.
- The controller gains – transformed through the system response – $B\Pi$ should act *against* any natural runaway motion of the system A, being proportional to the *negative* of A.
- There exist minimum limits to which the state variance can be maintained at. If, in this unfiltered setting (3.5), $A\Sigma_t^x(\Sigma_t^x + \Sigma_y^\varepsilon)^{-1}\Sigma_y^\varepsilon A^\intercal + \Sigma_x^\varepsilon \not\preceq \Sigma_t^x$, then the state variance will grow, regardless of what control gains Π are applied. Because we are using this linearisation, there is no bifurcation point in the above equation where we can increase the parameters Σ_y^ε such that the above equation is unsatisfied for any Σ_t^x (indicating an uncontrollable system). Although in the reality of nonlinear control, Σ_t^x might grow large enough that our linear approximation is inaccurate, the system may well become uncontrollable, which our analysis cannot detect. We also note an increasing noise variance Σ_y^ε increases the state variance Σ_t^x. So unfiltered control is suitable under low observation noise. Another method is to increase the observation sampling rate. Since we used discrete timesteps, the effect of the observation sampling rate is not immediately clear. An increased sampling rate effectively reduces the process noise Σ_x^ε between (shortened) timesteps. E.g. a zero-order-hold controller updates at 10Hz, yet we may wish to sample at 1000Hz if possible, taking an average of the iid random observations. However, in our experiments we assume 30Hz, being the typical maximum frame rate of a cheap camera at maximum resolution.

We shall revisit our analysis of system stability after introducing filtering in the following section, to understand how filtering can help improve system stability.

3.2 Filtering Observations

As an alternative to directly inputting noisy observations into a controller, the observations can be *filtered* (at execution time) before being inputted to the controller. To *filter* a sequence of observations is to infer a belief posterior distribution over the latent system state conditioned on the complete history of previous control outputs and observations received so far. To put filtering into context, some examples

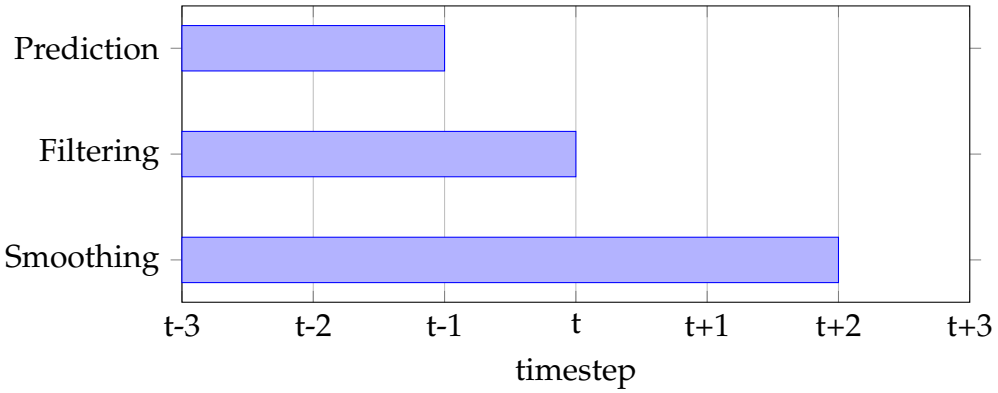


Fig. 3.2 Prediction, filtering, and smoothing: three types of probabilistic inference in time series models about latent state x_t. The bar symbolises the amount of measured data up until (and including) a particular timestep.

of different inference procedures are shown Fig. 3.2. To infer the latent state X_t given data up until: (a) a time less than t is prediction, e.g. $b_{t|t-1} = p(x_t|y_{0:t-1})$; (b) time t is filtering, e.g. $b_{t|t} = p(x_t|y_{0:t})$; (c) a time greater than t is smoothing, e.g. $b_{t|t+2} = p(x_t|y_{0:t+2})$. The dual subscript, using example (c), means belief of state x at time t given all observations up until time $t+2$ inclusive.

Implementing a filter is straightforward when the system dynamics are *known* and *linear* and the noise is Gaussian, referred to as Kalman filtering (Kalman, 1960). For nonlinear systems, the extended Kalman filter (EKF) is often adequate (Berg et al., 2012), as long as the dynamics are *locally linear*, meaning approximately linear within the region covered by the belief distribution. Otherwise, as we later show, the EKF's first order Taylor expansion approximation breaks down. Greater nonlinearities usually warrant the unscented Kalman filter (UKF, Julier et al. (1995)) or particle filtering (PF, Isard and Blake (1998)) (Ko and Fox, 2009; Ross et al., 2008). The UKF uses a deterministic sampling technique to estimate moments. However, if moments can be computed analytically and exactly, moment-matching methods are preferred. Moment-matching using distributions from the exponential family (e.g. Gaussians) is equivalent to optimising the Kullback-Leibler divergence $\mathrm{KL}(p||q)$ between the true distribution p and an approximate distribution q. In such cases, moment-matching is less susceptible to model bias than the EKF due to its conservative predictions (Bishop, 2006, section 10.7).

Our goal is to learn control of nonlinear systems under significant observation noise, which requires filtering with *unknown* and *locally nonlinear* dynamics.

Such regimes undermine traditional trajectory-based approaches to control, which assume model-correctness when filtering, including iLQGs. However, before

studying the problem of filtering with unknown dynamics, we shall review the Kalman filter.

3.2.1 KALMAN FILTERING

Around 1960, at the same time that optimal control was being defined (Bellman and Kalaba, 1965), Rudolf Kálmán introduced filtering allowing to optimise the *expected* sum of costs for known stochastic systems of linear dynamics, now known as Kalman filtering (Kalman, 1960). Consider the following linear system, the same as LQG setting we saw previously,

$$x_{t+1} = f(x_t, u_t, \varepsilon_t^x) = Ax_t + Bu_t + \varepsilon_t^x, \quad \varepsilon_t^x \overset{iid}{\sim} \mathcal{N}(0, \Sigma_x^\varepsilon), \tag{3.6}$$

$$y_t = g(x_t, u_t, \varepsilon_t^y) = Cx_t + Du_t + \varepsilon_t^y, \quad \varepsilon_t^y \overset{iid}{\sim} \mathcal{N}\left(0, \Sigma_y^\varepsilon\right). \tag{3.7}$$

Our robot begins with a prior belief distribution about the latent state:

$$b_{t|t-1}(X_t) = \mathcal{N}\left(X_t; m_{t|t-1}, V_{t|t-1}\right). \tag{3.8}$$

Kalman filtering generally alternates between two steps: an update step and a prediction step. During the update step, the robot 'updates' its prior belief with observation y_t to yield a posterior belief using Bayes rule. During the prediction step (and before the next observation y_{t+1} arrives), the robot projects its belief posterior of X_t through the known dynamics model to predict the next state X_{t+1}, just as (2.19) did. Often both steps alternate, but this is not strictly necessary. Multiple predict steps can occur in succession if observations are infrequent. Multiple updates steps can also occur if multiple observations are made at the same time. However, a predict step should always proceed an update step if multiple observations have different timestamps. We now describe each step in more detail.

UPDATE STEP

The update step concerns how our prior belief $b_{t|t-1}$ is updated given the observation function g (3.7). Start with robot's subjective prior probability $b_{t|t-1}(x_t) = p(x_t|y_{1:t-1}, u_{1:t-1}) = \mathcal{N}\left(x_t; m_{t|t-1}, V_{t|t-1}\right)$. Get an observation with likelihood $p(y_t|x_t) = \mathcal{N}\left(y_t; Cx_t + Du_t, \Sigma_y^\varepsilon\right)$. Using Bayes rule to combine prior with likelihood:

$$
\begin{aligned}
b_{t|t} &\sim \mathcal{N}\left(m_{t|t}, V_{t|t}\right) && (3.9)\\
&\doteq p(X_t|y_{1:t}, u_{1:t-1}) && (3.10)\\
&= \frac{p(y_t|X_t)\cdot p(X_t|y_{1:t-1}, u_{1:t-1})}{p(y_t)} && (3.11)\\
&\propto \mathcal{N}\left(y_t; \mu_t^y, \Sigma_y^\varepsilon\right)\cdot\mathcal{N}\left(X_t; m_{t+1|t}, V_{t+1|t}\right), && (3.12)
\end{aligned}
$$

where $\mu_t^y = CX_t + Du_t$. Both $m_{t|t}$ and $V_{t|t}$ of (3.9) are the product of two Gaussians (3.12), computed using (A.16) – (A.20). However, for intuition, let us derive $m_{t|t}$ and $V_{t|t}$ here. Given a joint:

$$
\begin{bmatrix} X_t \\ Y_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbb{E}[X_t] \\ \mathbb{E}[Y_t] \end{bmatrix}, \begin{bmatrix} \mathbb{C}[X_t,\, X_t] & \mathbb{C}[X_t,\, Y_t] \\ \mathbb{C}[Y_t,\, X_t] & \mathbb{C}[Y_t,\, Y_t] \end{bmatrix}\right), \tag{3.13}
$$

the conditional Gaussian distribution is:

$$
\begin{aligned}
\mathbb{E}[X_t|Y_t] &= \mathbb{E}[X_t] + \mathbb{C}[X_t,\, Y_t]\,\mathbb{C}[Y_t,\, Y_t]^{-1}(Y_t - \mathbb{E}[Y_t]), && (3.14)\\
\mathbb{V}[X_t|Y_t] &= \mathbb{C}[X_t,\, X_t] - \mathbb{C}[X_t,\, Y_t]\,\mathbb{C}[Y_t,\, Y_t]^{-1}\,\mathbb{C}[Y_t,\, X_t], && (3.15)
\end{aligned}
$$

which the reader may recognise as the familiar posterior mean and variance functions of a GP. By simply swapping in our symbols we've been using we have:

$$
\begin{aligned}
m_{t|t} &= m_{t|t-1} + K^{\text{gain}}(y_t - \mu_t^y), && (3.16)\\
V_{t|t} &= V_{t|t-1} - K^{\text{gain}} C V_{t|t-1}, && (3.17)\\
K^{\text{gain}} &\doteq \underbrace{V_{t|t-1}C^\intercal}_{\mathbb{C}[X_t,\, Y_t]} \cdot \underbrace{(\Sigma_y^\varepsilon + CV_{t|t-1}C^\intercal)^{-1}}_{\mathbb{C}[Y_t,\, Y_t]^{-1}}, && (3.18)
\end{aligned}
$$

where K^{gain} is the Kalman gain term. The Kalman gain represents the relative weight (or 'trust') we apply to observation y_t compared to prior belief $m_{t|t}$. The posterior belief-mean $m_{t|t}$ in (3.16) is equal to the prior belief-mean $m_{t|t-1}$ with an adjustment. The adjustment is the observation's y_t deviation from its expected value, $\mu_t^y = Cm_{t|t-1} + Du_t$, weighted by the Kalman gain K^{gain}. High gains place more emphasis on observations, creating more responsive but also noisy signals $m_{t|t}$ over time since any noise in the observation is amplified by factor K^{gain}. Low gains trust the prior prediction of the state $m_{t|t-1}$ more than observation y_t, creating a filter slow to respond but also smoothly changing signal $m_{t|t}$. Zero gain corresponds to open-loop control and dead-reckoning.

Notice that such belief monitoring is tractable since the output distribution is still Gaussian, with constant time complexity w.r.t. time t. For instance, the posterior $b_{t|t}$ in (3.9) which by definition equals $p(X_t|y_{1:t}, u_{1:t-1})$ can in fact be written as a linear combination of the current observation and the prior in (3.12), where the prior is a sufficient statistic of all previous observations $y_{1:t-1}$. The benefit is Kalman filtering need not store a continually-expanding history of $y_{1:t}$ and $u_{1:t}$, remaining computationally cheap.

Prediction Step

The prediction step maintains the belief function b given an understanding of the dynamics f in (3.6). Using Appendix A.1.2 we have:

$$b_{t+1|t} \sim p(X_{t+1}|y_{1:t}, u_{1:t}) = \mathcal{N}\left(m_{t+1|t}, V_{t+1|t}\right), \tag{3.19}$$

$$m_{t+1|t} = Am_{t|t} + Bu_t, \tag{3.20}$$

$$V_{t+1|t} = AV_{t|t}A^\intercal + \Sigma_x^\varepsilon. \tag{3.21}$$

Note V is not a function of observations y. Thus, all future values V conditioned on time are deterministic, and can be precomputed ahead of time. Also note m is a function of previous m but not V, a property known as *certainty equivalence* (Bar-Shalom and Tse, 1974). Similarly, V a function of previous V but not m. The certainty equivalence principle only applies to *linear* dynamical systems.

*Optimal Estimation

Kalman filtering is also known for 'optimal state estimation', with 'optimal' meaning outputting point-estimates of the state that minimise the 'least squares estimate' (expected quadratic error). Here we revisit the Kalman filtering update step, showing the same K^{gain} can alternatively be derived as the gain that yields the least squares state estimate.

The point-estimate in this case is $M_{t|t}$, which minimises $\mathbb{E}\left[(x_t - M_{t|t})^\intercal (x_t - M_{t|t})\right]$, assuming our prior is an unbiased estimators of x_t, i.e. if $M_{t|t-1} \sim \mathcal{N}\left(x_t, V_{t|t-1}\right)$ (capitalised to signify randomised variables), and additionally that $M_{t|t-1}$ is uncorrelated with $Y_t \sim \mathcal{N}\left(Cx_t + Du_t, \Sigma_y^\varepsilon\right)$. First note $M_{t|t}$ is an unbiased estimator for any gain K:

$$\begin{aligned}
\mathbb{E}\left[x_t - M_{t|t}\right] &= \mathbb{E}\left[x_t - M_{t|t-1} + K(Y_t - CM_{t|t-1} - Du_t\right] \\
&= \mathbb{E}\left[x_t - x_t + K(Cx_t + Du_t - Cx_t - Du_t)\right] \\
&= 0.
\end{aligned} \tag{3.22}$$

Since $M_{t|t}$ is an unbiased estimator of x_t, we proceed in deriving the gain which minimises the quadratic error of our state estimate, K^{optimal}:

$$\begin{aligned}
K^{\text{optimal}} &\doteq \operatorname*{argmin}_K \mathbb{E}\left[(x_t - M_{t|t})^\intercal (x_t - M_{t|t})\right] \\
&= \operatorname*{argmin}_K \mathbb{V}\left[M_{t|t}\right] \\
&= \operatorname*{argmin}_K \mathbb{V}\left[(I - KC)M_{t|t-1} + K(Y_t - Du_t)\right] \\
&= \operatorname*{argmin}_K (I - KC)V_{t|t-1}(I - KC)^\intercal + K\Sigma_y^\varepsilon K^\intercal \\
&= \operatorname*{argmin}_K K(\Sigma_y^\varepsilon + CV_{t|t-1}C^\intercal)K^\intercal - KCV_{t|t-1} - V_{t|t-1}C^\intercal K^\intercal \\
&= \operatorname*{argmin}_K (K - V_{t|t-1}C^\intercal(\Sigma_y^\varepsilon + CV_{t|t-1}C^\intercal)^{-1})(K - V_{t|t-1}C^\intercal(\Sigma_y^\varepsilon + CV_{t|t-1}C^\intercal)^{-1})^\intercal \\
&= V_{t|t-1}C^\intercal(\Sigma_y^\varepsilon + CV_{t|t-1}C^\intercal)^{-1} \\
&= K^{\text{gain}}. \qquad (3.23)
\end{aligned}$$

However, we argue there is no reason why optimising the state's *quadratic* error is especially meaningful. Instead, the importance of the Kalman gain's definition in (3.18) is being the gain corresponding to correct Bayesian inference of the state x_t given a robot's prior belief $b_{t|t-1}$ and observational likelihood in y_t.

Extended Kalman Filter

The Kalman filter was applicable to linear systems only. When either the dynamics function f or sensor function g are nonlinear, the Kalman update and prediction steps can instead be approximated using linearisation of both functions. Assuming f and g are differentiable, then a first order Taylor expansion of f and g is taken about some working point (e.g. the input mean). Such approximate filtering is known as Extended Kalman Filtering (EKF, Gelb (1974)). The EKF is a standard filtering technique in nonlinear control.

EKF's linearisations are similar to the iLQG in (2.24), with each matrix in (3.6) – (3.7) indexed by time:

$$A_t = \frac{\partial f}{\partial x}\Big|_{m_{t|t},u_t}, \qquad B_t = \frac{\partial f}{\partial u}\Big|_{m_{t|t},u_t}, \qquad (3.24)$$

$$C_t = \frac{\partial g}{\partial x}\Big|_{m_{t|t},u_t}, \qquad D_t = \frac{\partial g}{\partial u}\Big|_{m_{t|t},u_t}, \qquad (3.25)$$

which are used to linearise the dynamics function and observation function at each timestep:

$$x_{t+1} \approx A_t x_t + B_t u_t + \varepsilon_t^x, \tag{3.26}$$

$$y_t \approx C_t x_t + D_t u_t + \varepsilon_t^y. \tag{3.27}$$

Work by Berg et al. (2012); van den Berg et al. (2012) for example find locally-optimal controllers in continuous-state POMDPs using an analytic framework of filtering distributions over Gaussian-beliefs using an EKF and assuming known dynamics. Their analytic framework must reason about uncertain future observations (due to process and observation noises in (3.24) – (3.25)), which induces uncertainty in beliefs and control outputs, similar to PILCO. Using restrictions including a linear-in-the-vectorised-belief controller, and loss function J that must be quadratic in belief-mean and linear in vectorised belief-variance, a locally-optimal controller is found. The approximation assumes locally valid around deviations from a nominal belief-trajectory created from a noise-free deterministic simulation.

The EKF, although common, does not come with strong guarantees. Due to EKF's linear approximations of nonlinear functions, an EKF is not guaranteed to be stable, nor an optimal estimator (Gelb, 1974). If the initial state estimate is wrong, or models of f and g contain small errors, the state estimate $m_{t|t-1}$ can rapidly diverge.

3.2.2 UNSCENTED KALMAN FILTER

Unscented Kalman filtering (UKF, Julier et al. (1995)) is an alternative choice to EKF for nonlinear filtering, better suited to more nonlinear functions. The UKF's prediction step represents the input distribution $b_{t|t} \sim \mathcal{N}\left(m_{t|t}, V_{t|t}\right)$ by a set of $2\mathrm{X}+1$

weighted deterministic samples:

$$\mathcal{X}_0 = m_{t|t}, \tag{3.28}$$

$$\mathcal{X}_i = m_{t|t} + \left(\sqrt{(\mathrm{X}+\lambda)V_{t|t}}\right)_i, \qquad i = 1,...,\mathrm{X}, \tag{3.29}$$

$$\mathcal{X}_i = m_{t|t} - \left(\sqrt{(\mathrm{X}+\lambda)V_{t|t}}\right)_{i-\mathrm{X}}, \qquad i = \mathrm{X}+1,...,2\mathrm{X}, \tag{3.30}$$

$$W_0^s = \frac{\lambda}{\mathrm{X}+\lambda}, \tag{3.31}$$

$$W_0^c = \frac{\lambda}{\mathrm{X}+\lambda} + 1 - \alpha^2 + \beta, \tag{3.32}$$

$$W_i^s = W_i^c = \frac{\lambda}{2(\mathrm{X}+\lambda)}, \tag{3.33}$$

where $(A)_i$ mean the i'th column of a matrix A. Each sample is projected through the nonlinear function. The output distribution's moments, $b_{t+1|t} \sim \mathcal{N}\left(m_{t+1|t}, V_{t+1|t}\right)$ are then estimated using the empirical moments of the projected weighted samples:

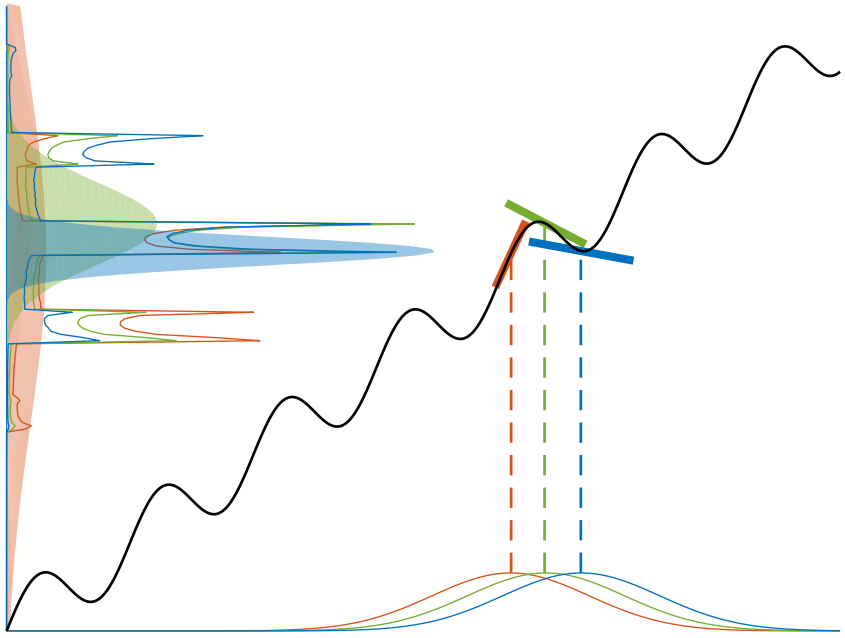
$$m_{t+1|t} \approx \sum_{i=0}^{2\mathrm{X}} W_i^s \mathcal{X}_i, \tag{3.34}$$

$$V_{t+1|t} \approx \sum_{i=0}^{2\mathrm{X}} W_i^c [\mathcal{X}_i - m_{t+1|t}][\mathcal{X}_i - m_{t+1|t}]^\mathsf{T}. \tag{3.35}$$

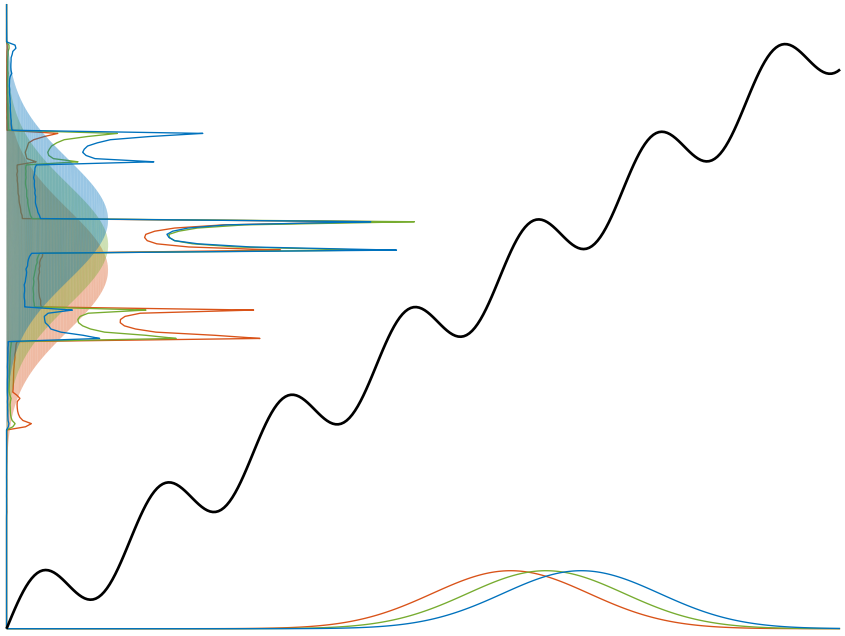
For further details on the (similar) update step and typical values of values α, β, and λ see Julier et al. (1995). Three advantages of the UKF over the EKF are 1) greater robustness to highly nonlinear functions, 2) avoids computing the (sometimes expensive) Jacobian, and 3) applicability to non-differentiable functions.

3.2.3 Assumed Density Filtering

Assumed Density Filtering (ADF, Maybeck (1982)) is yet another alternative to EKF and UKF for nonlinear filtering. The term ADF is refers to the prediction step only, but this thesis also uses 'ADF' to refer to a full filtering process, with identical update step as the EKF, and a different prediction step. PILCO used ADF-prediction to forwards predict Gaussian state distributions. An advantage of ADF with GPs, is that the moments of GP predictions given uncertain Gaussian inputs and a squared exponential covariance function are exactly computable analytically (Candela et al., 2003). This makes GP dynamics models especially useful, allowing PILCO to use ADF tractably in simulation.



(a) Extended Kalman Filter (EKF)



(b) Assumed Density Filter (ADF)

Fig. 3.3 ***Comparison of Filters' Prediction Step.*** *The EKF, Fig. 3.3a, uses linearisations about a working point to approximate the nonlinear function (black). We show three Gaussian input distributions along the horizontal axis (red, green, and blue), each with slightly different mean locations. Usually the working point is chosen as the input mean, shown by dashed lines. Note the EKF has very different predictive distributions (colour-filled Gaussian distributions on the vertical axis), each very sensitive to their working point locations. An outline of each true output distribution is also shown. In contrast, the ADF Fig. 3.3b yields three similar output distributions, not overly sensitive to the input mean locations, since ADF uses both input mean and input variance information. Note the three ADF output distributions even capture the upwards trend of the nonlinear function (black), as the red-green-blue input order is preserved along the output axis.*

In Fig. 3.3 we compare the EKF prediction step Fig. 3.3a with the ADF Fig. 3.3b. Previously we caveated the EKF's suitability only if the dynamics model (black line, assumed certain for our simple example) is *locally linear*, meaning approximately linear within the belief distribution's span. The EKF is ill suited for models that are not locally linear, where slight changes in selecting a working point from within the input distribution produce widely-varying linearisations and thus outputs, seen Fig. 3.3a. In Fig. 3.3a we use the input mean as a working point at which the Jacobian is computed. By shifting the input distribution between the red-green-blue inputs, notice the outputs change dramatically, being overly-sensitive to working point location. By contrast, ADF does not suffer from such sensitivity, seen Fig. 3.3b, and remains suitable for locally nonlinear models.

The ADF's moment matching is often compared to the Kullback–Leibler (KL) divergence. Integration of a distribution from family q through a non-linear function generally produces distributions p of different families than q (Bishop, 2006, section 10.7). The ADF projects the output distribution p back into a common tractable form q using moment matching. In Fig. 3.3b the output (vertical axis) contrasts the moment-matched outputs (filled-colour Gaussians, denoted q), with the true, intractable distribution (coloured-outlines, non-filled plots, denoted p). Moment matching where q is any exponential distribution (e.g. Gaussian) is equivalent to minimising the KL divergence: $\text{argmin}_q \text{KL}(p||q)$ (Barber et al., 2011, section 1.5.2), where $\text{KL}(p||q) \doteq \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \text{d}x$. By minimising $\text{KL}(p||q)$, the approximating distribution q has the interpretation in machine learning of 'spreading out' more than p, attempting to 'allocate comparable probability mass wherever p has probability mass'. The converse is not true, q may allocate probability mass where p has none. In other areas of machine learning such as variational methods, the reciprocal similarly measure is more familiar, $\text{argmin}_q \text{KL}(q||p)$, in which q might fit to only one mode of p. In Fig. 3.3b we can notice three modes in p. By minimising $\text{KL}(p||q)$, ADF acts conservatively, with q spread out more than p to cover each of p's three modes.

An alternative to ADF is numerical quadrature, which can approximate the long term distribution more accurately that ADF (Vinogradska et al., 2016). An open question is whether multi-modality is important to capture, since it is often a precursor to losing control of a system. Later in Chapter 5 we found unimodal modelling was *required* to successful learn control, even though we used MC sampling distributions for the flexibility of multi-model modelling. A comparison of numerical quadrature and ADF for learning to control remains an interesting avenue for future research.

3.3 Filtered Control with Belief-MDPs

The original PILCO algorithm by Deisenroth and Rasmussen (2011) assumes full state observability and can fail under moderate observation noise. One solution is to filter observations during controller execution (Deisenroth and Peters, 2012). However, without also simulating system trajectories w.r.t. the filtering process, the above method merely optimises controllers for unfiltered control, not for filtered control. The mismatch between unfiltered-simulation and filtered-execution restricts PILCO's ability to take full advantage of filtering. Dallaire et al. (2009) instead optimise a controller using a more realistic filtered-simulation (discussed more later). However, the method neglects model uncertainty by only using the MAP model. Unlike Deisenroth and Peters (2012), who give a full probabilistic treatment of the dynamics predictions, work by Dallaire et al. (2009) remains highly susceptible to model errors, hampering data efficiency.

Instead, we propose a method that simulates system trajectories using closed loop filtered control precisely because we execute closed loop filtered control, an important consistency stressed by McHutchon (2014, section 4.5). Our resulting controllers are thus optimised for the specific case in which they are used. Doing so, our method is applicable to tasks with high observation noise whilst retaining the same data efficiency properties of PILCO. To evaluate our method, we use the benchmark cartpole swing-up task as PILCO did, except now with noisy sensors, § 3.3.5. We show in § 3.3.6 that such realistic and probabilistic simulation helps our method outperform the aforementioned methods.

Our method uses the same high-level algorithm as PILCO seen Algorithm 1 on page 34, with the details of each step changing. We modify[1] two subroutines to extend PILCO from MDPs to a special-case of POMDPs (specifically where the partial observability has the form of additive Gaussian noise on the latent state). First, we filter observations during system execution (Algorithm 1, line 8), detailed § 3.3.1. Second, we simulate an analytic distribution of *belief*-trajectories (instead of state-trajectories) through the filter using PILCO's dynamics model (Algorithm 1, line 5), discussed § 3.3.3.

Our implementation continues PILCO's distinction between *executing* the system (resulting in a single real state-trajectory) and *simulating* system responses (resulting in an analytic distribution of predictive state-trajectories). In our case, execution yields a *belief*-trajectory, and simulation yields an analytic distribution of predictive

[1] PILCO's source code is available at http://mlg.eng.cam.ac.uk/pilco/.

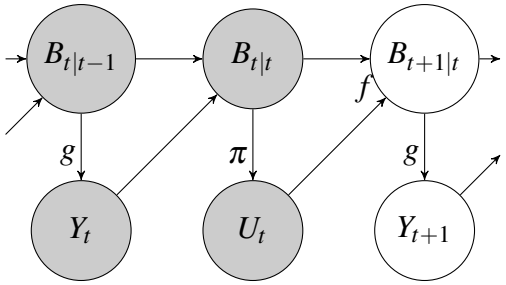


Fig. 3.4 ***Filtered control for belief-MDPs****, an extension of PILCO to Bayesian filtering. Our prior belief $B_{t|t-1}$ (over latent system X_t), generates observation Y_t. The prior belief $B_{t|t-1}$ then combines with observation Y_t resulting in posterior belief $B_{t|t}$ (the update step). Then, the mean posterior belief $\mathbb{E}\left[B_{t|t}\right]$ is inputted into controller function π to decide control U_t. Finally, the next timestep's prior belief $B_{t+1|t}$ is predicted using dynamics model f (the prediction step).*

belief-trajectories, seen in Fig. 3.4. For instance, during the execution phase, the system reads specific observations y_t and decides specific control u_t, which when filtered, results in a single belief state $b \sim \mathcal{N}(m,V)$. The belief b can be treated as a random variable with a distribution, parameterised by belief-mean m and belief-certainty V, summarising the robot's subjective belief about latent state x_t. Note both m and V are functions of previous observations $y_{1:t}$. Consider, during the probabilistic system simulation phase, future observations are *uncertain* (since they are not yet observed), distinguished as Y. Since the belief parameters m and V are functions of the (now-randomised) observations, then m and V are randomised also, distinguished as M and V'. Given the belief (a random variable) is distributed according to parameters, which are themselves also random, the belief can be thought of as *hierarchically*-random, denoted $B \sim \mathcal{N}(M,V')$. Our framework allows us to consider multiple future belief-states *analytically* during controller evaluation. Intuitively, this framework is the analytical analogue of POMDP controller evaluation using particle methods. In particle methods, each particle is associated with a distinct belief, due to each conditioning on different samples of future observations. A particle distribution thus defines a distribution over beliefs. Our method is the analytical analogue of such a particle distribution. By additionally restricting the robot's beliefs as (parametric) Gaussian, we can tractably encode a distribution over beliefs by a distribution over belief-parameters.

3.3.1 System Execution Phase

When an actual filter is applied, it starts with three pieces of information: $m_{t|t-1}$, $V_{t|t-1}$ and a noisy observation of the system y_t.

Filtering Update Step

A filtering update combines prior belief $b_{t|t-1} \sim \mathcal{N}\left(m_{t|t-1}, V_{t|t-1}\right) \doteq p(x_t|y_{1:t-1}, u_{1:t-1})$ with observational likelihood. Given we assumed our observation function was $y_t = x_t + \varepsilon_t^y$, the observational likelihood is $p(Y_t|x_t) = \mathcal{N}\left(x_t, \Sigma_y^\varepsilon\right)$. Both the prior and likelihood are combined using Bayes rule to yield the posterior belief $b_{t|t}$:

$$
\begin{aligned}
b_{t|t} &\sim \mathcal{N}\left(m_{t|t}, V_{t|t}\right) = p(x_t|y_{1:t}, u_{1:t-1}), && (3.36)\\
m_{t|t} &= W_m m_{t|t-1} + W_y y_t, && (3.37)\\
V_{t|t} &= W_m V_{t|t-1}, && (3.38)\\
W_m &= \Sigma_y^\varepsilon (V_{t|t-1} + \Sigma_y^\varepsilon)^{-1}, && (3.39)\\
W_y &= V_{t|t-1}(V_{t|t-1} + \Sigma_y^\varepsilon)^{-1}, && (3.40)
\end{aligned}
$$

with weight matrices W_m and W_y derived using the product of two Gaussians identities (A.10)–(A.12). For the filtering update equations (3.36)–(3.40) we use absolute weight matrices W_m and W_y, 'absolute' meaning $W_m + W_y = I$. The posterior mean $m_{t|t}$ in (3.37) is simply a weighted average between the prior mean $m_{t|t-1}$ and observation y_t. Note, the above *absolute* weights are unlike the *relative* weights commonly used in Kalman filtering (§ 3.2.1), i.e. the Kalman gain K^{gain}, which specifies the relative proportion of how much more the observational-likelihood should be trusted over the prior belief. The difference is mostly presentation or mathematical convenience, and (3.37) is expressible using relative weights if desired: $m_{t|t} = m_{t|t-1} + K^{\text{gain}}(y_t - m_{t|t-1})$, where $K^{\text{gain}} = W_y$.

Our filtered system inputs the updated belief-mean $m_{t|t}$ into the controller π instead of the observation (seen Fig. 3.1) to decide the control $u_t = \pi(m_{t|t}, \psi)$. $m_{t|t}$ is a smoother signal than y_t over time, injecting less noise into the controller. Also, $m_{t|t}$ is better informed than y_t, being conditioned on, not just the most recent observation, but all previous observations and controls, seen Fig. 3.4, helping to make better control decisions. Giving the controller access to the uncertainty information $V_{t|t}$ too would likely be even more advantageous, but this is left for future work.

The control u_t is not random during execution, rather a specific set of numbers, e.g. voltages applied to actuators. Nevertheless, a joint distribution over the updated (uncertain) belief and the (certain) control is conveniently expressed:

$$\tilde{b}_{t|t} \doteq \begin{bmatrix} b_{t|t} \\ u_t \end{bmatrix} \sim \mathcal{N}\left(\tilde{m}_{t|t} \doteq \begin{bmatrix} m_{t|t} \\ u_t \end{bmatrix}, \tilde{V}_{t|t} \doteq \begin{bmatrix} V_{t|t} & 0 \\ 0 & 0 \end{bmatrix}\right). \tag{3.41}$$

FILTERING PREDICTION STEP

Next, the filtering 'prediction step' computes the predictive-distribution of $b_{t+1|t} \sim p(x_{t+1}|y_{1:t}, u_{1:t})$ from the output of dynamics model f given uncertain input $\tilde{b}_{t|t}$ from (3.41). The distribution of $f(\tilde{b}_{t|t})$ is intractable and non-Gaussian yet has analytically computable moments (Candela et al., 2003). For tractability, $b_{t+1|t}$ is approximated as Gaussian-distributed by matching the moments of Gaussian $b_{t+1|t}$ with the moment of $f(\tilde{b}_{t|t})$:

$$b_{t+1|t} \sim \mathcal{N}\left(m_{t+1|t}, V_{t+1|t}\right). \tag{3.42}$$

To compute the mean output $m_{t+1|t}$ we use the law of iterated expectations, since both the input and function itself are uncertain (given both uncertainties the law of iterated expectations states the output mean equals the mean projection of the uncertain input distribution through the mean of the uncertain function) computed using Appendix B.3.1:

$$\begin{aligned} m^a_{t+1|t} &= \mathbb{E}_{\tilde{b}_{t|t}}\left[f_a(\tilde{b}_{t|t})\right] \\ &= s_a^2 \beta_a^\intercal q_a + \phi_a^\intercal \tilde{m}_{t|t}, && (3.43) \\ \beta_a &\doteq (K_a + \Sigma_a^\varepsilon)^{-1}(y_a - \phi_a^\intercal X), && (3.44) \\ q_a^i &\doteq q(X_i, \tilde{m}_{t|t}, \Lambda_a, \tilde{V}_{t|t}). && (3.45) \end{aligned}$$

The output variance is computed using the law of iterated variances Appendix B.3.3:

$$\begin{aligned} V^{ab}_{t+1|t} &= \mathbb{C}_{\tilde{b}_{t|t}}\left[f_a(\tilde{b}_{t|t}),\ f_b(\tilde{b}_{t|t})\right] \\ &= s_a^2 s_b^2 \big[\beta_a^\intercal (Q_{ab} - q_a q_b^\intercal)\beta_b + \\ &\quad \delta_{ab}\big(s_a^{-2} - \mathrm{tr}((K_a + \Sigma_a^\varepsilon)^{-1} Q_{aa})\big)\big] + C_a^\intercal \tilde{V}_{t|t}\phi_b + \phi_a^\intercal \tilde{V}_{t|t} C_b + \phi_a^\intercal \tilde{V}_{t|t}\phi_b, && (3.46) \\ Q^{ij}_{ab} &\doteq Q\big(X_i, X_j, \Lambda_a, \Lambda_b, 0, \tilde{m}_{t|t}, \tilde{V}_{t|t}\big), && (3.47) \\ C_a &\doteq s_a^2(\Lambda_a + \tilde{V}_{t|t})^{-1}(X - \tilde{m}_{t|t})\beta_a^\intercal q_a, && (3.48) \end{aligned}$$

where a and b refer to the a th and b th dynamics output, q and Q Gaussian functions are defined in Appendix B.1, s_a^2 is the signal variance of the a th GP outputs, Λ_a is a diagonal matrix of squared length scales for GP number a, X the GP inputs, y_a the a'th GP outputs, $\beta_a \doteq (K_a + \Sigma_a^\varepsilon)^{-1}(y_a - \phi_a^\intercal X)$, $K_a \in \mathbb{R}^{\mathrm{N}\times\mathrm{N}}$ is a Gram matrix, and Σ_a^ε is an identity matrix multiplied by the (a,a) element of observation variance matrix Σ_y^ε.

The time series process then repeats using the predictive belief (3.42) as the prior belief in the following timestep. This completes the specification of a filtered system in execution.

3.3.2 Learning Dynamics from Noisy Observations

The original PILCO algorithm ignored sensor noise when training each GP by assuming each observation y_t to be the latent state x_t. However, this approximation breaks down under significant noise. More complex training schemes are required to correctly treat each training datum x_t as latent, yet noisily-observed as y_t. We resort to GP state space model methods, specifically the Direct method (McHutchon, 2014, section 3.5). Given a single episode of data, the Direct method infers the marginal likelihood $p(y_{1:T})$ approximately using moment-matching and a single forward-pass. Doing so, it specifically exploits the time series structure (see Fig. 3.1) that generated observations $y_{1:T}$. We use the direct method to set the GP's training data and observation noise variance Σ_y^ε to the inducing point parameters and noise parameters that optimise the marginal likelihood. In this chapter, we use the superior Direct method to train GPs, both in this extended version of PILCO and in our implementation of the original PILCO algorithm for fair comparison in the experiments.

3.3.3 System Simulation Phase

In *system simulation*, we probabilistically predict the filtered system's behaviour as an analytic distribution over beliefs. A distribution over beliefs b is in principle a distribution over its parameters m and V. To distinguish m, V and b as now being *random* and *hierarchically-random* respectively, we capitalise them: M, V' and B. As an approximation we do not consider the full distribution of V', instead only its mean value $\bar{V} \doteq \mathbb{E}[V']$ (a matrix, fixed for a given timestep, derived later). We assume M is Gaussian distributed. Whilst V' could be Wishart distributed, we leave this to future work, and only use the mean of V'. Their relationship is shown in graphical model Fig. 3.5.

.

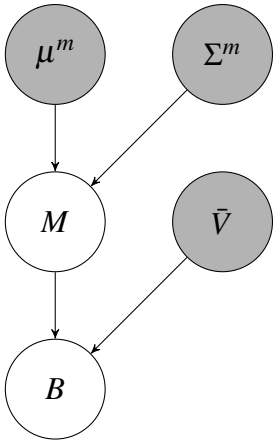


Fig. 3.5 ***Hierarchical Gaussian distribution we use****. The belief function B is Gaussian distribution with mean parameter M and variance $\bar{V}$. The mean parameter M itself is also Gaussian distributed with mean μ^m and vairance Σ^m.*

FILTERING UPDATE STEP

In the simulation phase there is no real world to generate an observation x_t nor do we have explicit modelling of latent system X_t like we did in the original PILCO model (Fig. 3.1). Instead, as per the belief-MDP interpretation of POMDPs (Kaelbling et al., 1998), we anticipate the plausible set of future observations from our beliefs and known observation noise. From the robot's point of view, the probability distribution $p(X)$ is inextricably tied to its beliefs $p(B)$ of the system state. So we have:

$$Y_t = B_{t|t-1} + \varepsilon_t^y \sim \mathcal{N}\left(\mu_t^y, \Sigma_t^y\right), \tag{3.49}$$

$$\mu_t^y = \mu_{t|t-1}^m, \tag{3.50}$$

$$\Sigma_t^y = \Sigma_{t|t-1}^m + \bar{V}_{t|t-1} + \Sigma_y^\varepsilon, \tag{3.51}$$

where (3.51) is computed by marginalising out $M_{t|t-1}$ or 'flattening' the hierarchical structure of $B_{t|t-1}$. Restricting M to being Gaussian-distributed, we have the joint:

$$\begin{bmatrix} M_{t|t-1} \\ Y_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_{t|t-1}^m \\ \mu_{t|t-1}^m \end{bmatrix}, \begin{bmatrix} \Sigma_{t|t-1}^m & \Sigma_{t|t-1}^m \\ \Sigma_{t|t-1}^m & \Sigma_t^y \end{bmatrix}\right). \tag{3.52}$$

Now we apply the filtering update rules for the belief parameters (3.37)–(3.38) except now the belief-mean parameters are now uncertain: $M_{t|t} = W_m M_{t|t-1} + W_y Y_t$. Taking

this into account, the hierarchically-uncertain updated belief posterior is:

$$B_{t|t} \sim \mathcal{N}\left(M_{t|t}, \bar{V}_{t|t}\right), \quad \text{where} \quad M_{t|t} \sim \mathcal{N}\left(\mu^m_{t|t}, \Sigma^m_{t|t}\right), \tag{3.53}$$

$$\mu^m_{t|t} = W_m \mu^m_{t|t-1} + W_y \mu^m_{t|t-1} = \mu^m_{t|t-1}, \tag{3.54}$$

$$\Sigma^m_{t|t} = W_m \Sigma^m_{t|t-1} W_m^\intercal + W_m \Sigma^m_{t|t-1} W_y^\intercal + W_y \Sigma^m_{t|t-1} W_m^\intercal + W_y \Sigma^y_t W_y^\intercal, \tag{3.55}$$

$$\bar{V}_{t|t} = W_m \bar{V}_{t|t-1}. \tag{3.56}$$

Both M and $\bar{V}$ should be initialised according to initial state mean and variance (if known), whereas Σ^m should be initialised as a zero matrix.

Control Decision

Because the controller's input is randomised, $M_{t|t}$, so too is the control output (even though our controller π is a deterministic function):

$$U_t = \pi(M_{t|t}, \psi), \tag{3.57}$$

which is implemented by overloading the controller function:

$$(\mu^u_t, \Sigma^u_t, C_{m_{t|t}u}) = \pi(\mu^m_{t|t}, \Sigma^m_{t|t}, \psi), \tag{3.58}$$

where μ^u_t is the output mean, Σ^u_t the output variance, and $C_{m_{t|t}u}$ is the input-output covariance with premultiplied inverse input variance $C_{m_{t|t}u} \doteq (\Sigma^m_{t|t})^{-1} \mathbb{C}_{M_{t|t}}\left[M_{t|t},\, U_t\right]$. Making a joint Gaussian approximation using moment matching, we have

$$\tilde{M}_{t|t} \doteq \begin{bmatrix} M_{t|t} \\ U_t \end{bmatrix} \sim \mathcal{N}\left(\mu^{\tilde{m}}_{t|t} \doteq \begin{bmatrix} \mu^m_{t|t} \\ \mu^u_t \end{bmatrix},\ \Sigma^{\tilde{m}}_{t|t} \doteq \begin{bmatrix} \Sigma^m_{t|t} & \Sigma^m_{t|t} C_{m_{t|t}u} \\ C^\intercal_{m_{t|t}u} \Sigma^m_{t|t} & \Sigma^u_t \end{bmatrix}\right). \tag{3.59}$$

Filtering Prediction Step

Finally we discuss the prediction step:

$$B_{t+1|t} \sim \mathcal{N}\left(M_{t+1|t}, \bar{V}_{t+1|t}\right), \quad \text{where} \quad M_{t+1|t} \sim \mathcal{N}\left(\mu^m_{t+1|t}, \Sigma^m_{t+1|t}\right), \tag{3.60}$$

where the belief mean-distribution and expected belief variance are computed using Appendix B.4. Starting with the mean of the belief-mean prediction (Appendix B.4.1):

$$\begin{aligned}
\mu^{m,a}_{t+1|t} &= \mathbb{E}_{\tilde{M}_{t|t}}\left[M^a_{t+1|t}\right] \\
&= s_a^2 \beta_a^\intercal \hat{q}_a + \phi_a^\intercal \mu^{\tilde{m}}_{t|t}, && (3.61)\\
\hat{q}^i_a &\doteq q\Big(X_i, \mu^{\tilde{m}}_{t|t}, \Lambda_a, \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t}\Big). && (3.62)
\end{aligned}$$

The variance of the belief-mean (Appendix B.4.3) is:

$$\begin{aligned}
\Sigma^{m,ab}_{t+1|t} &= \mathbb{C}_{\tilde{M}_{t|t}}\left[M^a_{t+1|t},\, M^b_{t+1|t}\right] \\
&= s_a^2 s_b^2 \beta_a^\intercal (\hat{Q}_{ab} - \hat{q}_a \hat{q}_b^\intercal)\beta_b + C^{a\intercal}_{\tilde{m}m'} \Sigma^{\tilde{m}}_{t|t} \phi_b + \phi_a^\intercal \Sigma^{\tilde{m}}_{t|t} C^b_{\tilde{m}m'} + \phi_a^\intercal \Sigma^{\tilde{m}}_{t|t} \phi_b, && (3.63)\\
\hat{Q}^{ij}_{ab} &\doteq Q(X_i, X_j, \Lambda_a, \Lambda_b, \tilde{V}_{t|t}, \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t}), && (3.64)\\
C^a_{\tilde{m}m'} &\doteq s_a^2 (\Lambda_a + \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t})^{-1} (X - \mu^{\tilde{m}}_{t|t}) \beta_a^\intercal \hat{q}_a. && (3.65)
\end{aligned}$$

The mean of the belief-variance (Appendix B.4.5) is:

$$\begin{aligned}
\bar{V}^{ab}_{t+1|t} &= \mathbb{E}_{\tilde{M}_{t|t}}\left[V^{ab}_{t+1|t}\right] \\
&= s_a^2 s_b^2 \left[\beta_a^\intercal (\tilde{Q}_{ab} - \hat{Q}_{ab})\beta_b + \delta_{ab}\left(s_a^{-2} - \mathrm{tr}((K_a + \Sigma^\varepsilon_a)^{-1} \tilde{Q}_{aa})\right)\right] \\
&\quad + C^{a\intercal}_{\tilde{m}m'} \tilde{V}_{t|t} \phi_b + \phi_a^\intercal \tilde{V}_{t|t} C^b_{\tilde{m}m'} + \phi_a^\intercal \tilde{V}_{t|t} \phi_b, && (3.66)\\
\tilde{Q}^{ij}_{ab} &\doteq Q(X_i, X_j, \Lambda_a, \Lambda_b, 0, \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t}). && (3.67)
\end{aligned}$$

We have now discussed the one-step prediction of the filtered system, from $B_{t|t-1}$ to $B_{t+1|t}$. Using this process repeatedly, we can simulate from an initial belief $B_{0|0}$ to $B_{1|1}$, then to $B_{2|2}$ etc., up to $B_{T|T}$.

3.3.4 Controller Evaluation and Improvement

To evaluate a controller we again apply the loss function J (Algorithm 1, line 6) to the multi-step prediction (§ 3.3.3). The controller is again optimised using the analytic gradients of loss J. Since J now is a function of beliefs, we additionally consider the gradients of $B_{t|t-1}$ w.r.t. ψ. As the belief is distributed by $B_{t|t-1} \sim \mathcal{N}\left(M_{t|t-1}, \bar{V}_{t|t-1}\right)$, where $M_{t|t-1} \sim \mathcal{N}\left(\mu^m_{t|t-1}, \Sigma^m_{t|t-1}\right)$, we use partial derivatives of $\mu^m_{t|t-1}$, $\Sigma^m_{t|t-1}$ and $\bar{V}_{t|t-1}$ w.r.t. each other and ψ.

Let $\mathrm{vec}(\cdot)$ be a 'vectorise operator' that reshapes matrices columnwise into vectors. We define $S_t = [M^\intercal_{t|t-1},\, \mathrm{vec}(\bar{V}_{t|t-1})^\intercal]^\intercal$ as the Markov filtered-system from the

belief's parameters. To predict the system evolution, the state distribution is defined:

$$p(S_t) \;=\; \mathcal{N}\left(\mu_t^s = \begin{bmatrix} \mu_{t|t-1}^m \\ \mathrm{vec}(\bar{V}_{t+1|t}) \end{bmatrix},\; \Sigma_t^s = \begin{bmatrix} \Sigma_{t|t-1}^m & 0 \\ 0 & 0 \end{bmatrix}\right), \tag{3.68}$$

and the expected cost at each timestep is $\bar{c}_t = \mathbb{E}_{X_t}[\mathrm{cost}(X_t)]$, where $X_t \sim \mathcal{N}\left(\mu_{t|t-1}^m, \Sigma_{t|t-1}^m + \bar{V}_{t|t-1}\right)$. Now to compute the loss gradient $\mathrm{d}J/\mathrm{d}\psi$ where $J = \sum_{t=0}^{T} \bar{c}_t$, we require $\mathrm{d}\bar{c}_t/\mathrm{d}\psi$ at each timestep:

$$\begin{aligned} \frac{\mathrm{d}\bar{c}_t}{\mathrm{d}\psi} &= \frac{\mathrm{d}\bar{c}_t}{\mathrm{d}p(S_t)}\frac{\mathrm{d}p(S_t)}{\mathrm{d}\psi} \\ &= \frac{\partial \bar{c}_t}{\partial \mu_t^s}\frac{\mathrm{d}\mu_t^s}{\mathrm{d}\psi} + \frac{\partial \bar{c}_t}{\partial \Sigma_t^s}\frac{\mathrm{d}\Sigma_t^s}{\mathrm{d}\psi}, \qquad \text{and} \end{aligned} \tag{3.69}$$

$$\frac{\mathrm{d}p(S_{t+1})}{\mathrm{d}\psi} = \frac{\partial p(S_{t+1})}{\partial p(S_t)}\frac{\mathrm{d}p(S_t)}{\mathrm{d}\psi} + \frac{\partial p(S_{t+1})}{\partial \psi}. \tag{3.70}$$

Application of the chain rule backwards from the state distribution at the horizon S_T, to S_0 is analogous to that detailed in PILCO (Deisenroth and Rasmussen, 2011), where we use S_t, μ_t^s and Σ_t^s in the place of x_t, μ_t and Σ_t respectively.

3.3.5 Experimental Setup

We test our algorithm on the cartpole swing-up problem (Fig. 3.6, parameters values Table D.1), a benchmark for comparing controllers of nonlinear dynamical systems. Although the cart-pole setup is a control-affine system, the system is nevertheless underactuated, has control-constraints and dynamics uncertainty, thus precluding a feedback linearisation control solution. We experiment using a physics-simulator of the cart-pole which solves the differential equations (unknown to the robot) of the system. Note the physics-simulator is used to generate the latent ground truth of the system during experiments, our robot has no direct access the physics-simulator itself, nor its ODEs, nor its outputs.

Our task begins with a downwards hanging pendulum and a goal of swinging up and stabilising the pendulum. The cart mass is $m_c = 0.5$kg. A zero-order hold controller applies horizontal forces to the cart within control constants of $[-10, 10]$N. The controller function is a mixture of 100 RBF centroids, chosen from previous experience of the required expressibility of the controller's function. The centroids are initialised by randomly sampling a Gaussian distribution whose mean is the initial state, and an identity variance. Friction resists the cart's motion with damping

coefficient $b = 0.1\text{Ns/m}$. Connected to the cart is a pole of length $l = 0.2\text{m}$ and mass $m_p = 0.5\text{kg}$ located at its endpoint, which swings due to gravity's acceleration $g = 9.82\text{m/s}^2$. An inexpensive camera observes the system. Frame rates of \$10 webcams are typically 30Hz at maximum resolution, thus the time discretisation is $\Delta t = 1/30s$. The state x comprises the cart position, pendulum angle, and their time derivatives $x = [x_c, \theta, \dot{x}_c, \dot{\theta}]^\intercal$. The cartpole's motion is described with the differential equation:

$$\dot{x}^\intercal = \begin{bmatrix} \dot{x_c} \\ \dot{\theta} \\ \frac{-2m_p l\dot{\theta}^2 s + 3m_p gsc + 4u - 4b\dot{x_c}}{4(m_c+m_p) - 3m_p c^2} \\ \frac{-3m_p l\dot{\theta}^2 sc + 6(m_c+m_p)gs + 6(u-b\dot{x_c})c}{4l(m_c+m_p) - 3m_p lc^2} \end{bmatrix}, \tag{3.71}$$

using shorthand $s = \sin\theta$ and $c = \cos\theta$. We both randomly-initialise the system and set the initial belief of the system according to $B_{0|0} \sim \mathcal{N}\left(M_{0|0}, V_{0|0}\right)$ where $M_{0|0} \sim \delta([0, \pi, 0, 0]^\intercal)$ and $V_{0|0}^{1/2} = \text{diag}([0.2\text{m}, 0.2\text{rad}, 0.2\text{m/s}, 0.2\text{rad/s}])$. Each episode lasts two seconds (a 60 timestep horizon).

The camera's noise standard deviation is: $(\Sigma_y^\varepsilon)^{1/2} = \text{diag}([0.03\text{m}, 0.03\text{rad}, \frac{0.03}{\Delta t}\text{m/s}, \frac{0.03}{\Delta t}\text{rad/s}])$, noting $0.03\text{rad} \approx 1.7°$. Since the camera cannot observe velocities, only estimate velocities using finite differences of positions, their estimated standard deviation errors are related to their associated positional errors divided by Δt. For our experiment we chose $(\Sigma_y^\varepsilon)^{1/2}$ to be large enough that algorithms cannot simply make zero-noise assumptions without deterioration in performance, and to be small enough that control of the system was still possible.

The cost function we impose is $1 - \exp\left(-\frac{1}{2}d^2/\lambda_c^2\right)$ where $\lambda_c = 0.25\text{m}$ and d^2 is the squared Euclidean distance between the pendulum's end point (x_p, y_p) and its goal $(0, l)$. The squared distance $d^2 = x_p^2 + (l - y_p)^2 = (x_c - l\sin\theta)^2 + (l - l\cos\theta)^2$.

We compare four algorithms:

1. PILCO by Deisenroth and Rasmussen (2011) as a baseline (unfiltered execution, and unfiltered full-prediction);
2. Dallaire et al. (2009) (filtered execution, and filtered MAP-prediction);
3. Deisenroth and Peters (2012) (filtered execution, and unfiltered full-prediction); and lastly
4. our method (filtered execution, and filtered full-prediction).

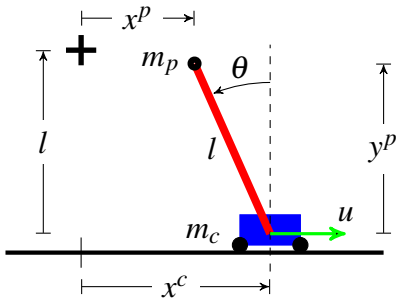


Fig. 3.6 ***The cartpole swing-up task.*** *A pendulum of length l is attached to a cart by a frictionless pivot. The cart has mass m_c and position x_c. The pendulum's endpoint has mass m_p and position (x_p, y_p), with angle θ from vertical. The cart begins at position $x_c = 0$ and pendulum hanging down: $\theta = \pi$. The goal is to accelerate the cart by applying horizontal force u_t at each timestep t to invert then stabilise the pendulum's endpoint at the goal (black cross), i.e. to maintain $x_c = 0$ and $\theta = 0$.*

For clear comparison we first opted for a tightly controlled experiment. We control for data and dynamics models: each algorithm has access to the exact same data and exact same dynamics model. The reason is to eliminate variance in performance caused by different algorithms choosing different control decisions. We generate a single dataset by running the baseline PILCO algorithm for $E = 10$ episodes (totalling 22 seconds of system interaction). Each method is an off-policy RL algorithm, meaning there is no direct importance concerning *how* the data was generated, only *what* the data is. The independent variables of our experiment are 1) the method of system prediction and 2) the method of system execution. We then optimise each controller using their respective prediction methods. Finally, we measure and compare their performances in both prediction and execution.

3.3.6 Results with a Common Dataset

We compare algorithm performance, both predictive (from simulation using the dynamics model) seen Fig. 3.7, and empirical (from execution on the physics-simulator) seen Fig. 3.8.

Predictive Performance

First, we analyse predictive costs per timestep (Fig. 3.7). Since predictions are probabilistic, the costs have distributions, with the exception of Dallaire et al. (2009) which predicts MAP trajectories and therefore has deterministic cost. Even though we plot distributed costs, controllers are optimised w.r.t. expected total cost only. Using the same dynamics, the different prediction methods optimise different controllers

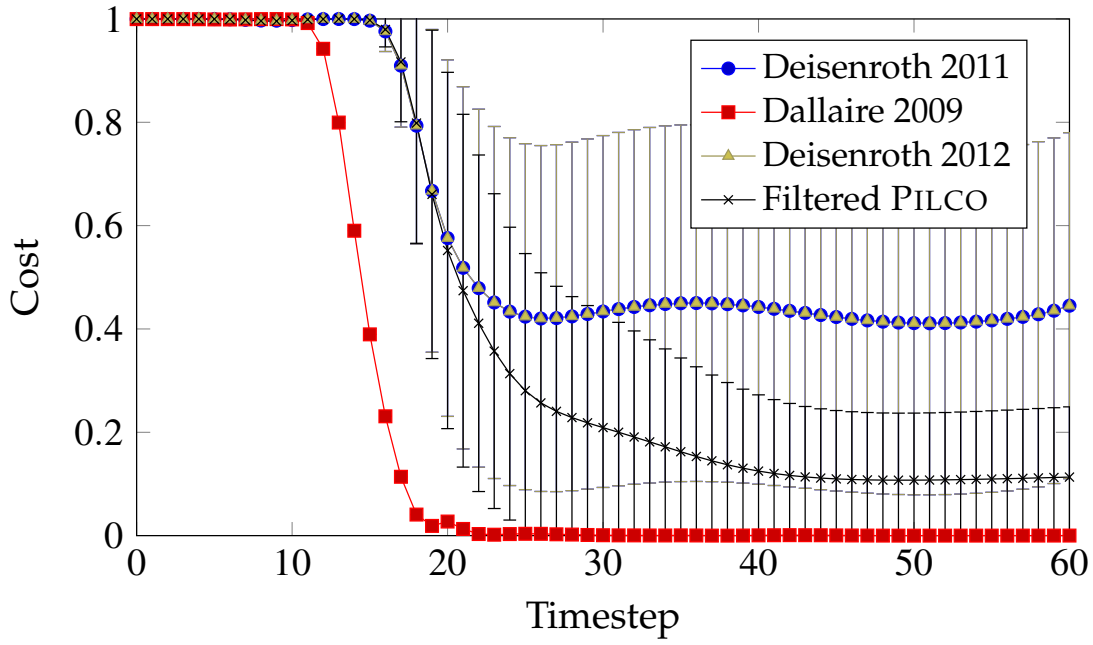


Fig. 3.7 ***Predictive costs per timestep.*** *The error bars show ±1 standard deviation. Each algorithm has access to the same data set (generated by Deisenroth 2011) and dynamics model. Algorithms differ in their simulation methods (except Deisenroth's algorithms whose predictions overlap).*

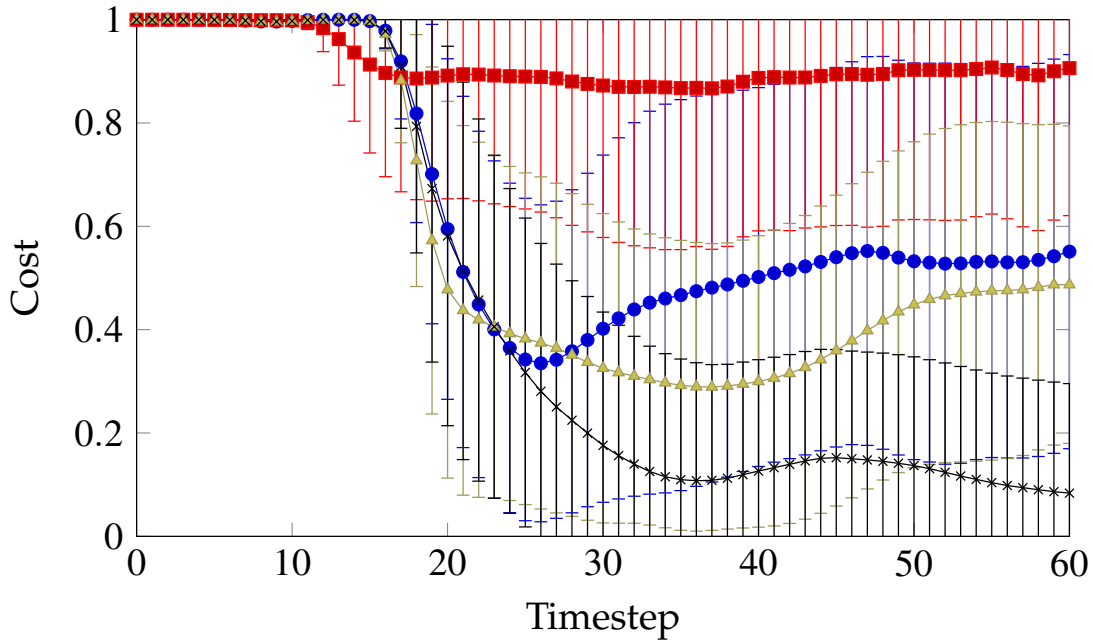


Fig. 3.8 ***Empirical costs per timestep.*** *We generate empirical cost distributions from 100 executions per algorithm. Error bars show ±1 standard deviation. The plot colours and shapes correspond to the legend in Fig. 3.7.*

(with the exception of Deisenroth and Rasmussen (2011) and Deisenroth and Peters (2012), whose prediction methods are identical). During the first 10 timesteps, we note identical performance with maximum cost due to the non-zero time required physically swing the pendulum up near the goal. Performances thereafter diverge. Since we predict w.r.t. a filtering process, less noise is predicted to be injected into the controller, and the optimiser can thus afford higher gain parameters w.r.t. the pole at balance point. If we linearise our controller around the goal point (black cross, Fig. 3.6), our controller has a gain of -81.7N/rad w.r.t. pendulum angle, a larger-magnitude than both Deisenroth method gains of -39.1N/rad (negative values refer to *left* forces in Fig. 3.6). Being afforded higher gains our controller is more reactive and more likely to catch a falling pendulum. Finally, we note Dallaire et al. (2009) predict very high performance. Without balancing the costs across multiple possible trajectories, the method instead optimises a sequence of deterministic states to near perfection.

Empirical Performance

To compare the predictive results against the empirical, we execute each algorithm 100 times (Fig. 3.8). Note, we do not use 100 different generated datasets, but evaluate using 100 executions from 1 generated dataset. First, we notice a stark difference between predictive and executed performances from Dallaire et al. (2009), due to neglecting model uncertainty, suffering from model bias. In contrast, the other methods consider uncertainty and have relatively unbiased predictions, judging by the similarity between predictive-vs-empirical performances. Deisenroth's methods, which differ only in execution, illustrate that filtering during execution-only can be better than no filtering at all. However, the major benefit comes when the controller is evaluated from multi-step predictions of a filtered system. Opposed to Deisenroth and Peters (2012), our method's simulation reflects reality closer because we both simulate and execute system trajectories using closed loop filtering control.

To test statistical significance of empirical cost differences given 100 executions, we use a Wilcoxon rank-sum test at each timestep (Wilcoxon et al., 1970). The Wilcoxon test uses paired samples to see if samples are drawn from populations with different means. Excluding timesteps ranging $t = [0, 29]$ (whose costs are similar), the minimum Wilcoxon z-score over timesteps $t = [30, 60]$ that our method has superior average-cost than each other methods follows: Deisenroth 2011 $\min(z) = 4.99$, Dallaire 2009's $\min(z) = 8.08$, Deisenroth 2012's $\min(z) = 3.51$. Since the minimum

$\min(z) = 3.51$, we have $p > 99.9\%$ certainty our method's average empirical cost is superior than each other method over timesteps $t = [30, 60]$.

3.3.7 Results of Full Reinforcement Learning Task

In the previous experiment (§ 3.3.6) we used a common dataset to compare each algorithm, to isolate and focus on how well each algorithm makes *use* of data, rather than also considering the different ways each algorithm *collects* different data. Here, we remove the constraint of a common dataset, and test the full reinforcement learning task by letting each algorithm collect its own data over repeated trails of the cart-pole task. Each algorithm is allowed 15 trails (episodes), repeated 10 times with different random seeds. For a particular re-run experiment and episode number, an algorithm's predicted loss is unchanged when repeatedly computed, yet the empirical loss differs due to random initial states, observation noise, and process noise. We therefore average the empirical results over 100 random executions of the controller at each episode and seed.

The predictive loss (cumulative cost) distributions of each algorithm are shown Fig. 3.9. Perhaps the most striking difference between the full reinforcement learning predictions and those made with a controlled dataset (Fig. 3.7) is that Dallaire does not predict it will perform well. The quality of the data collected by Dallaire within the first 15 episodes is not sufficient to predict good performance. Our method, 'Filter PILCO', predicts it will outperform the competing algorithms, which we verify against empirical results.

Empirical results of each algorithm are shown Fig. 3.10. Of interest is how each algorithm each perform equally poorly in the first 4 episodes, but a big gain is made by Filtered PILCO by the end of the 7th trial. Such a learning rate was similar to the original PILCO on the cart-pole problem. We can see that Dallaire again overestimated its performance and performed more poorly than expected. As before, both Deisenroth methods predict their empirical performance accurately. Filtered PILCO method performs approximated as it predicted it would and outperformed the competing algorithms. Since each algorithm's performance plateaus after 10 episodes, we are reasonable confident each algorithm (except possibly Dallaire) is now limited by its simulator's accuracy, opposed to data.

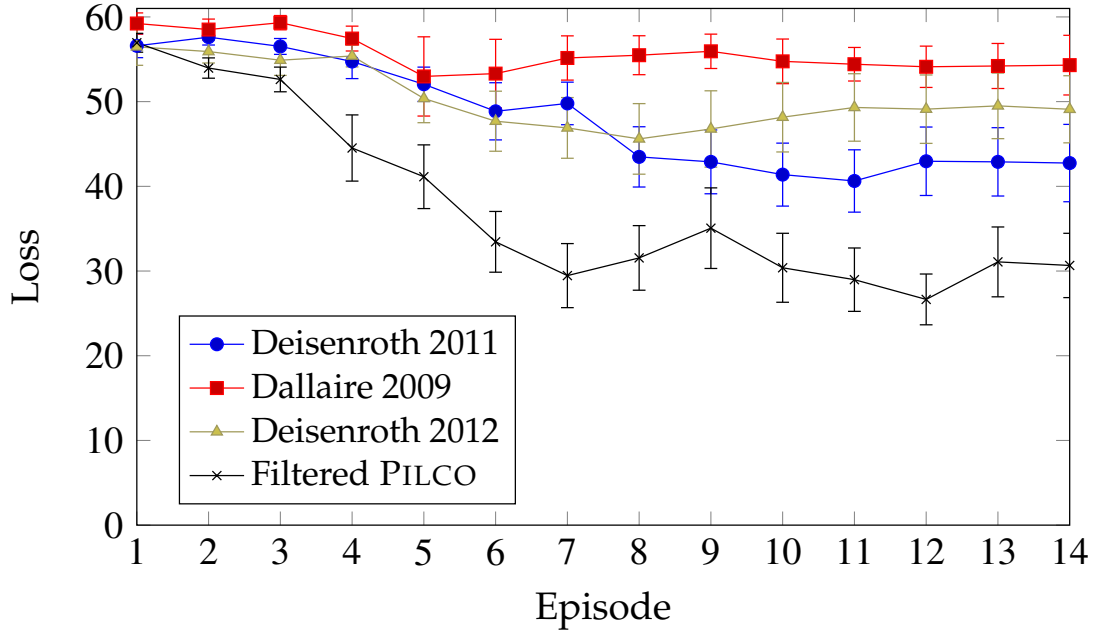


Fig. 3.9 ***Predictive loss per episode.*** *Error bars show ±1 standard error of the mean predicted loss given 10 repeats of each algorithm.*

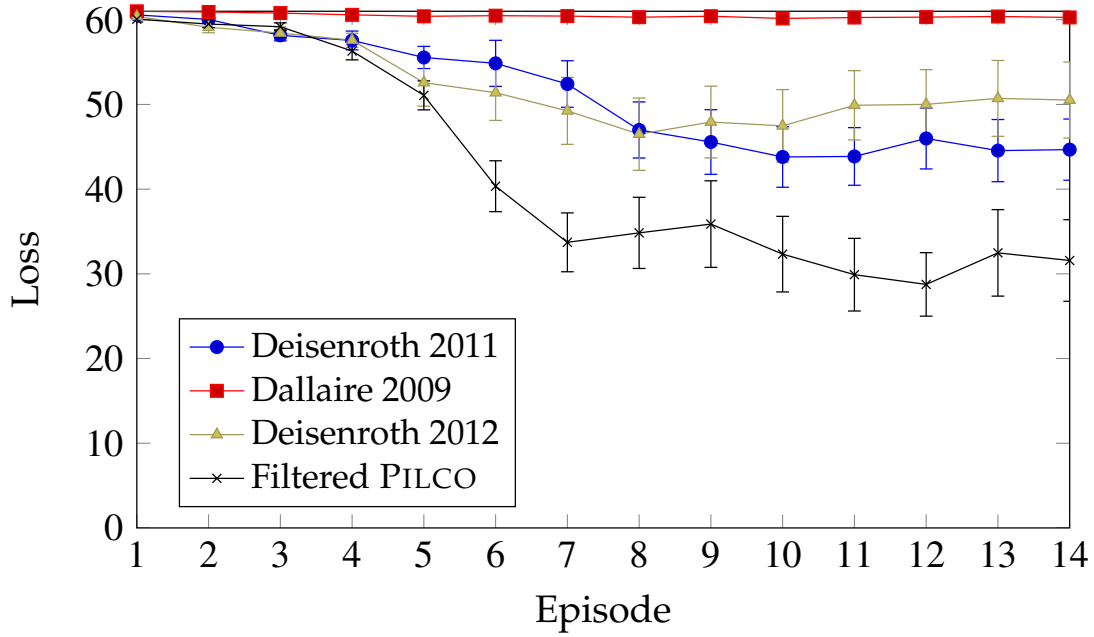


Fig. 3.10 ***Empirical loss per episode.*** *Error bars show ±1 standard error of the mean empirical loss given 10 repeats of each algorithm. In each repeat we computed the mean empirical loss using 100 independent executions of the controller.*

3.3.8 Results with Various Observation Noises

We also experimented with different observation noise levels, comparing PILCO (Fig. 3.11) with Filtered PILCO (Fig. 3.12). Both figures show a noise factors n, such that the observation noise is: $(\Sigma_y^\epsilon)^{1/2} = n \times \text{diag}([0.01\text{m}, 0.01\text{rad}, \frac{0.01}{\Delta t}\text{m/s}, \frac{0.01}{\Delta t}\text{rad/s}])$. For reference, our previous experiments (§ 3.3.6 and § 3.3.7) used a noise factor of $n = 3$. Notice at low noise factor $n = 1$, both algorithms perform similarly-well, since any zero-noise assumptions are approximately correct, and the observations are precise enough to control a system without a filter. However, as observations noise increases, the performance of unfiltered PILCO soon drops. Eventually the Filtered PILCO system breaks down too (Fig. 3.12), although not until $n = 16$.

3.3.9 Further Analysis

Computational Complexity

We demonstrated how our algorithm outperforms others, but we should ask at what computational cost. Training the GP dynamics model involved $\text{N} = 660$ data, $\text{M} = 50$ inducing points under a sparse GP FITC, $\text{R} = 100$ controller function RBF centroids, $\text{X} = 4$ state dimensions, $\text{U} = 1$ control dimensions, and $T = 60$ timestep horizon. Dynamics model training scales $\mathcal{O}(\text{XNM}^2)$. Whilst sparse methods are perhaps not entirely necessary when the total data is under 1000 points, we found typically $50 - 100$ points are usually sufficient, and sparse methods become more necessary for more complex robots (such as the cart double-pole used next chapter). Policy optimisation (with 300 steps, each of which require trajectory prediction with gradients) is the most intense part: our method and both Deisenroth's methods scale $\mathcal{O}(\text{M}^2\text{X}^2(\text{X}+\text{U})^2T + \text{R}^2\text{X}^2\text{U}^2T)$, whilst Dallaire's only scales $\mathcal{O}(\text{MX}(\text{X}+\text{U})T + \text{RXU}T)$. Worst case we require $\text{M} = \mathcal{O}(\exp(\text{X}+\text{U}))$ inducing points to totally cover the state space to capture the dynamics, the average case is unknown. Total training time was four hours to train the original PILCO method with an additional one hour to re-optimise the controller. Thus, our method's do not scale as well as Dallaire, but under significant observation noise, where Dallaire's method empirically fails, we may have no other choice. Both Deisenroth methods are more robust to observation noise than Dallaire, but considering Deisenroth's methods scale similarly with our method, we recommend our method under significant observation noise.

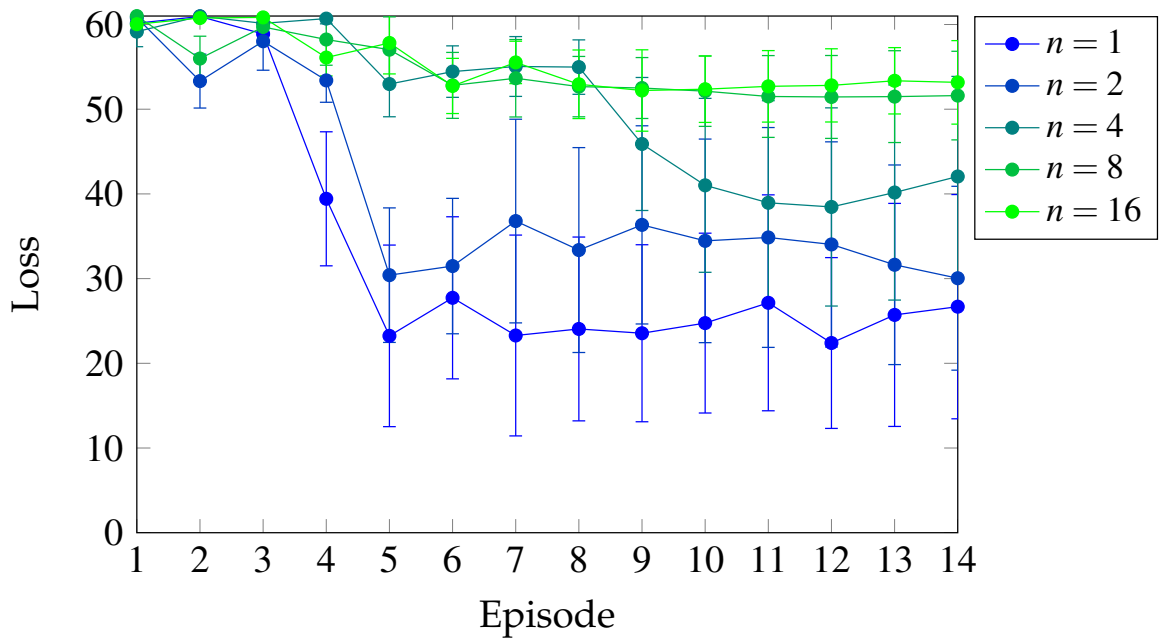


Fig. 3.11 ***Empirical loss of Deisenroth 2011 for various noise levels.*** *The error bars show* ± 1 *empirical loss distribution given 100 executions per noise level.*

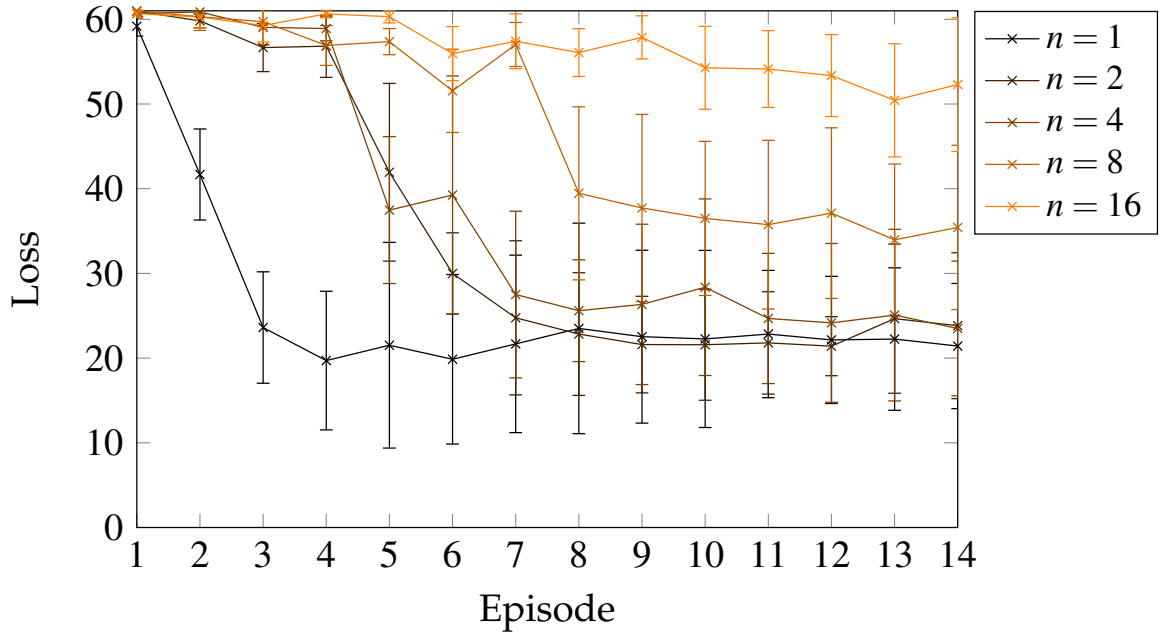


Fig. 3.12 ***Empirical loss of Filtered*** PILCO ***for various noise levels.*** *The error bars show* ± 1 *empirical loss distribution given 100 executions per noise level.*

STABILITY ANALYSIS CONTINUED

Earlier in § 3.1.2 we analysed stability of unfiltered system by analysing the growth of the state-variance over time. Here, we continue that analysis, instead now in the context of filtered observations to better understand the effects of filtering. Consider a filtered system where $B_{t|t-1} \sim \mathcal{N}\left(M_{t|t-1}, \bar{V}_{t|t-1}\right)$, where $M_{t|t-1} \sim \mathcal{N}\left(0, \Sigma^m_{t|t-1}\right)$, and using $\Sigma^m_{t|t-1} + \bar{V}_{t|t-1} = \Sigma^x_t$ we get $\mathbb{V}[Y_t] = \Sigma^x_t + \Sigma^\varepsilon_y$ similar to the unfiltered analysis § 3.1.2. Then according to (3.52), the joint distribution of $M_{t|t-1}$ and Y_t is

$$\begin{bmatrix} M_{t|t-1} \\ Y_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \Sigma^m_{t|t-1} & \Sigma^m_{t|t-1} \\ \Sigma^m_{t|t-1} & \Sigma^m_{t|t-1} + \bar{V}_{t|t-1} + \Sigma^\varepsilon_y \end{bmatrix} \right). \tag{3.72}$$

Because $m_{t|t} = W_m m_t + W_y y_t$, we have

$$\begin{aligned} \mathbb{V}\left[M_{t|t}\right] &= [W_m, W_y] \begin{bmatrix} \Sigma^m_{t|t-1} & \Sigma^m_{t|t-1} \\ \Sigma^m_{t|t-1} & \Sigma^m_{t|t-1} + \bar{V}_{t|t-1} + \Sigma^\varepsilon_y \end{bmatrix} [W_m, W_y]^\intercal \\ &= \Sigma^m_{t|t-1} + W_y(\bar{V}_{t|t-1} + \Sigma^\varepsilon_y) W_y^\intercal \\ &= \Sigma^x_t - \bar{V}_{t|t}, \qquad (3.73) \\ \bar{V}_{t|t} &= \bar{V}_{t|t-1}(\bar{V}_{t|t-1} + \Sigma^\varepsilon_y)^{-1} \Sigma^\varepsilon_y \\ &= W_y \Sigma^\varepsilon_y, \qquad (3.74) \end{aligned}$$

using the fact that $W_m + W_y = I$, $\Sigma^m_{t|t-1} + \bar{V}_{t|t-1} = \Sigma^x_t$, and $W_y = \bar{V}_{t|t-1}(\bar{V}_{t|t-1} + \Sigma^\varepsilon_y)^{-1}$ from (3.40). Note the total variance after the update step $\mathbb{V}\left[M_{t|t}\right] + \bar{V}_{t|t} = \Sigma^x_t$, i.e. equal to the total state variance $\mathbb{V}[X_t]$ before the update. Now we move onto computing $\mathbb{V}\left[\tilde{M}_{t|t}\right]$. By linearising the controller function, as was done § 3.1.2, so $\pi(m_{t|t}) \approx \Pi m_{t|t}$ locally (note the Jacobian matrix B here is not to be confused with the belief $B_{t|t-1}$), we get

$$\mathbb{V}\left[\tilde{M}_{t|t}\right] \doteq \mathbb{V}\begin{bmatrix} M_{t|t} \\ U_t \end{bmatrix} = \begin{bmatrix} I \\ \Pi \end{bmatrix} \mathbb{V}\left[M_{t|t}\right] \begin{bmatrix} I \\ \Pi \end{bmatrix}^\intercal. \tag{3.75}$$

Similarly linearising the dynamics such that $f(m_{t|t}, u_t) \approx A m_{t|t} + B u_t + \varepsilon^x_t$ locally, we get

$$\begin{aligned} \mathbb{V}\left[M_{t+1|t}\right] &= [A, B] \mathbb{V}\left[\tilde{M}_{t|t}\right] [A, B]^\intercal + \Sigma^\varepsilon_x \\ &= (A + B\Pi)(\Sigma^x_t - \bar{V}_{t|t})(A + B\Pi)^\intercal + \Sigma^\varepsilon_x, \qquad (3.76) \\ \bar{V}_{t+1|t} &= A \bar{V}_{t|t} A^\intercal. \qquad (3.77) \end{aligned}$$

Now by summing the variance of the belief-mean $\mathbb{V}\left[M_{t+1|t}\right]$ with the mean of the belief-variance $\bar{V}_{t+1|t}$ using the total law of variance, we can recover the total variance on the state after the prediction step:

$$\begin{aligned}\mathbb{V}\left[X_{t+1}^{\text{filtered}}\right] &= \mathbb{V}\left[M_{t+1|t}\right] + \bar{V}_{t+1|t} \\ &= \underbrace{(A+B\Pi)(\Sigma_t^x - \bar{V}_{t|t})(A+B\Pi)^\intercal}_{\text{down-weighted system response}} + \underbrace{AW_y\Sigma_y^\varepsilon A^\intercal}_{\text{amplified obs. noise}} + \underbrace{\Sigma_x^\varepsilon}_{\text{process noise}} \end{aligned} \tag{3.78}$$

The above shows the total state variance after one timestep of a filtered system. An unfiltered system's state variance (copied for reader's convenience from (3.4) on page 47) for comparison was:

$$\mathbb{V}\left[X_{t+1}^{\text{unfiltered}}\right] = \underbrace{(A+B\Pi)\Sigma_t^x(A+B\Pi)^\intercal}_{\text{system response}} + \underbrace{(B\Pi)\Sigma_y^\varepsilon(B\Pi)^\intercal}_{\text{amplified obs. noise}} + \underbrace{\Sigma_x^\varepsilon}_{\text{process noise}}. \tag{3.79}$$

We can now see that filtering reduces the rate of state-variance change over time compared with unfiltered control. To simplify the comparison, we break (3.78) – (3.79) into three terms, and again assume full-actuation (invertible B and Π). First, the base 'system response' variance (independent of observation noise) of unfiltered systems (3.79) has weighting Σ_t^x. Filtered systems instead have reduced weightings: $\Sigma_t^x - \bar{V}_{t|t} \preceq \Sigma_t^x$, seen (3.78). Second, the 'amplified observation noise' in (3.79) is approximately down-weighted by $W_y \preceq I$ in (3.78) (compared to (3.79)), considering that the least-variance solution of Π in (3.78) is $A = -B\Pi$. Thus, the amplified observation noise is (3.78) becomes $(B\Pi)W_y\Sigma_y^\varepsilon(B\Pi)^\intercal$, i.e. down-weighted from (3.79). Third is the unavoidable process noise, unchanged.

Another interesting difference is the controller gain that minimises the state-variance. From § 3.1.2, the optimal gain from unfiltered controllers was $\Pi^*_{\text{unfiltered}} = -B^{-1}A\Sigma_t^x(\Sigma_t^x + \Sigma_y^\varepsilon)^{-1}$, whilst the filtered is $\Pi^*_{\text{filtered}} = -B^{-1}A$. In terms of variance growth, assuming optimal gain Π^*, from § 3.1.2 an unfiltered state variance grow if $A\Sigma_t^x(\Sigma_t^x + \Sigma_y^\varepsilon)^{-1}\Sigma_y^\varepsilon A^\intercal + \Sigma_x^\varepsilon \not\preceq \Sigma_t^x$, whilst the filtered system has: $A\bar{V}_{t|t}A^\intercal + \Sigma_x^\varepsilon \not\preceq \Sigma_t^x$. We re-express these terms in Table 3.1 for easy comparison.

Table 3.1 ***Comparison of stability between unfiltered and filtered controllers***

Observations	Optimal gain	Unsustainable variance if
Unfiltered	$\Pi^* = -B^{-1}A(I + \Sigma_y^\varepsilon/\Sigma_t^x)^{-1}$	$A((\Sigma_y^\varepsilon)^{-1} + (\Sigma_t^x \quad)^{-1})^{-1}A^\intercal + \Sigma_x^\varepsilon \not\preceq \Sigma_t^x$
Filtered	$\Pi^* = -B^{-1}A$	$A((\Sigma_y^\varepsilon)^{-1} + (\bar{V}_{t\|t-1})^{-1})^{-1}A^\intercal + \Sigma_x^\varepsilon \not\preceq \Sigma_t^x$

From Table 3.1 we first note the optimal gain Π^* of unfiltered systems is a function of the relative amount of observation noise (relative to state variance) $\Sigma_y^\varepsilon / \Sigma_t^x$. Under significant noise, the gain is forced to be small, to avoid injection of noise into a controller. By contrast, the filtered system's gain is independent of the amount of observation noise. The filtering process naturally counters high-variance observations with small weighting W_y before controller input. This effect explains our experimental observations in § 3.3.6, with larger-magnitude pendulum angle gains of -81.7N/rad for filtered control, compared to unfiltered -39.1N/rad.

As for variance growth, both unfiltered and filtered are equivalent except for a single term Σ_t^x and $\bar{V}_{t|t-1}$. Note that $\bar{V}_{t|t-1} \preceq \bar{V}_{t|t-1} + \Sigma_t^m = \Sigma_t^x$, meaning the filtered expression is always less than the unfiltered expression. If the unfiltered expression is satisfied, then the filtered expression can always tolerate more observation noise, at least up to the amount: $\hat{\Sigma}_y^\varepsilon = ((\Sigma_y^\varepsilon)^{-1} + (\Sigma_t^x)^{-1} - \bar{V}_{t|t-1}^{-1})^{-1}$, or a factor $(I - \Sigma_y^\varepsilon(\bar{V}_{t|t-1}^{-1} - (\Sigma_t^x)^{-1}))^{-1}$ greater than the original noise level Σ_y^ε.

3.4 Filtered Control with Latent-Variable Belief-MDPs

So far we have discussed two possible models of a controlled system, unfiltered control (Fig. 3.1 on page 45) and filtered control (Fig. 3.4 on page 59). A key difference between models is the planning space, either planning w.r.t. future observations (unfiltered systems), or future beliefs (filtered systems). An obvious difference between both figures is the lack of latent variables X in Fig. 3.4. Indeed the robot's belief B over latent system X is sufficient to avoid modelling X in the directed graph, as per the belief-MDP interpretation of POMDPs (Kaelbling et al., 1998). Nevertheless, Fig. 3.4 can be confusing due to the absence of the latent system variables X we care about.

Alternatively, one can model both latent states X *and* beliefs B, seen Fig. 3.13. Arguably a more intuitive model of filtered control, Fig. 3.13 'correctly' shows the latent state X generating observation Y, not the robot's belief as in Fig. 3.4. Additionally it is easier to see how the system and the robot's beliefs of the system interact. Fig. 3.13 demonstrates how the physical state X affects the robot's belief via observations, and how the belief can affect the system via control decisions.

Before using model Fig. 3.13, we should investigate two important questions:

1. how does Fig. 3.13 relate to Fig. 3.4?

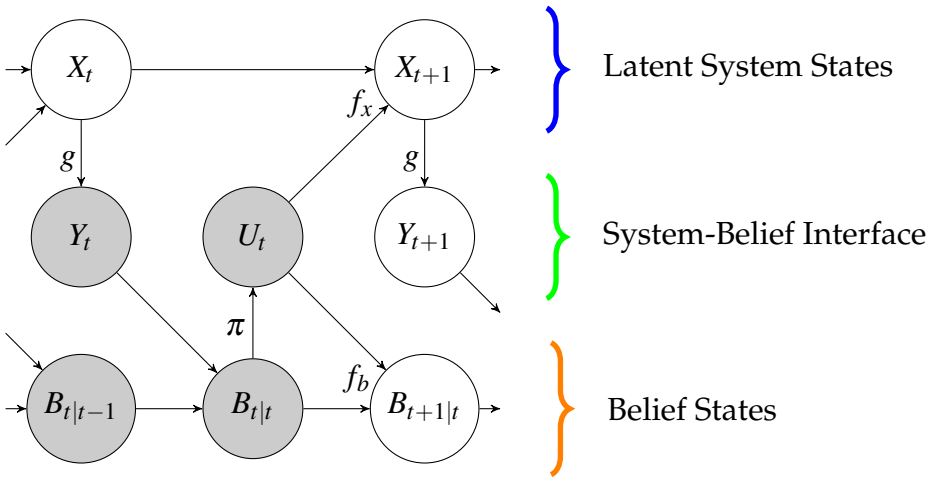


Fig. 3.13 ***Filtered control for latent-variable belief-MDPs****, a more-general extension of PILCO to Bayesian filtering than Fig. 3.4. The latent system (top row) interacts with the robot's belief (bottom row) via a series of observations and control decisions (middle row). At each timestep the latent system X_t is observed noisily as Y_t. The prior belief $B_{t|t-1}$ (whose dual subscript means belief of the latent physical state at time t given all observations up until time $t-1$ inclusive) is combined with observation Y_t resulting in posterior belief $B_{t|t}$ (the update step). Then, the mean posterior belief $\mathbb{E}\left[B_{t|t}\right]$ is inputted into controller function π to decide control U_t. Finally, the next timestep's prior belief $B_{t+1|t}$ is predicted using dynamics model f_b (the predict step).*

2. what additional functionality does Fig. 3.13 provide over Fig. 3.4 to warrant the additional complexity?

Answering the first question: Fig. 3.13 is a generalisation of the Fig. 3.4 framework for filtered control, and thus generalises POMDPs. A proof is provided in Appendix E.6. Answering our second question: the generality of Fig. 3.13 enable the framework to answer two questions that POMDPs cannot:

- What are the consequence of 'incorrect' prior beliefs? For instance, what will happen if the robot's initial prior beliefs $p(B_{t|t-1})$ are not identically distributed with the initial state of the system $p(X_t)$?
- What are the consequences of using a different dynamics model f_b for online execution, and f_x for simulation?

Arguably, considering incorrect prior beliefs is not very interesting, since from the robot's point of view, a belief is all it has. The robot has no direct way to validate its beliefs against the latent ground truth X. Thus we concentrate on the second additional functionality of our framework, different dynamics models.

Consider the situation where system execution must happen online. A dynamics model used for filtering (f_b) during the system execution phase must implemented

for real-time use, i.e. must compute predictions within Δt seconds, before the next observation arrives. Given time constraint Δt, we may be forced to trade off prediction accuracy for speed. For example, GPs predict faster with fewer inducing points, but are less accurate. However, between online system executions, the system simulation phase of Algorithm 1 is often offline. Without a time constraint of Δt on making predictions, we can afford a slow but accurate dynamics model f_x (e.g. a full, not sparse, GP). During simulation, it would be unfaithful to anticipate that our filter has access to such an accurate dynamics model as f_x, since we know the filter only has access to inaccurate f_b. Nevertheless, we can accurately simulate how a system uses an inaccurate-model to filter observations.

To do so, we need to build such a simulator. Note the system execution phase for filtering control with latent variables is identical to that of § 3.3.1. The only difference is using dynamics model f_b instead of a shared model f. The simulation phase is where this generalised framework for filtered control will differs from § 3.3. To simulate Fig. 3.13, with multiple one-step predictions, the belief B is no longer a Markov state state to simulate the system, so we cannot simulate by iteratively predicting from $B_{t|t-1}$ to $B_{t+1|t}$, for $t = [0, T]$, since belief now affect the latent state, and the latent state affects the belief. To simulate our generalised system, a Markov state would be to iteratively simulate the joint Markov state $\{X_t, B_{t|t-1}\}$ forwards to $\{X_{t+1}, B_{t+1|t}\}$ for $t = [0, T]$

3.4.1 System Simulation Phase

In *simulation* we need to compute the behaviour of the filter and system over the *state distribution*. A state distribution is in principle a distribution over the variables x, m and V. To distinguish these variables as now being *random*, we capitalise them: X, M and V'. Ideally, we would randomise the variance using a Wishart distribution. However, as an approximation we are going to assume that the distribution on the variance is just a delta function (a fixed value): $\bar{V} = \mathbb{E}[V']$

Filtering Update Step

Since simulating forwards from one timestep to the next now requires prediction of both X_{t+1} and $B_{t+1|t}$, each of which depends on both X_t and $B_{t|t-1}$. Thus to make predictions, we require a Markov state which summarises all information in X_t and $B_{t|t-1}$. We begin with the assumption that the system state and belief-mean inputted

are jointly Gaussian and denoted as a 'hybrid' state H:

$$H_t \doteq \begin{bmatrix} X_t \\ M_{t|t-1} \end{bmatrix} \sim \mathcal{N}\left(\mu_t^h = \begin{bmatrix} \mu_t^x \\ \mu_{t|t-1}^m \end{bmatrix}, \Sigma_t^h = \begin{bmatrix} \Sigma_t^x & \Sigma_{t|t-1}^{xm} \\ \Sigma_{t|t-1}^{mx} & \Sigma_{t|t-1}^m \end{bmatrix}\right), \tag{3.80}$$

then

$$\begin{bmatrix} Y_t \\ M_{t|t-1} \end{bmatrix} \sim \mathcal{N}\left(\mu_h,\, \Sigma_h + \begin{bmatrix} \Sigma_y^\varepsilon & 0 \\ 0 & 0 \end{bmatrix}\right). \tag{3.81}$$

Application of the filtering update rules for the belief parameters (3.37) – (3.38) results in the physical state and posterior belief-mean being jointly Gaussian:

$$\begin{bmatrix} X_t \\ M_{t|t} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_t^x \\ \mu_{t|t}^m \end{bmatrix}, \begin{bmatrix} \Sigma_t^x & \Sigma_{t|t}^{xm} \\ \Sigma_{t|t}^{mx} & \Sigma_{t|t}^m \end{bmatrix}\right), \tag{3.82}$$

$$\mu_{t|t}^m = W\mu_{t|t-1}^h, \tag{3.83}$$

$$\Sigma_{t|t}^{mx} = W_y\Sigma_t^x + W_m\Sigma_{t|t-1}^{mx}, \tag{3.84}$$

$$\Sigma_{t|t}^m = W\Sigma_{t|t-1}^h W^\intercal + W_y\Sigma_y^\varepsilon W_y^\intercal, \tag{3.85}$$

$$W \doteq [W_y, W_m]. \tag{3.86}$$

Thus the hierarchically-uncertain updated belief posterior is $B_{t|t} \sim \mathcal{N}\left(M_{t|t}, \bar{V}_{t|t}\right)$, where $M_{t|t} \sim \mathcal{N}\left(\mu_{t|t}^m, \Sigma_{t|t}^m\right)$ and $\bar{V}_{t|t} = W_m\bar{V}_{t|t-1}$.

Control Decision

Simulated control decisions are a function of the (uncertain) belief-mean, as before in § 3.3.3,

$$(\mu_t^u, \Sigma_t^u, C_{m_{t|t}u}) \;=\; \pi(\mu_{t|t}^m, \Sigma_{t|t}^m, \psi), \tag{3.87}$$

with μ_t^u the output mean, Σ_t^u the output variance and $C_{m_{t|t}u} \doteq (\Sigma_{t|t}^m)^{-1}\mathbb{C}_{M_{t|t}}\left[M_{t|t},\, U_t\right]$. Thus, making a joint Gaussian approximation using moment matching, we have

$$\begin{bmatrix} X_t \\ M_{t|t} \\ U_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_t^x \\ \mu_{t|t}^m \\ \mu_t^u \end{bmatrix}, \begin{bmatrix} \Sigma_t^x & \Sigma_{t|t}^{xm} & \Sigma_{t|t}^{xm}C_{m_{t|t}u} \\ \Sigma_{t|t}^{mx} & \Sigma_{t|t}^m & \Sigma_{t|t}^m C_{m_{t|t}u} \\ C_{m_{t|t}u}^\intercal\Sigma_{t|t}^{mx} & C_{m_{t|t}u}^\intercal\Sigma_{t|t}^m & \Sigma_u \end{bmatrix}\right). \tag{3.88}$$

Filtering Prediction Step

Finally, the simulator must probabilistically predict:

1. the physical state $p(X_{t+1})$,

2. the belief-mean $p(M_{t+1|t})$,
3. the covariance between the two $\mathbb{C}\left[X_{t+1},\, M_{t+1|t}\right]$,
4. the expected belief-variance $\bar{V}_{t+1|t}$,

Both $p(M_{t+1|t})$ and $\bar{V}_{t+1|t}$ are computed as before using (3.61) – (3.67). The only new derivations required are to compute $p(X_{t+1})$ the way PILCO did, (2.40) – (2.45). We now discuss two ways of computing $\mathbb{C}\left[X_{t+1},\, M_{t+1|t}\right]$ (required to estimate future values of Σ^{xm} as used in (3.88) required for future prediction steps).

APPROXIMATE (BUT FAST) COMPUTATION OF $\mathbb{C}\left[X_{t+1},\, M_{t+1|t}\right]$: First, we define a set of linear transformations which preserves the input-output covariances between variables connected via generic (linear or nonlinear) functions using Appendix A.2.6:

$$
\begin{aligned}
C_{hx} &\doteq \Sigma_h^{-1}\mathbb{C}_{H_t}\left[H_t,\, X_t\right] &&= [I, 0]^\intercal, && (3.89)\\
C_{hm_{t|t}} &\doteq \Sigma_h^{-1}\mathbb{C}_{H_t}\left[H_t,\, M_{t|t}\right] &&= W^\intercal, && (3.90)\\
C_{hu} &\doteq \Sigma_h^{-1}\mathbb{C}_{H_t}\left[H_t,\, U_t\right] &&= C_{hm_{t|t}} C_{m_{t|t}u}, && (3.91)\\
C_{h\tilde{x}} &\doteq \Sigma_h^{-1}\mathbb{C}_{H_t}\left[H_t,\, \tilde{X}_t\right] &&= [C_{hx},\, C_{hu}], && (3.92)\\
C_{h\tilde{m}} &\doteq \Sigma_h^{-1}\mathbb{C}_{H_t}\left[H_t,\, \tilde{M}_{t|t}\right] &&= [C_{hm_{t|t}},\, C_{hu}], && (3.93)\\
C_{hx'} &\doteq C_{h\tilde{x}} C_{\tilde{x}x'} &&\approx \Sigma_h^{-1}\mathbb{C}_{H_t}\left[H_t,\, X_{t+1}\right], && (3.94)\\
C_{hm'} &\doteq C_{h\tilde{m}} C_{\tilde{m}m'} &&\approx \Sigma_h^{-1}\mathbb{C}_{H_t}\left[H_t,\, M_{t+1|t}\right], && (3.95)\\
C_{hh'} &\doteq [C_{hx'},\, C_{hm'}] &&\approx \Sigma_h^{-1}\mathbb{C}_{H_t}\left[H_t,\, H_{t+1}\right], && (3.96)
\end{aligned}
$$

where some terms were previously defined: $C_{m_{t|t}u}$ in (3.87), $C_{\tilde{x}x'}$ in (2.45), and $C_{\tilde{m}m'}$ in (3.65). Using the identity in Appendix A.2.6, we approximately compute our desired covariance between the new latent state and the new belief mean:

$$
\begin{aligned}
\mathbb{C}_{H_t}\left[X_{t+1},\, M_{t+1|t}\right] &\approx \mathbb{C}_{H_t}\left[X_{t+1},\, H_t\right](\Sigma_t^h)^{-1}\mathbb{C}_{H_t}\left[H_t,\, M_{t+1|t}\right]\\
&= C_{hx'}^\intercal \Sigma_t^h C_{hm'}. && (3.97)
\end{aligned}
$$

*EXACT (BUT SLOW) COMPUTATION OF $\mathbb{C}\left[X_{t+1},\, M_{t+1|t}\right]$: Alternatively, to compute $\mathbb{C}\left[X_{t+1},\, M_{t+1|t}\right]$ exactly[2] we combine both GP dynamics models f_x and f_b into one 'super' GP model. This is slightly complicated by the fact that X_{t+1} and $M_{t+1|t}$ use partially-overlapping subsets of $\{X_t, B_{t|t}, U_t\}$ as input. The derivation is quite tedious, so this section is marked optional for the reader. To simplify to problem, we

[2] Assuming the joint Gaussian assumption on $\{X_t, M_{t|t}, U_t\}$ is valid.

make use of the existing identities on Gaussian process predictions of hierarchical inputs (Appendix B.4) by defining 1) a single hierarchically-uncertain input vector $Z \doteq \{X_t, B_{t|t}, U_t\}$, and 2) a single hierarchically-uncertain output vector $\{X_{t+1}, B_{t+1|t}\}$. The hierarchical-distribution of input Z is known to be:

$$\underbrace{\begin{bmatrix} X_t \\ B_{t|t} \\ U_t \end{bmatrix}}_{Z} \sim \mathcal{N}\left(\underbrace{\mathcal{N}\left(\underbrace{\begin{bmatrix} \mu_t^x \\ \mu_{t|t}^m \\ \mu_t^u \end{bmatrix}}_{\mu^z}, \underbrace{\begin{bmatrix} \Sigma_t^x & \Sigma_{t|t}^{xm} & \Sigma_t^{xu} \\ \Sigma_{t|t}^{mx} & \Sigma_{t|t}^m & \Sigma_{t|t}^{mu} \\ \Sigma_t^{ux} & \Sigma_{t|t}^{um} & \Sigma_t^u \end{bmatrix}}_{\Sigma^z} \right)}_{M^z}, \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & \bar{V}_{t|t} & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{V^z} \right). \tag{3.98}$$

To preserve the individual predictions of both $X_{t+1} = f_x(\tilde{X}_t)$ and $B_{t+1|t} = f_b(\tilde{B}_{t|t})$ we expand each GP's lengthscale matrix Λ_k^x to match the size of Σ^z (3.98) whilst encoding the fact that X_{t+1} is conditionally independent of $B_{t|t}$ given X_t and U_t. We do this by creating a new (diagonal) lengthscale matrix where the elements corresponding to $B_{t|t}$ are rendered ineffectual by setting them to infinity. The associated linear model parameters and data points for each GP on dynamics models f_x and f_b are likewise rendered ineffectual by setting to zero. For the k th output of f_x, and l th output of f_b we have:

$$\Lambda_k^x = \text{diag}(\lambda_{kx}^x, \lambda_{ku}^x) \quad \rightarrow \quad \hat{\Lambda}_k^x = \text{diag}(\lambda_{kx}^x, \infty_D, \lambda_{ku}^x), \tag{3.99}$$

$$\Lambda_l^b = \text{diag}(\lambda_{lb}^b, \lambda_{lu}^b) \quad \rightarrow \quad \hat{\Lambda}_l^b = \text{diag}(\infty_D, \lambda_{lb}^b, \lambda_{lu}^b), \tag{3.100}$$

$$\phi_k^x = [\phi_{kx}^x; \phi_{ku}^x] \quad \rightarrow \quad \hat{\phi}_k^x = [\phi_{kx}^x; 0_D; \phi_{ku}^x], \tag{3.101}$$

$$\phi_l^b = [\phi_{lb}^b; \phi_{lu}^b] \quad \rightarrow \quad \hat{\phi}_l^b = [0_D; \phi_{lb}^b; \phi_{lu}^b], \tag{3.102}$$

$$X^x = [X_x^x, X_u^x] \quad \rightarrow \quad \hat{X}^x = [X_x^x, 0_D, X_u^x], \tag{3.103}$$

$$X^b = [X_b^b, X_u^b] \quad \rightarrow \quad \hat{X}^b = [0_D, X_b^b, X_u^b], \tag{3.104}$$

where each ∞_D and 0_D symbol above represents a D-length column vector of infinite and zero values respectively, and $[\cdot;\cdot]$ is a column-wise concatenation matrix operator. Now to compute the covariance of f_x k th output and f_b l th output given the common

uncertain input Z (3.98) of both $\mathcal{GP}$s, we use Appendix B.4.3:

$$\begin{aligned}
\Sigma^{xm}_{t+1|t} &= \mathbb{C}_Z\left[X_{t+1},\, M_{t+1|t}\right], \\
\mathbb{C}_Z\left[X^k_{t+1},\, M^l_{t+1|t}\right] &= \mathbb{C}_{M^z}\left[\mathbb{E}_{\tilde{X}_t}\left[\mathbb{E}_f\left[f^k_x(\tilde{X}_t)\right]\right],\, \mathbb{E}_{\tilde{B}_{t|t}}\left[\mathbb{E}_f\left[f^l_b(\tilde{B}_{t|t})\right]\right]\right] \qquad (3.105)\\
&= s^{x2}_k s^{b2}_l\left[\beta^{x\intercal}_k(\hat{\hat{Q}}_{kl} - q^x_k q^{b\intercal}_l)\beta^b_l\right] + \hat{C}^{x\intercal}_k\Sigma^z\hat{\phi}^b_l + \hat{\phi}^{x\intercal}_k\Sigma^z\hat{C}^b_l + \hat{\phi}^{x\intercal}_k\Sigma^z\hat{\phi}^b_l,
\end{aligned}$$

where $\hat{C}$ is computed using (B.40) and

$$\begin{aligned}
q^x_{ki} &= q(\hat{X}^x_i, \mu^z, \hat{\Lambda}^x_k, \Sigma^z + V^z), \quad &&= q(X^x_i, \mu^x_t, \Lambda^x_k, \Sigma^{\tilde{x}}_t), && (3.106)\\
q^b_{li} &= q(\hat{X}^b_i, \mu^z, \hat{\Lambda}^b_l, \Sigma^z + V^z), \quad &&= q(X^b_i, \mu^m_{t|t}, \Lambda^b_l, \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t}), && (3.107)\\
\hat{\hat{Q}}_{klij} &= Q\big(\hat{X}^x_i, \hat{X}^b_j, \hat{\Lambda}^x_k, \hat{\Lambda}^b_l, V^z, \mu^z, \Sigma^z\big). && && (3.108)
\end{aligned}$$

3.4.2 *Controller Evaluation and Improvement

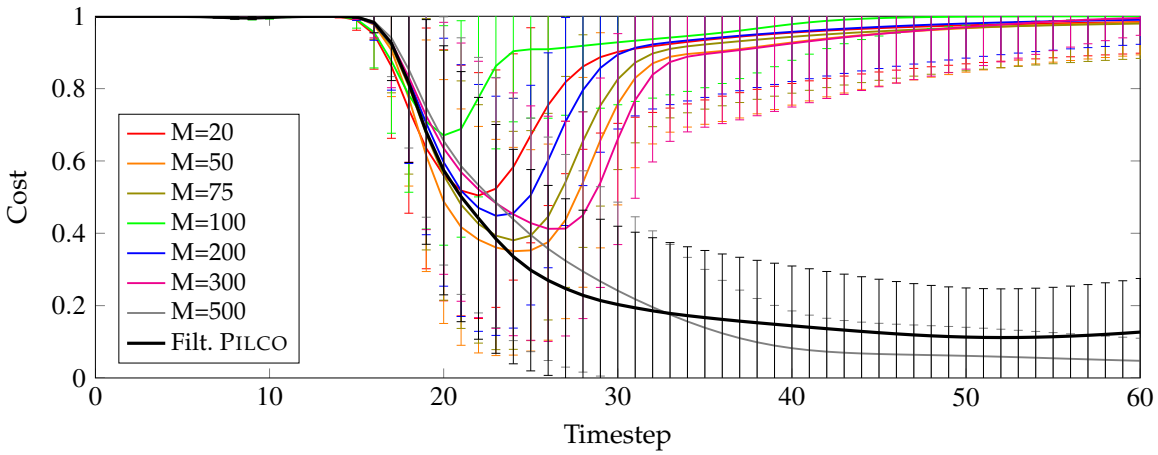
Thus far, we have discussed a one step prediction from $S_t \doteq [X_t^\intercal,\, M^\intercal_{t|t-1},\, \mathrm{vec}(\bar{V}_{t|t-1})^\intercal]^\intercal$ to $S_{t+1} \doteq [X^\intercal_{t+1},\, M^\intercal_{t+1|t},\, \mathrm{vec}(\bar{V}_{t+1|t})^\intercal]^\intercal$, where

$$p(S_t) = \mathcal{N}\left(\mu_s = \begin{bmatrix} \mu_x \\ \mu_m \\ \mathrm{vec}(\bar{V}_{t+1|t}) \end{bmatrix}, \Sigma_s = \begin{bmatrix} \Sigma_x & \Sigma_{xm} & 0 \\ \Sigma_{mx} & \Sigma_m & 0 \\ 0 & 0 & 0 \end{bmatrix}\right). \qquad (3.109)$$

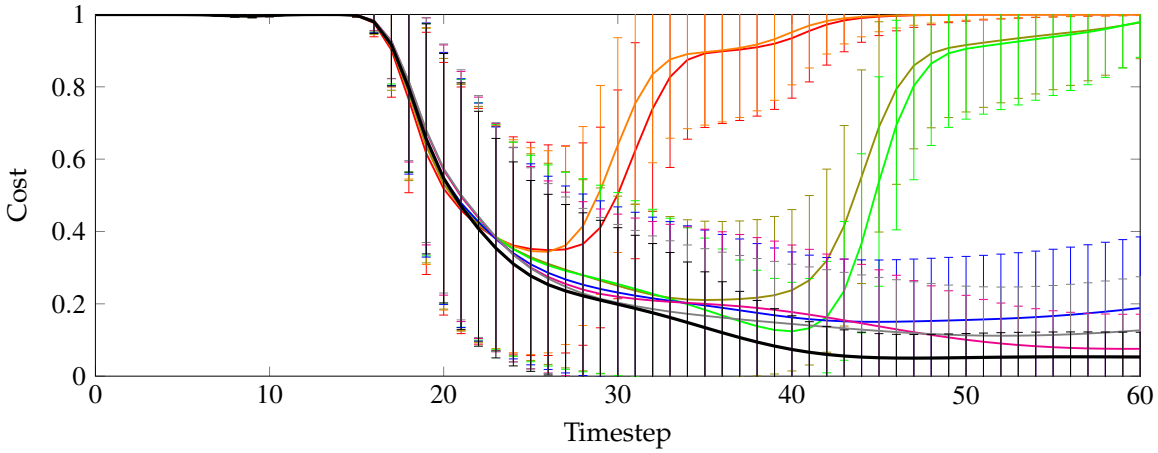
We use the same process to simulate forwards from S_{t+1} to S_{t+2} etc. up to S_T. Controller evaluation and improvement is analogous to § 3.3.4. A controller is then evaluated using by applying the cost function at each timestep: $J(\psi) = \sum_{t=0}^T \gamma^t \bar{c}_t$ where $\bar{c}_t = \mathbb{E}_{X_t}[\mathrm{cost}(X_t)|\psi]$. The controller is then improved using analytic gradients of the loss $J(\psi)$ w.r.t. controller parameters ψ, $\mathrm{d}J/\mathrm{d}\psi$. To compute $\mathrm{d}J/\mathrm{d}\psi$ we require derivatives $\mathrm{d}\bar{c}_t/\mathrm{d}\psi$ and thus $\mathrm{d}p(S_t)/\mathrm{d}\psi$, at each time t.

3.4.3 Demonstration

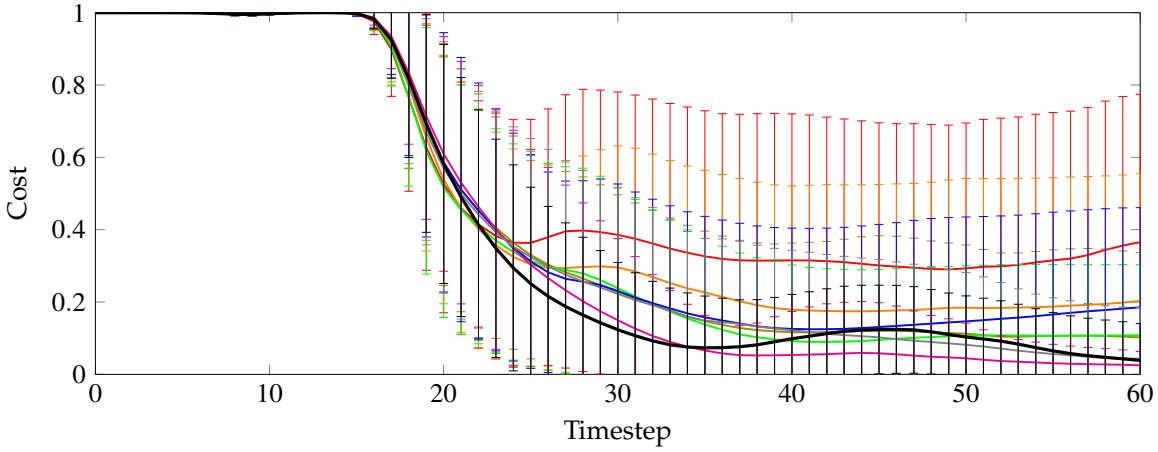
Here we demonstrate how various accurate-but-slow dynamical models f_x can evaluate systems using various inaccurate-but-fast dynamical models f_b, shown Fig. 3.14. We used a common controller, the controller optimised for our 'Filtered PILCO' in § 3.3.6, and several dynamics models for both f_x and f_b. The set of dynamics models comprised the model f used in § 3.3.6, and additionally seven more models with various numbers of inducing points: $\mathrm{M} = \{20, 50, 75, 100, 200, 300, 500\}$. Let $\mathcal{F}$ define the set of all eight dynamics models. We trained the models starting



(a) Predictive costs per timestep, according to each method itself.



(b) Predictive costs per timestep, according to 'Our Method'.



(c) Empirical costs per timestep, from 100 executions per algorithm.

Fig. 3.14 ***Controller evaluation using latent-variable belief-MDPs.*** *Each subfigure compares costs (predictive or empirical) of a common controller but different filtering-dynamics models f_b, applied to the cart-pole system with observation noise. For a common controller, we use the controller optimised by 'Filtered* PILCO *' from § 3.3.6. For different models f_b, we compared 'Filtered* PILCO*' (black) and several other GP dynamics models (coloured) with different numbers of inducing points M. All error bars show ±1 standard deviations, either of propbabilistic predictions or empirical spread. Each subfigure shares the same legend in Fig. 3.14a.*

with 500 randomly selected datapoints from the dataset collected in our previous experiment in § 3.3.6. Then we optimised the inducing point locations, resulting in our M = 500 model. Afterwards, we randomly moved some inducing points, leaving 300 remaining, re-optimised inducing point location, resulting in our M = 300 model, and so on. As a reminder, the more inducing inputs, the more accurate the model, and the slower the predictions are to compute. Finally we simulated various combinations of models for f_x and f_b, using the 'exact but slow' computation of $\mathbb{C}\left[X_{t+1}, M_{t+1|t}\right]$ discussed page 81.

The results of our simulations are shown Fig. 3.14. Three subfigures are included. First, Fig. 3.14a shows probabilistically predicted costs using the old belief-MDP framework, where $f_x = f_b = f$, with displayed costs $\forall f \in \mathcal{F}$. Second, Fig. 3.14b shows probabilistically predicted costs using the new latent-variable belief-MDP framework, where $f_x =$ Filtered PILCO, with displayed costs $\forall f_b \in \mathcal{F}$. Third, Fig. 3.14c shows empirical distribution of costs, where f_x is not used (since empirical, not simulated) and with displayed costs $\forall f_b \in \mathcal{F}$.

The results offer some interesting insights. First, when each model is used for both simulation and filtering, most models were overly pessimistic about their performance, seen Fig. 3.14a. Every model except Filtered PILCO and a 500 inducing point model falsely predicted the common controller was unable to control the noisily-observed cart-pole system. Second, by using a more accurate model for simulation in Fig. 3.14b, $f_x =$ Filtered PILCO, we see that predictions improved dramatically, and all models with at least 200 inducing points were correctly predicted to control the cart-pole well. Even filters using 75 or 100 inducing points were recognised to be much more useful than less accurate models of 20 or 50 inducing points. Third, we see the ground truth in Fig. 3.14c that in fact all models in $\mathcal{F}$ are adequate for filtered control, although generally the more inducing points the better the performance.

Given all models $\mathcal{F}$ evidently stabilise the cart-pole (although, some better than others) seen Fig. 3.14c, it is interesting f_x only predicted the more accurate models in Fig. 3.14b could control the system. The chosen model f_x is not perfect, yet there appears some interplay, or accumilation of inaccuracies between f_x and the less-accurate f_b models where $\mathrm{M} \leq 200$ causing the prediction of failure for all f_b models with $\mathrm{M} \leq 200$. Currently we are unsure why this is exactly, and is left for future work.

Also of note is that model from Filtered PILCO used for f_x itself only uses M = 50 inducing points, yet outperforms most other models here. We were currently unsure

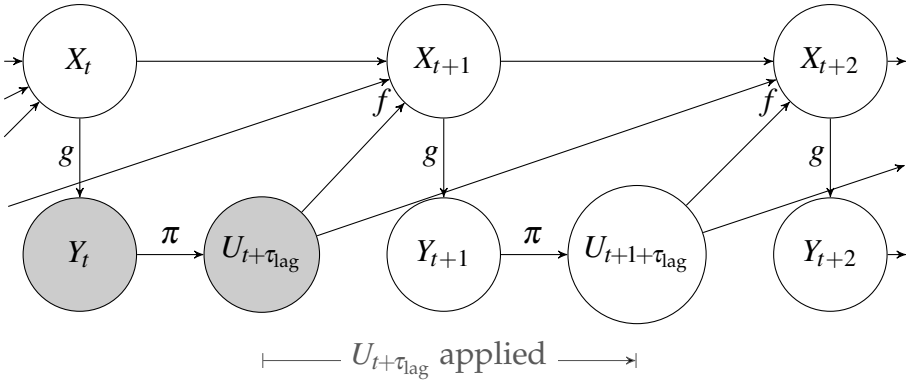


Fig. 3.15 ***Sensor time delay****: the* PILCO *framework with sensory time lag. At each timestep, the system state X_t is observed as Y_t. The observation Y_t records the state at time t, however, requires a non-zero amount of time (τ_{lag}) to be processed and transmistted to controller π. Consequently, the controller only receives observation Y_t at time $t + \tau_{lag}$, outputting a new control signal $U_{t+\tau_{lag}}$. The control $U_{t+\tau_{lag}}$ influences part of the state transition from $X_t \to X_{t+1}$, as per the original* PILCO*. The difference now, is our zero-order-hold controller continues to apply $U_{t+\tau_{lag}}$ until a new observation Y_{t+1} is recieved. Since Y_{t+1} is not recieved until time $t + 1 + \tau_{lag}$, then $U_{t+\tau_{lag}}$ additionally affects the following state transition $X_{t+1} \to X_{t+2}$. Thus, control $U_{t+\tau_{lag}}$ affects two state transitions: $X_t \to X_{t+1}$ and $X_{t+1} \to X_{t+2}$. From the perspective of the single transition $X_{t+1} \to X_{t+2}$, control $U_{t+\tau_{lag}}$ was applied the first τ_{lag} fraction of the timestep, then $U_{t+1+\tau_{lag}}$ for the remaining $1 - \tau_{lag}$ fraction.*

why this is, but may be to do with the fact that f_x we re-optimised over 11 episodes in § 3.3.6, potentially finding better local optima for inducing point locations compared to each f_b tested here was only optimised once.

3.5 *Additional Topics

Additional topics were experimented with briefly. In particular our code was written to handle flexible state representations, especially N-Markov representations, which included previous control outputs useful for sensory time delay.

3.5.1 Sensor Time Delay

During some brief work with Martin Kukla using a video camera to capture motions of a real cart-pole robot, we noticed a time delay when observing the robot. Each frame recorded by the camera was first transmitted via cable to a local computer, which then processing the image using OpenCV from pixels to an estimated state vector y_t. The time between acting on y_t and when the frame was recorded was

approximately 30ms. A 30ms lag is significant. From our experiment in § 3.3.5, 30ms almost corresponds to one whole timestep.

Sensory time delay breaks the Markov state assumption: that predicting X_{t+1} given data up until time t was conditionally independent of all other variables given $\tilde{X}_t = [X_t^\intercal, U_t^\intercal]^\intercal$. Now, because of time lag, older controls affect X_{t+1} too. Fig. 3.15 shows how a model of unfiltered control can account for time delay.

As long as the time lag is within a timestep, $\tau_{\text{lag}} \leq \Delta t$, then our solution to regain the state's Markov property is to include the previous control output, defining a $\mathrm{X}+\mathrm{U}$ dimensional state. In § 3.3.5 for example we actually used the following state representation: $x_t = [u_{t-1}, x_t^c, \theta_t, \dot{x}_t^c, \dot{\theta}_t]^\intercal$. For further reading, an excellent analysis of optimal control with *random* time delays is given by Nilsson (1998).

3.5.2 Alternate State Representations

Previously, in the cart-pole system we used a state representation x consisting four state variables, two position variables, and two time-derivative variables: $x_t = [x_t^c, \theta_t, \dot{x}_t^c, \dot{\theta}_t]^\intercal$. When using a camera's sensor, any speed-based or time-derivative variables are not directly observable. Only a sequence of frames encoding position variables is directly observable. One solution is to use finite differences between frames to estimate speeds. Another solution is a 2-Markov state representation. We briefly looked at the 2-Markov representations, e.g. : $x_t = [x_{t-1}^c, \theta_{t-1}, x_t^c, \theta_t]^\intercal$.

We also discussed filtering in this chapter, as an alternative to inputting a noisy observation $y_t = [x_t^c, \theta_t, \dot{x}_t^c, \dot{\theta}_t]^\intercal$ directly into a controller. We did so using a nontrivial method of simulating a filtering process. A conceptually-easier alternative solution is to use a more general N-Markov state representation. Since filtering is about summarising the complete history of observations and control $\{y_{0:t}, u_{0:t-1}\}$, N-Markov state representations can act as a substitute for filtering since they condition on the N most recent observations, the N most important for observations for estimation of latent X_t. E.g. we could simply train an unfiltered controller with modified inputs of: $y_t = [u_{t-N}, x_{t-N+1}^c, \theta_{t-N+1}, ..., u_{t-1}, x_t^c, \theta_t]^\intercal$. The advantage is easy implementation. The disadvantage is the state-dimensionality X increases, which considering PILCO scales $\mathcal{O}(\mathrm{X})$, is slower computation and reduced data efficiency. By increasing the dynamics model input dimensionality by N-fold, we require exponentially-in-N more data to capture the dynamics before obtaining good controllers.

3.6 Discussion and Future Work

In this chapter, we extended the original PILCO framework (Deisenroth and Rasmussen, 2011) to learn locally-optimal controllers with *filtered* observations as input. Our extended framework enabled learning in *partially-observed* environments (POMDPs, also known as belief-MDPs) without sacrificing PILCO's high data efficiency. Our algorithm – Filtered PILCO – achieved high data efficiency by evaluating controllers using simulation faithful to reality. As a reminder, system simulation involves multi-step probabilistic prediction from time $t = 0$ to $t = T$ required for controller evaluation and subsequent improvement. Just as PILCO faithfully simulates closed-loop control because it executed closed-loop control, we similarly simulated closed-loop filtered control precisely because we execute closed-loop filtered control. By contrast, methods such a MPC conducted simulation not faithful to reality, by executing closed-loop control and inconsistency simulating open-loop.

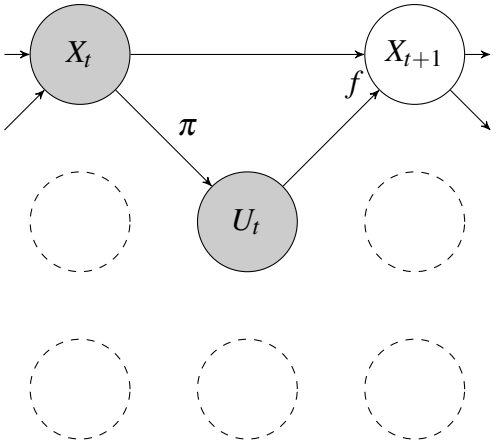
We began this chapter exploring the theoretical consequences of observation noise in § 3.1. Observation noise affects both modelling of the dynamics and also injects noise into the controller's input. In § 3.1.2 we analysed the stability of systems, in the context of controllers with unfiltered observations as input, to determine some necessary conditions of system stability. We revisited our stability analysis later in § 3.3.9, for the context of filtered control. A comparison of the two showed the theoretical benefit that filtering brings to the conditions of system stability, and how much more observation noise a filtered controller can accept before the system destabilises. For a more comprehensive analysis of stability of PILCO we refer the reader to Umlauft (2014, chapter 5).

We demonstrated experimentally the benefits of simulating the filtering process by applying our filtered-PILCO algorithm to a benchmark control problem: the noisily-observed cartpole swing-up in § 3.3. For comparison, we also applied other algorithms which either did not filter, including PILCO, or only filtered during system execution. We showed experimentally in § 3.3.5 that *faithful* and *probabilistic* simulation gives greater performance gains than otherwise. This is because the controller optimisation process – when aware of the filtering process – optimises a controller for the specific circumstance in which it is used, namely: filtered closed-loop control. We retained PILCO's data efficiency property, and learned a controller more effectively than the original PILCO algorithm, under significant observation noise in just 22 seconds of system interaction.

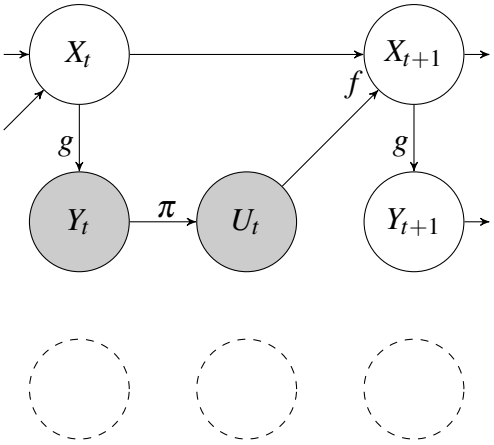
For clear comparison of each algorithm, our first variational of the experiment (§ 3.3.6) constrained each algorithm to use the same dynamics dataset rather than each algorithm interacting with the system to generate their own. By isolating the dataset, we showed superior data-*usage* (to ensure any superior results were not caused by superior data-*collection* abilities of any method). By controlling the dataset, we also avoided the extra variance in empirical performance caused by selection of different data. Next we relaxed the experimental constraint of a common dataset in § 3.3.7 and clearly saw the our Filtered PILCO learned with more data efficient under the noisy conditions than the competing algorithms. Afterwards, in § 3.3.8 we examined some of the range of noise levels that both PILCO and Filtered PILCO handle.

Later, in § 3.4 we further generalised our framework from § 3.3 of control in 'belief MDPs' to 'latent variable belief MDPs', where both belief states and latent system states are modelled. The generalisation's main benefit is the use of two different dynamics models, one for the belief-state filtering, and another for the latent state forwards prediction. In belief-MDPs, both these dynamics models are identical, in many cases, such as ours, real-time constraints can incentivise two dynamics models. Since the belief-state filtering occurs in both the system execution and simulation phase, the belief filtering computations must be fast to occur in real-time during system execution. For fast computation, we may need to sacrifice model accuracy, using an inaccurate-yet-fast dynamics model f_b. By contrast, forwards simulation of the latent state only occurs during the offline system phase of the PILCO Algorithm 1. Thus, predictions concerning latent state x can afford to use an accurate-yet-slow dynamics model f_x, which is perhaps so slow it cannot be used in real-time. The advantage is the accurate model f_x can be used to simulate how the inaccurate dynamics model f_b will react to accurately-predicted future state distributions X_{t+1}. Such a situation is clearly more favourable than simulating how an inaccurate model reacts to an *also*-inaccurate predicted state. A secondary benefit is an ability to understand the consequences of the robot's bad prior assumptions, where the robot's prior of states at time $t = 0$ is different from the initial state distribution. Latent variable belief MDPs are a generalisation of the belief-MDP, and thus a generalisation of POMDPs, proved Appendix E.6.

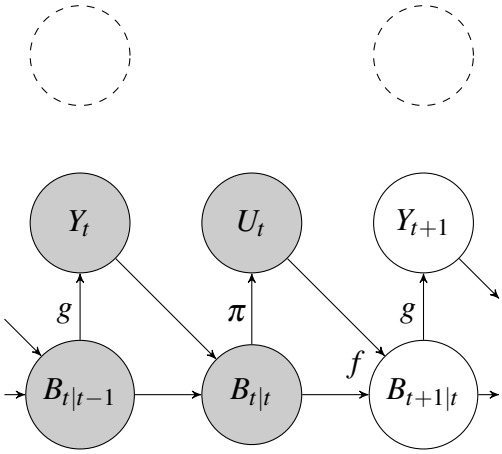
After introducing three variants of PILCO, each modelling the probabilistic process of control differently, we briefly compare each, seen side-by-side in Fig. 3.16. The original PILCO (Fig. 3.16a) models noiseless control, introduced in the previous chapter. In the current chapter, we first analysed unfiltered control (Fig. 3.16b) to



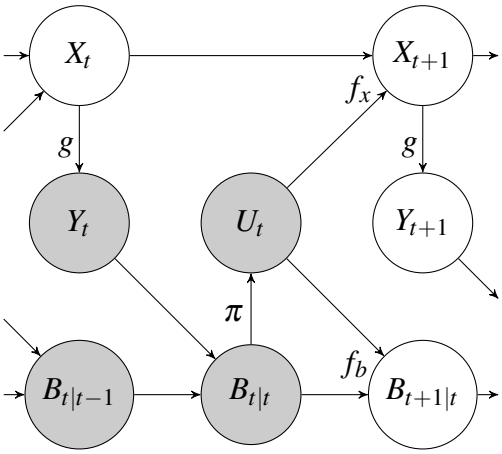
(a) Noiseless Control (PILCO), from § 2.5



(b) Unfiltered Control, from § 3.1



(c) Filtered Control for Belief-MDPs, from § 3.3



(d) Filtered Control for Latent-Variable Belief-MDPs, from § 3.4

Fig. 3.16 ***Comparison of Models.*** *Four variants of the* PILCO *algorithm, each modelling the process of control differently, reflected in the different PGMs each use. In each figure, eight possible variables are either modelled (solid circles) or not modelling (dashed circles). Solid circles which are white are not observed (either not yet, or never), with grey signifying an observed variable relative to current time t (also $t|t$).*

understand the affects of injecting observation noise into a controller. Then, we exchanged the controller's input from observations to beliefs, framing control with belief-MDPs (Fig. 3.16c). In belief MDPs, the belief B is a sufficient statistic of the latent state X. Since modelling X is thus redundant, we can simulate future observations as if they were generated from the belief B. Finally, we reintroduce the latent state X for the specific case where the belief B ceases to be a sufficient statistic of latent state X, a new Markov process of control we called 'latent variable belief-MDPS' (Fig. 3.16d). The belief ceases to be a sufficient statistic either when we have differing dynamics models, $f_b \neq f_x$, or when the total variance of prior belief $B_{0|0}$ differs from the initial state variance: $\Sigma^m_{0|0} + V_{0|0} \neq \Sigma^x_0$. In either case, the observation Y must revert to being generated from the latent state X, not what the robot believes the state to be B. The process in Fig. 3.16d is an explanation of the *causal* relationship between variables. It is the latent variable X, which causes an observation Y, not the belief as in Fig. 3.16c. However, PGMs are *not* representations of causal relationships, and should not be interpreted as such. PGMs only show conditional dependency structure between variables.

Several additional challenges remain for future work. Firstly, when randomising the belief from $b \sim \mathcal{N}(m, V)$ to $B \sim \mathcal{N}(M, \bar{V})$, we did so by randomising the belief-mean m, not the belief-variance V. We instead used a delta distribution for $\bar{V} \sim \mathcal{N}(\mathbb{E}[V], 0)$. Instead, a better approach would be to relax our assumption of zero variance of the belief-variance parameters. A relaxation allows distributed trajectories to more accurately consider distributions over belief states that have various degrees of certainty (belief-variance) at a given future timestep. Because of our restriction, filtered-PILCO can only consider one possible belief-variance at future timesteps. This restriction causes a worryingly unknown amount 'unfaithfulness' in our simulation, going against PILCO's principle of being faithful to reality. In our simulation, at a particular timestep, the belief-variance is predicted to be consistent no matter which belief-state we sample from our analytic distribution over belief-states. However, in reality, the belief-state variance depends heavily on whether the state trajectory passes through data-dense, or data-sparse regions of state-space. During execution through data-sparse regions, a filter relies more on observational information, resulting in a larger observation weight matrix W_y and larger belief-variance. Conversely, data-dense regions result in smaller W_y and smaller belief-variance. By approximating the belief-variance to be a delta distribution, our simulation can only consider a single weight W_y per future timestep, across various system trajectories, even though the some of those trajectories require small

W_y, other large W_y. Thus, because of our approximation, our controller optimisation must find an unnecessary balance of W_y, unlikely suited for any one sampled trajectory in particular. A better solution may be to consider distributions over positive semi-definite variance V using the Wishart distribution. An open question is if the Wishart distribution could be combined with our analytical filtering framework in a tractable way.

A second promising avenue for future work, and perhaps the lowest hanging fruit adapting the controller to be a function of the full belief distribution (mean and variance). Currently, out methods only uses the belief-mean. Such additional flexibility could enable the controller to react more 'cautiously' when uncertain about the state (large V). Conversely a controller could then react more 'boldly' when very certain about the state (small V). Unlike integrating Wishart distributions into our filtering framework, expanding the controller's input to also include belief-variance would likely require less work and may notably improve controller performance.

Finally, the use of numerical quadrature (as done by Vinogradska et al. (2016)) may provide more accurate simulations of long term state distributions than our moment matched ADF approach, and should be tested in future work.

Chapter 4

Directed Exploration for Improved Data Efficiency

Learning control of unfamiliar systems requires optimising a controller w.r.t. the current state of system knowledge whilst the system knowledge changes as a function of the controller's output, a problem known as *dual control* (Wittenmark, 1995). 'Dual control' refers to the situation of seemingly having two goals: 1) system identification and 2) controller optimisation. In RL, the situation is framed as a dilemma between choosing controls u at a particular state x that either *explore* more (to improve system knowledge, improving future episode losses indirectly) or *exploit* more (directly reducing the current episode's loss based on current system knowledge). Both types of behaviour are important since both contribute to minimising the total loss summed over all episodes. Many control methods depend heavily on the quality and the extent to which the system has been identified, before an effective controller can be designed, or a good filter implemented. The difficulty is ascertaining how much system ID is required: too little and we may not understand how to control the system well, too much and waste precious system interaction learning about the system but not controlling it (our ultimate concern). In this chapter, we extend PILCO to the dual control setting. We note that PILCO is already an *active* machine learning algorithm, meaning PILCO's outputs influence the data it collects in the future, but it is nevertheless a greedily active learner. Until now, PILCO has only exploited, greedily minimising the current episode's loss only, without any regard to the benefit of exploratory type behaviour which could drastically improve the losses in future episodes.

Data-efficient learning of control involves both 1) modelling aspects of the system (e.g. the dynamics f or sensor g), and 2) balancing exploration and exploitation.

Already we have discussed the first point, concerning dynamics modelling. This chapter investigates the second point. Simple ways to implement exploratory behaviour in a controller is by occasionally outputting random controls. The idea is the robot will hopefully test most state-control combinations eventually, not just those which appear to be optimal based on limited knowledge. However, such *undirected* random exploration (e.g. ε-greedy or Boltzmann exploration) can be data-intensive, opposed to *directing* exploration towards *known unknowns*, for deliberate and systematic reduction of subjective system uncertainties. We extend PILCO to using directed exploration, which generates non-repetitive datasets relevant to rapidly identifying optimal controllers.

This chapter proceeds as follows. First we introduce undirected exploration in § 4.1. We then discuss how uncertainty information can help direct exploration in § 4.2, a more date, exploration can be aimed to reduce uncertainty in our dynamics model $p(f)$. Better exploration strategies instead reduce uncertainty in our objective directly, in our case: cumulative-cost. Our main contribution this chapter is directing PILCO's exploration using the full cumulative-cost distribution (§ 4.4) using Bayesian optimisation (BO). We discuss four different BO methods, including our own approximation of the Gittin's index in § 4.3. Experiments to compare against PILCO using PILCO's original cart double-pole swing-up task are also given. Afterwards, we discuss how improvements can be made by distinguishing between different sources of uncertainty behind the cumulative-cost variance in § 4.5.

4.1 UNDIRECTED EXPLORATION

The exploitation-exploration trade-off characterises a perceived dilemma between controlling a system to minimise expected costs immediately (exploitation); against informative control decisions to later improve the controller, minimising future costs (exploration). A balance between exploitation and exploration generally performs better than a pure exploitation or pure exploration approach. Below we discuss two prominent examples of undirected exploration popular in the early days of RL. Both use are stochastic controllers, denoted $\pi(u|x)$, whose outputs are partly random.

Our first example of exploration is the classic ε-greedy heuristic which randomly alternates between two distinct controller behaviours:

- Exploration: select a control uniformly at random.
- Exploitation: select control of least loss.

The overall controller then outputs exploratory controls $100\varepsilon\%$ of timesteps, and exploitatory controls $100(1-\varepsilon)\%$ of timesteps:

$$\pi_{\varepsilon\text{-greedy}}(u|x) = \varepsilon \cdot \underbrace{\mathcal{U}^u}_{\text{explore}} + (1-\varepsilon) \cdot \underbrace{\delta(u - \pi^*(x))}_{\text{exploit}}, \tag{4.1}$$

where we use $\mathcal{U}^u$ as the uniform distribution over a bounded space of possible controls u, and δ is the delta distribution. Typical ε values range [0.01, 1.0], either remaining constant or reducing over time Mnih et al. (2015). By decaying ε over time t, most exploration occurs early on, a desired behaviour since more time steps remains to capitalise on information learned. With time running out, the robot then focuses on exploiting its knowledge.

An alternative to the ε-greedy exploration is using the Boltzmann distribution. The Boltzmann distribution does not neatly swap between two modes, 'exploration' and 'exploitation', but blurs the lines between both behaviours by simply increasing the probability of choosing controls which have less loss:

$$\pi_{\text{Boltzmann}}(u|x) = \frac{\exp\left(kJ_t^*(x,u)\right)}{\sum_{u'}\exp\left(kJ_t^*(x,u')\right)}, \tag{4.2}$$

where Boltzmann's constant k regulates the balance between exploration and exploitation, and $J_t^*(x,u)$ was defined (2.8). For example, $k=0$ corresponds to a uniform distribution as $\varepsilon = 1$ does (pure exploration), whilst $k \to \infty$ corresponds to pure exploitation, always choosing the control of least predicted loss, as did $\varepsilon = 0$.

Boltzmann exploration is often preferable to ε-greedy since it considers all control-conditional loss-estimates $J_t^*(x,u)$, i.e. across multiple possible controls u (Sutton and Barto, 1998, Section 2.3). By contrast ε-greedy only considered one loss-estimate during exploitation: $u = \pi^*(x) = \text{argmin}_{u'} J^*(x,u')$. Thus Boltzmann exploration can distinguish between the 'second best control' and the 'third best' (whereas ε-greedy cannot) for more informed exploration. Nevertheless, both algorithms perform poorly in settings with a large number of states, due to a negligence of known uncertainties about the system. Each may 'explore' aspects of the system already known to the dynamics model, resulting in wasted opportunities to gather information. Without uncertainty information for direction, such strategies are termed *undirected* exploration. The only attraction of both strategies is they are simple to implement, and thus still used in current RL research (Mnih et al., 2015).

4.2 Directed Exploration

Undirected random exploration (e.g. ε-greedy exploration) or exploration that is only a function of the loss function (e.g. Boltzmann exploration) are not fully informed of the robot's state of knowledge. A reasonable representation of the robot's subjective uncertainties about the dynamics or the cumulative cost can be used to *direct* exploration for faster learning. In finite state spaces for example, directed exploration learns optimal controllers with an amount of data polynomial in the number of states, opposed to undirected exploration, which is instead exponential in the number of states (Thrun, 1992). We are thus more interested in extending PILCO with directed exploration.

Before we can proceed, we should clarify what our overall objective is. So far, we have only discussed exploration and exploitation in vague terms for intuition. In our episodic problems, the controller we choose at the e'th episode must balance: 1) optimising loss for the current episode J_0^e (exploitation), and 2) gathering information that help to optimise future losses $J_0^{e+1:E}$ (exploration). In § 2.4 we defined a single objective of data-efficient algorithms, which combines both desiderata above, from seemingly two objectives into one, copied below for reader convenience:

$$\mathbf{J} \doteq \sum_{e=1}^{E} J_0^e = \sum_{e=1}^{E} \sum_{t=0}^{T} \gamma^t c_t^e. \tag{4.3}$$

Thus, balancing exploration and exploitation can simply be framed as a single objective: minimisation of the total loss $\mathbf{J}$, the loss summed over all episodes. This is equivalent to the cumulative regret objective from the bandit literature. Other objectives are possible, such as *simple regret* from the bandits literature where only the final loss matters $\mathbf{J} = J_0^E$. However, we will only concentrate on *cumulative regret* of (4.3). Notice also that (4.3) has the benefit of being parameter-free. There is no ε constant or Boltzmann parameters k to tune a balance between exploration and exploitation (noting E and T are defined by the task, not parameters the robot tunes). This is because (4.3) evaluates exploration – the value of information – in the same units as exploitation[1] (being the units of the objective function).

Given our objective for data-efficient control, to minimise $\mathbf{J}$, which implicitly balanced exploration and exploitation (resolving RL's perceived dilemma), we move on to discussing how to minimise $\mathbf{J}$ given our limited knowledge and thus uncertainties of the dynamics $p(f)$.

[1] Or more colloquially: we compare apples with apples, not apples with oranges.

4.2.1 Bayesian Reinforcement Learning

The total loss in (4.3) comprises E-many episodes, in between which we have the opportunity to update the controller's parameterisation according to data collected from the episode before. We wish to optimise $\mathbf{J}$ w.r.t. to our subjective belief $p(f)$, noting that $p(f)$ changes given new data. To capture how to optimise $\mathbf{J}$ given changing $p(f)$, an RL solution proposed by Duff (2002) is to define partially-observed hyperstates h which concatenate the original state x with partially-observed transition model parameters. Doing so, the reinforcement *learning* problem in x-space is framed as a POMDP (see § 2.2.2) *planning* problem in h-space. The resultant 'Bayes-optimal' controller, which optimises (4.3) w.r.t. both the present model $p(f)$ and importantly: knowledge of how the model updates given new data. Given knowledge of how the model updates, the POMDP solution can estimate the expected value of information of any type of exploratory behaviour, naturally trading off exploration and exploitation in the original x state space. The Bayes-optimal solution is 'optimal' in the sense of optimising the average total loss, averaged over many experiments re-sampling the dynamics parameters from the belief prior $p(f)$ at $e = 0$. Unfortunately, since POMDPs are generally intractable to solve exactly (Mundhenk et al., 1997; Papadimitriou and Tsitsiklis, 1987), so too are Bayes-optimal controllers. This is especially true in continuous state-action-observation spaces, and in PILCO's case using a nonparametric dynamics model, but approximations exist which we now discuss.

Bayesian Reinforcement Learning (BRL) algorithms model the dynamics and/or the rewards, or value function, to approximate the (intractable) Bayes-optimal controller. Many approximate approaches exist, including myopic belief lookahead (which considers how the belief updates over at most one episode into the future) and deeper lookahead. Myopic lookahead methods apply a heuristic function (or 'exploration bonus') to the robot's current uncertainties, or predicted uncertainties by next episode, an improvement on simpler methods such as Boltzmann-exploration (Meuleau and Bourgine, 1999) including sums of cost standard deviations (Deisenroth, 2009), Shannon entropy (Deisenroth et al., 2008), and a variants on the expected-improvement heuristic (Dearden et al., 1998; Delage and Mannor, 2007). For simplicity, myopic exploration often ignores loss correlations between different control decisions (or in our case between choosing controllers for the next episode) which are in fact correlated via the Bellman equation (2.8). Another method is modelling loss distributions using GPs (Engel et al., 2003, 2005) and exploring according to

upper confidence bounds, suitable for small state spaces, and smooth loss functions, unfortunately inapplicable to PILCO.

Deeper (non-myopic) sparse-tree lookahead trees are also possible (Kearns et al., 2002; Ross et al., 2008; Walsh et al., 2010; Wang et al., 2005). Alternatives to sampling can include discretisation of the belief space (Wang et al., 2012). However, this is only feasible up to 3–4 dimensions. Discretisation must be carefully tuned: if too large the robot will not learn anything, and if too small the learning will be slow. If the dynamics model is easily sampled, then following an optimal controller of a sample from the model for several timesteps, called Thompson sampling (Thompson, 1933), is an effective solution still popular in RL (Asmuth et al., 2009; Gal and Ghahramani, 2015; Osband et al., 2016; Stadie et al., 2015; Strens, 2000). Thompson sampling generally outperforms ε-greedy and Boltzmann exploration since the sampling the uncertain model posterior corresponds to uncertainty-directed exploration.

We can think of PILCO (§ 2.5) as another BRL algorithm, which assumes fully observable states x and a partially observable transition model $p(f)$. PILCO approximates Bayes-optimal control by assuming no observation function associated with the transition model (i.e. assuming its current uncertainty over transition models is fixed) and is a pure exploitation RL algorithm. However, even though PILCO does not intentionally explore the probabilistic trajectories, the saturating cost function has the indirect effect of favouring more uncertain polices when the expected cumulative-cost is poor (Deisenroth et al., 2015). This above effect together with system randomness (observation noise ε_t^y and process noise ε_t^x) usually ensure PILCO visits enough of the state space 'accidentally' to learn enough dynamics to optimise a reasonable controller. Nevertheless, PILCO achieved unprecedented data efficiency in the cart double-pole swing-up problem. As mentioned before, the key to PILCO's success is the use of its probabilistic nonparametric dynamics model. The probabilistic nonparametric model enables probabilistic predictions from model uncertainty (distinct from state uncertainty) from arbitrarily complex models, helpful for uncertainty-directed exploration. Earlier work by Deisenroth (2009, section 3.7.1) optimises the sum of the individual cost means and (weighted) cost standard deviations, as an approximate myopic BRL extension of PILCO. However, as they state, the sum of individual cost standard deviations is an approximation to using the full cumulative cost variance, which should include many cross terms, discussed § 4.4.

We build on this BRL extension of PILCO, to take PILCO from a pure exploitation algorithm to one that balances exploration and exploitation to achieve even greater

data efficiency than before. PILCO greedily optimises an expected cumulative-cost of the states using a probabilistic dynamics model to predict distributions of future states. Since Gaussian process models can be approximately sampled, a Thompson sampling approach could be employed for stochastic directed exploration. However, given PILCO's *probabilistic* dynamics model can compute full cumulative-cost *distributions* (approximated as Gaussian) of controllers analytically, *deterministic* directed exploration is possible. Deterministic exploration is preferred for data efficient learning in single agent tasks since the specific actions which optimise the expected value of information gained are not random! Unlike Deisenroth (2009) who compute the marginal cost distributions per time step, we compute the full joint covariance of all timestep's costs in an episode, required for computing the loss-variance (cumulative-cost variance). The cross terms in the joint cost covariance matrix typically contributes between 40% to 85% of the cumulative-cost variance, and thus must be included to avoid significantly underestimating the cumulative-cost variance. We use the additional cumulative-cost variance information for uncertainty-directed exploration. We evaluate the value of information by evaluating the uncertainty in the cumulative-cost function, as other myopic BRL algorithms do, to direct exploration, opposed to PILCO. As a second BRL extension of PILCO later in § 4.5, we discuss a better BRL extension to PILCO, also myopic, which considers how the dynamics model might change in response to future data we might see, giving us a more accurate estimate of the value of information.

4.2.2 PROBABLY APPROXIMATELY CORRECT (PAC) LEARNING

An alternate definition of 'data efficiency' as minimising cumulative regret (4.3) is instead minimising the number of episodes that the robot's expected performance fails to be within a specified tolerance of the optimal loss. Such algorithms are called PAC-MDP (Probably Approximately Correct Markov Decisions Process), discovering 'near-optimal' controllers with high-probability within time polynomial to the number of states and controls (Even-Dar et al., 2002). Example PAC-MDP algorithms include R-max (Brafman and Tennenholtz, 2003), E^3 (Kearns and Singh, 2002), and Delayed Q-Learning (Strehl et al., 2006). PAC-MDP methods provide powerful probabilistic-guarantees on the data-complexity required before asymptotic convergence to an optimal controller. However, such strong guarantees come at the price of over-exploration (Delage and Mannor, 2007; Kolter and Ng, 2009). PAC-MDP algorithms either systematically explore the complete state-space, or follow the principle of optimism under uncertainty (e.g. upper confidence bounds).

To be consistently *optimistic* under uncertainty is to overvalue exploration, and thus over-explore the state-space. For example, when priors over different transitions are artificially biased ('optimistic'), a controller tends to explore less visited states even when Bayes-optimal controllers under same prior would deem the cost of doing so too high. The PAC-MDP formulation disregards costs incurred in the short term by instead concentrating on discovery of near-optimal controllers in the long term. In doing so, PAC-MDP methods solve a different formulation of data efficiency, ensuring long-term controller near-optimality instead of maximising the expected cumulative costs. Indeed, Bayes-optimal controllers are *not* PAC-MDP (Kolter and Ng, 2009). As such, the PAC-MDP formulation is undesirable when the system always incurs real-world cost to interact with (regardless of whether the robot is exploring or exploiting). Indeed many methods' authors are willing to trade the probabilistic guarantees that PAC-MDP methods provide for practical performance gains (Jung and Stone, 2010). Similarly, any other methods that direct exploration by reducing dynamics uncertainty also incur data inefficiencies. The robot should not waste time modelling aspects of the system that are unlikely to help it achieve its goal. Since the goal of RL is optimisation of the expected cumulative costs (or rewards), exploratory actions should seek to reduce uncertainty of the *objective* (cumulative-cost) only. By evaluating exploration according to the objective distribution, we evaluate exploration in the same 'units' as we evaluate exploitation, making the trade-off between the two more straightforward.

4.3 Bayesian Optimisation

By simulating our system using our probabilistic dynamics model, is it possible to compute the cumulative-cost *distribution* given data up until current episode e exclusive: $C_e^\Psi \sim \mathcal{N}\left(\mu_e^C, \Sigma_e^C\right)$. We have not yet discussed *how* to compute C^Ψ's distribution, which we leave for § 4.4, but for the moment assume C^Ψ's distribution is computable. In this section, we discuss how to direct exploration using C^Ψ's full distribution with Bayesian Optimisation (BO) methods. We change the controller evaluation step (see Algorithm 1, line 6). No longer do we optimise the mean cumulative cost μ_e^C as PILCO did, but instead optimise a function of the mean *and* variance: $BO(\mu_e^C, \Sigma_e^C)$.

Bayesian optimisation is the problem of optimising an unknown function, often expensive to evaluate and without gradient information, through frugal successive samples of data. Good introductions are Brochu et al. (2010), Shahriari et al.

(2016) and Snoek et al. (2012). This corresponds to our problem of optimising **J** by frugal sampling one set of controller parameters $\psi \in \Psi$ per episode for execution. Generally, BO balances exploration-exploitation by selecting a controller π^{ψ} with low cumulative-cost mean $\mu_e^{\mathcal{C}}$ (exploitation) and high cumulative-cost variance $\Sigma_e^{\mathcal{C}}$ (exploration). High-variance controllers are attractive options to execute, since they may reveal that controller π^{ψ} in fact has much lower average cumulative-cost by the next episode $\mu_{e+1}^{\mathcal{C}}$ than currently believed (under expectation) $\mu_e^{\mathcal{C}}$. The benefit is that surprisingly-good controllers can be exploited *repeatedly* in future episodes, whereas surprisingly-poor controllers need not be executed again. We experiment with four BO functions, a simple upper confidence bound (UCB) heuristic with a single parameter, and three heuristics without free parameters: probability of improvement (PI), expected improvement (EI), and Gittins index (GI).

As with PILCO, during the execution of a single episode, the controller does not change, since we assume controller optimisation is slow and episodes must execute online in real-time. Thus, we consider changes to controllers necessary for exploration to occur only between episodes (not between timesteps as well).

4.3.1 UPPER CONFIDENCE BOUND (UCB)

As a simple baseline for exploration, we consider perhaps the simplest balance of exploration and exploitation using the upper confidence bound:

$$UCB(\mathcal{C}_e^{\psi}) \quad = \quad \mu_e^{\mathcal{C}} - \beta \sigma_e^{\mathcal{C}}, \tag{4.4}$$

where we treat β as a free parameter which balances exploitation ($\mu_e^{\mathcal{C}}$) with exploration ($\sigma_e^{\mathcal{C}}$). Notice both terms $\mu_e^{\mathcal{C}}$ and $\sigma_e^{\mathcal{C}}$ have the same units, so β is a dimensionless quantity. For more in depth discussion on how to set β as a function of the number of episodes remaining see Auer (2003).

4.3.2 PROBABILITY OF IMPROVEMENT (PI)

Consider our robot has already executed several different controllers (i.e. different controller parameterisations in Ψ) so far. Let the lowest cumulative cost witnessed so far be $\mathcal{C}^* \sim \mathcal{N}\left(\mu_*^{\mathcal{C}}, \Sigma_*^{\mathcal{C}}\right)$. Normally $\mathcal{C}^*$ has no distribution in BO, however, we generalise to handle cases where $\mathcal{C}^*$ is noisily observed. The Probability of Improvement (PI) exploration heuristic decides that the next controller to execute π^{ψ} should be the one that maximises the probability of improving its cumulative cost $\mathcal{C}_e^{\psi}$ over the ref-

erence $\mathcal{C}^*$. Thus, we search over parameterisations $\psi \in \Psi$, evaluating each controller using our simulator to predict their uncertain cumulative costs: $\mathcal{C}_e^\psi \sim \mathcal{N}\left(\mu_e^\mathcal{C}, \Sigma_e^\mathcal{C}\right)$, noting $\Sigma_e^\mathcal{C}$ is scalar. We can compute the probability that executing controller π^ψ improves $\mathcal{C}^*$:

$$\begin{aligned} PI(\mathcal{C}_e^\psi) &\doteq -p\left(\mathcal{C}_e^\psi < \mathcal{C}^*\right) \\ &= \Phi\left(\frac{\mu_e^\mathcal{C} - \mu_*^\mathcal{C}}{\sqrt{\Sigma_e^\mathcal{C} + \Sigma_*^\mathcal{C}}}\right) - 1, \end{aligned} \tag{4.5}$$

where $\phi(\cdot)$ is the standard normal distribution function, and $\Phi(\cdot)$ be its cumulative distribution function. Since we *minimise* loss functions, for consistency sake, we will also write each BO objective as functions to be minimised. Thus, we write our PI objective in (4.5) as the *negative* probability of improvement, an objective we intend to minimise. For a more detailed working of (4.5), see Appendix C.1. So, instead of a greedy algorithm, which minimises the expected cumulative cost as PILCO does, we can change Algorithm 1, line 6, to instead evaluate controllers using the metric $PI(\mathcal{C}^\psi)$, which we optimise greedily. Doing so, at each episode we maximise the probability of finding a superior controller compared to the best controller we have thus seen so far. This tends to avoid the trap of re-selecting the same controller at every episode when in a mean-cost optima in Ψ-space, as can happen with PILCO.

4.3.3 EXPECTED IMPROVEMENT (EI)

Above we discussed how the PI exploration heuristic maxmimises the probability of improving a controller, however, PI does not consider by *how much* a controller may be improved by. A pathological case for PI is choosing between controller π^ψ which has a 99% chance of *insignificant* improvement, and controller $\pi^{\psi'}$ with 98% chance of *significant* improvement. PI would choose the former, since $99\% > 98\%$, even though the latter intuitively offers better exploratory outcomes under expectation. The Expected Improvement (EI) heuristic instead weights the probability of improvement by the magnitude of the improvement:

$$\begin{aligned} EI(\mathcal{C}_e^\psi) &\doteq \mathbb{E}_{\mathcal{C}_e^\psi}\left[\min(\mathcal{C}_e^\psi - \mathcal{C}^*, 0)\right] \\ &= \Phi\left(-\frac{\mu_e^\mathcal{C} - \mu_*^\mathcal{C}}{\sqrt{\Sigma_e^\mathcal{C} + \Sigma_*^\mathcal{C}}}\right)(\mu_e^\mathcal{C} - \mu_*^\mathcal{C}) - \phi\left(\frac{\mu_e^\mathcal{C} - \mu_*^\mathcal{C}}{\sqrt{\Sigma_e^\mathcal{C} + \Sigma_*^\mathcal{C}}}\right)\sqrt{\Sigma_e^\mathcal{C} + \Sigma_*^\mathcal{C}}, \end{aligned} \tag{4.6}$$

with derivation in Appendix C.2.

4.3.4 GITTINS INDEX (GI)

Gittins index (GI) by Gittins et al. (1989) is a Bayes-optimal solution to the cumulative-regret, infinite-horizon, independent bandits problem (Berry and Fristedt, 1985). 'Bandits' colloquially refer to 'slot machines' in casinos, due to their tendency to leave people without money. In a typical bandits problem, an agent is presented with multiple bandits, and at each timestep they choose one bandit to play. Each bandit returns a random cost, usually independent samples from a Bernoulli or Gaussian distribution whose parameters are unknown, but have some prior distribution. Each bandit's prior distribution is usually i.i.d. The goal is to sequentially select bandits in a way that minimises the cumulative costs. The trade off between exploration and exploitation exists here, as the agent chooses between bandits not tested before (exploration) against bandits previously played which tend to return low cost (exploitation). Good solutions maintain posterior distributions over each bandit's uncertain parameters to direct exploration. The Gittins index of a bandit is a real-number used to compare different bandits before one bandit is selected, a single number score which balances exploration and exploitation optimally w.r.t. the posterior belief associated with each bandit. The Bayes-optimal decision is to simply select the bandit of greatest Gittins index at each episode. Executing a bandit generates a stochastic cost, used to update the bandit's Gittins index, and so the procedure repeats. A critical result is a Gittins index of independent bandit i is a function of belief of bandit i only, independent of other bandits. Decoupling bandit evaluation to decide which bandit to select next is termed an 'index policy' and renders Bayes-optimal solutions exponentially more tractable than brute force tree search, whose number of branches is proportional to the number of bandits to choose from. Even so, bandits with Gaussian uncertainty over their latent loss are still difficult to compute exactly, since the branching factor is still proportional to the number of cost outcomes, infinite for Gaussian bandits and only two for Bernoulli bandits. PILCO requires index policies since the space of controller parameters Ψ to select is uncountably infinite. An approximation to GI is Interval Estimation (IE) by Kaelbling (1993), a 'frequentist version' of the Bayesian GI loosely speaking. IE selects bandit of least lower confidence bound, computed from the average cost and counts or previously selection each bandit has. Meuleau and Bourgine (1999) show the striking similarity between IE and GI's loss reduction (compared to a baseline of random bandit selection), although IE has fatter tails ($\propto n_i^{-1/2}$) compared to GI ($\propto n_i^{-1}$, where n_i is the number of times bandit i has been selected so far) so IE tends

to over-explore. IE is easily computed, yet relies on discrete states to count their visits, thus inapplicable for PILCO's continuous state-action tasks.

Both PI and EI are well-known exploration heuristics in the BO community. However, GI is perhaps not as widely known in the BO community. We choose to also use GI from the bandits literature since GI is the exact Bayes-optimal solution to the Bayesian Optimisation problem under assumption of uncorrelated costs between actions, opposed to other, more popular myopic heuristic acquisition functions: PI, EI, UCB, and entropy methods. In addition, because of GI's principled nature (GI is *not* a heuristic) theoretical results are readily available to quantify how much GI-exploration is expected to improve the loss in advance of executing a controller. Both PI and EI are myopic heuristics, only considering the effects of a controller one episode into the future. However, GI being a Bayes-optimal controller, is non-myopic, conducting a deep yet efficient lookahead over the multiple episodes remaining.

We must approximate our problem in three ways to allow for computationally-cheap yet exact Bayes-optimal solutions using Gittins index:

1. Since our horizon is finite we convert our undiscounted finite-horizon problem to being discounted and infinite-horizon (Gittins index is only valid for infinite horizons). We do so using an 'effective discounting'[2] of $\gamma = 1 - 1/\hat{E}$, where $\hat{E} = E - e + 1$ is the number of episodes remaining (not yet executed), e is the current episode number, and E is the finite horizon of total episodes to execute (past and future). Doing so, we are applying an exact solution to a similar problem (infinite horizon) to our own problem (finite horizon). This first approximation is potentially mild, although tends to overestimate the amount of loss reduction, seen later in § 4.4.3, leading to mild over-exploration.

2. We ignore correlations between controller evaluations, assuming $\mathbb{C}\left[\mathcal{C}_e^{\psi}, \mathcal{C}_e^{\psi'}\right] = 0$ if $\psi \neq \psi'$, for tractability reasons. This second assumption is almost always violated for PILCO's tasks in continuous state spaces. The ensuring suboptimality of the GI performance is dependent on the amount of correlation (Ryzhov et al., 2012), but given the infinite number of policies (and therefore infinite number of bandits), it is difficult to determine how suboptimal our approach is.

[2] I.e. receiving fixed cost λ over a finite horizon without discounting returns a total of $\hat{E}\lambda$. An infinite horizon with discounting γ returns $\lambda/(1-\gamma)$. Thus, for equal returns $\hat{E}\lambda = \lambda/(1-\gamma)$, then $\gamma = 1 - 1/\hat{E}$.

3. We assume there exists only one more opportunity to learn about $\mathcal{C}_e^\psi$. We assume after one execution of π^ψ, we will gain an observation y_e related to the true latent value of $\mathcal{C}_{\text{true}}^\psi$ in which to learn a posterior on $\mathcal{C}_{e+1}^\psi$, at which point learning ceases. We assume we do not learn anything thereafter about $\mathcal{C}_{\text{true}}^\psi$, either by executing π^ψ a second time or another controller. Depending on whether we underestimate or overestimate how much the posterior variance in $\mathcal{C}_{e+1}^\psi$ reduces, we might under-explore or over-explore. This third assumption is our most severe assumption, being unclear how much it causes the robot to under (or over) explore.

Under the above three assumptions (or approximations), we treat each prior $\mathcal{C}_e^\psi \sim \mathcal{N}\left(\mu_e^\mathcal{C}, \Sigma_e^\mathcal{C}\right)$ for $\psi \in \Psi$ as a Gaussian bandit, and to each bandit $\mathcal{C}_e^\psi$ we compare with a deterministic bandit $\mathcal{C}^\lambda$, which offers constant cost λ per timestep. Gittins index is the value of λ that makes us ambivalent between choosing to execute the uncertain bandit $\mathcal{C}_e^\psi$ (which has exploratory benefits) or receiving a deterministic cost λ (Gittins et al., 1989) (pure exploitation). For shorthand sake during this section, let $\sigma_e^2 \doteq (\sigma_e^\mathcal{C})^2 \doteq \Sigma_e^\mathcal{C}$.

Now let us analyse the exploratory benefits of choosing uncertain bandit $\mathcal{C}_e^\psi$. We begin by studying the effect that a cost observation y_e would have on the robot's posterior $\mathcal{C}^\psi$. On receiving a noisy observation related to the latent $\mathcal{C}_{\text{true}}^\psi$,

$$y_e \overset{iid}{\sim} \mathcal{N}\left(\mathcal{C}_{\text{true}}^\psi, \sigma_y^2\right), \tag{4.7}$$

both our prior $p(\mathcal{C}_e^\psi) = \mathcal{N}\left(\mu_e^\mathcal{C}, \sigma_e^2\right)$ and the observational likelihood from y_e at episode e would combine to form our posterior for the following episode $e+1$:

$$\mathcal{C}_{e+1}^\psi \sim \mathcal{N}\left(\mu_{e+1}^\mathcal{C}, \sigma_{e+1}^2\right), \text{ where} \tag{4.8}$$

$$\sigma_{e+1}^2 = (\sigma_e^{-2} + \sigma_y^{-2})^{-1}, \tag{4.9}$$

$$\mu_{e+1}^\mathcal{C} = \sigma_{e+1}^2(\sigma_e^{-2}\mu_e^\mathcal{C} + \sigma_y^{-2} y_e). \tag{4.10}$$

Consider we make the pessimistic assumption that the observation y_e will be the last chance the robot has to learn about the latent $\mathcal{C}_{\text{true}}^\psi$. Such a pessimistic assumption is akin to expecting bandit ψ will only payout an immediate cost sample from $\mathcal{C}_e^\psi$ and then payout according to the next-episode expected cost $\mu_{e+1}^\mathcal{C}$ for each episode thereafter. As we consider bandit ψ (before executing it), the observation y_e is not yet observed, and is thus a random variable, denoted Y_e. Using our prior information,

we can anticipate a plausible set of observations:

$$Y_e \sim \mathcal{N}\left(\mathcal{C}_e^{\psi}, \sigma_y^2\right) = \mathcal{N}\left(\mu_e^{\mathcal{C}}, \sigma_e^2 + \sigma_y^2\right). \tag{4.11}$$

Because Y_e is random, so too must the future posterior-mean at episode $e+1$ be:

$$\begin{aligned}
\mu_{e+1}^{\mathcal{C}} &= \sigma_{e+1}^2(\sigma_e^{-2}\mu_e^{\mathcal{C}} + \sigma_y^{-2}Y_e) \\
&\sim \mathcal{N}\left(\sigma_{e+1}^2(\sigma_e^{-2}\mu_e^{\mathcal{C}} + \sigma_y^{-2}\mu_e^{\mathcal{C}}),\ \sigma_{e+1}^4\sigma_y^{-4}(\sigma_e^2 + \sigma_y^2)\right) \\
&= \mathcal{N}\left(\mu_e^{\mathcal{C}}, s^2\right), \text{ where} \qquad (4.12) \\
s^2 &\doteq \tfrac{\sigma_e^4}{(\sigma_e^2+\sigma_y^2)}. \qquad (4.13)
\end{aligned}$$

Now, since the Gittins index is the value of λ, which makes the robot ambivalent between choosing a fixed cost λ or a stochastic cost from our beliefs, we equate the expected cumulative cost of both the deterministic bandit $\mathcal{C}^{\lambda}$ (left) with the stochastic bandit $\mathcal{C}^{\psi}$ (via $\mu_{e+1}^{\mathcal{C}}$) (right) using identities from geometric series:

$$\begin{aligned}
\frac{\lambda}{1-\gamma} &= \mathbb{E}_{\mathcal{C}_e^{\psi}}[\mathcal{C}_e^{\psi}] + \frac{\gamma}{1-\gamma}\cdot\mathbb{E}_{\mu_{e+1}^{\mathcal{C}}}\left[\min(\mu_{e+1}^{\mathcal{C}}, \lambda)\right] \\
&= \mu_e^{\mathcal{C}} + \frac{\gamma}{1-\gamma}\cdot\Big(\int_{-\infty}^{\lambda}\mu_{e+1}^{\mathcal{C}}\cdot\mathcal{N}\left(\mu_{e+1}^{\mathcal{C}}; \mu_e^{\mathcal{C}}, s^2\right)\mathrm{d}\mu_{e+1}^{\mathcal{C}} \\
&\qquad + \int_{\lambda}^{\infty}\lambda\cdot\mathcal{N}\left(\mu_{e+1}^{\mathcal{C}}; \mu_e^{\mathcal{C}}, s^2\right)\mathrm{d}\mu_{e+1}^{\mathcal{C}}\Big) \\
&= \mu_e^{\mathcal{C}} + \frac{\gamma}{1-\gamma}\cdot\left(-s\phi(\tfrac{\lambda-\mu_e}{s}) + \mu_e^{\mathcal{C}}\Phi(\tfrac{\lambda-\mu_e}{s}) + \lambda\left(1-\Phi(\tfrac{\lambda-\mu_e}{s})\right)\right). \qquad (4.14) \\
\therefore \lambda' &\doteq \frac{\lambda-\mu_e^{\mathcal{C}}}{s} = -\frac{\gamma}{1-\gamma}\cdot\big(\phi(\lambda') + \lambda'\Phi(\lambda')\big), \qquad (4.15)
\end{aligned}$$

using Gaussian integral identities from Appendix A.2.8. We can solve for λ', which has an unique solution, by rearranging (4.15) and using Newton's method:

$$\begin{aligned}
g(\lambda') &\doteq \gamma\big(\phi(\lambda') + \lambda'\Phi(\lambda') - \lambda'\big) + \lambda' = 0, \qquad (4.16) \\
\frac{\mathrm{d}g(\lambda')}{\mathrm{d}\lambda'} &= \gamma\big(-\lambda'\phi(\lambda') + \Phi(\lambda') + \lambda'\phi(\lambda') - 1\big) + 1 \\
&= \gamma\big(\Phi(\lambda') - 1\big) + 1, \qquad (4.17) \\
\text{find } \lambda' &: g(\lambda') = 0, \\
\lambda_0' &\leftarrow 0, \qquad (4.18) \\
\lambda_{i+1}' &\leftarrow \lambda_i' - g(\lambda_i')\left(\frac{\mathrm{d}g(\lambda_i')}{\mathrm{d}\lambda_i'}\right)^{-1}. \qquad (4.19)
\end{aligned}$$

Table 4.1 ***Example values of the upper-bounded Gittins index of a standard Gaussian bandit:*** $GI^{upper}(\mu_e^{\mathcal{C}}=0,\ \sigma_e^{\mathcal{C}}=1,\ \sigma_y,\ \gamma=1-1/\hat{E})$***.*** *Each element represents a conservative estimate of the reduction in average-loss the robot can expect (averaged over the remaining episodes* $\hat{E}$*) if it follows the exploratory behaviour directed by* GI^{upper} *in* (4.20)*.*

$\hat{E}$ \ σ_y^2	0	0.1	1	10	∞
1	0	0	0	0	0
2	-0.2760	-0.2632	-0.1952	-0.0832	0
3	-0.4363	-0.4160	-0.3085	-0.1316	0
4	-0.5492	-0.5236	-0.3883	-0.1656	0
5	-0.6360	-0.6064	-0.4497	-0.1918	0
6	-0.7065	-0.6736	-0.4996	-0.2130	0
7	-0.7658	-0.7301	-0.5415	-0.2309	0
8	-0.8168	-0.7788	-0.5776	-0.2463	0
9	-0.8616	-0.8215	-0.6092	-0.2598	0
10	-0.9015	-0.8595	-0.6374	-0.2718	0

Thus, our approximate Gittins index of controller π^{ψ} (substituting for s^2 (4.13)) is:

$$GI^{\text{upper}}(\mu_e^{\mathcal{C}},\sigma_e^{\mathcal{C}},\sigma_y,\gamma)\ =\ \lambda\ =\ \mu_e^{\mathcal{C}}+\lambda'_{\gamma}\frac{\sigma_e^2}{\sqrt{\sigma_e^2+\sigma_y^2}},\qquad\text{where }\ \gamma\ =\ 1-1/\hat{E}. \tag{4.20}$$

Notice the converged value of λ' from (4.19) (now denoted λ'_{γ}) is independent of any one controller's parameters ψ (i.e. not a function of $\mu_e^{\mathcal{C}}$ nor σ_e), only the discounting factor γ (or finite horizon $\hat{E}$). Therefore, λ'_{γ} is a constant during PILCO's controller optimisation phase (Algorithm 1, line 7), meaning we do not have to recompute λ'_{γ} when comparing different controllers in a different space, of which may be difficult to compute gradient information given that λ'_{γ} has no closed form analytical expression (we resorted to Newton's method). Thanks to this independence, we can compute the Gittins index of each controller as a continuous function over controller-parameter space Ψ. Additionally, given the availability of cumulative-cost gradients $\mathrm{d}\mu_e^{\mathcal{C}}/\mathrm{d}\psi$ and $\mathrm{d}\sigma_e^2/\mathrm{d}\psi$, gradient-based optimisation using GI is possible, since the gradient of the Gittins index λ from (4.20) is simply $\mathrm{d}\lambda/\mathrm{d}\psi=\mathrm{d}\mu_e^{\mathcal{C}}/\mathrm{d}\psi+\lambda'_{\gamma}\mathrm{d}s/\mathrm{d}\psi$, where $\mathrm{d}s/\mathrm{d}\psi=(\sigma_e^2/2+\sigma_y^2)(\sigma_e^2+\sigma_y^2)^{-3/2}\mathrm{d}\sigma_e^2/\mathrm{d}\psi$.

Examples values of GI^{upper} are shown Table 4.1. As a reminder, $\hat{E}$ is the number of episodes remaining (including current episode e), and σ_y^2 is noise associated with observing C_{true}^{ψ} after one episode of executing controller π^{ψ}. Two important trends are immediately observed in Table 4.1, where table elements reflect the expected reduction in average-loss if they explore in a way directed by this GI bound. First,

moving top-to-bottom along Table 4.1, the index values reduce for greater number of episodes remaining $\hat{E}$. This reflects that for more episodes remaining for a robot to trial a task, then there exist more future opportunities from which to (re-)capitalise from information learned in the next episode. I.e. the value of information increases. Note the first row, where one trial remains $\hat{E} = 1$, the robot will not be able to capitalise on any information gained (since this is the last attempt at controlling a task), and so the value of exploration is (correctly) deemed worthless, with a row of zeros. Second, moving left-to-right along Table 4.1, note the worsening of the value of information (increasing loss) as the observation noise-variance σ_y^2 increases. The left represent noiseless observations, giving us maximal information, whilst the right side represents so much noise that exploration is also (correctly) deemed worthless. Without an expectation to gain information on the right side, the number of episodes remaining $\hat{E}$ is irrelevant, and the robot anticipates no possibility in reducing loss via exploration, hence a column of zeros.

We have now discussed an upper bound of the intractable GI^{true}, an *upper* bound (worse cost) because of its pessimistic assumption about an ability to collect information (only anticipating one more observation y_e to learn from, assuming nothing can be learned thereafter). Now, we discuss a lower bound of GI^{true} based on the opposite: an *optimistic* assumption on our ability to collect information. An optimistic assumption is to assume the next observation will be noiseless $\sigma_y = 0$ (the left column of Table 4.1), and thus everything about $\mathcal{C}^{\Psi}$ will be learned by episode $e+1$ with certainty. By setting noise $\sigma_y = 0$ in (4.20) we get an lower bound:

$$GI^{\text{lower}}(\mu_e^{\mathcal{C}}, \sigma_e^{\mathcal{C}}, \sigma_y, \gamma) \;=\; \lambda \;=\; \mu_e^{\mathcal{C}} + \lambda'_{\gamma}\sigma_e^{\mathcal{C}}, \qquad \text{where } \gamma \;=\; 1 - 1/\hat{E}. \tag{4.21}$$

where our uncertainty by next episode is $\mathcal{C}_{e+1}^{\Psi} \sim \mathcal{N}\left(\mu_{e+1}^{\mathcal{C}}, 0\right)$ with $\mu_{e+1}^{\mathcal{C}} \sim \mathcal{N}\left(\mu_e^{\mathcal{C}}, \sigma_e^2\right)$ from our current perspective at episode e. This myopic approximation that will optimistically 'learn everything' about $\mathcal{C}^{\Psi}$ is similar to the 'knowledge gradient' approximate solution introduced by Powell and Ryzhov (2012, section 6.4). However, the knowledge gradient is a non-index policy (the evaluation of one controller depends on other controllers) which is only suited to finitely-many bandits. Note that if σ_y is equal to zero, then $GI^{\text{lower}} = GI^{\text{true}}$ for all values γ. This lower bound is also well known in dynamic programming, referred to as the Q_{MDP} solution, which approximates the *convex* value function of a POMDP as instead *linear* over belief space (Cassandra and Kaelbling, 1995). The consequence would be underestimation of the cost-to-go, leading to a non-optimal balance of over-exploration and under-exploitation.

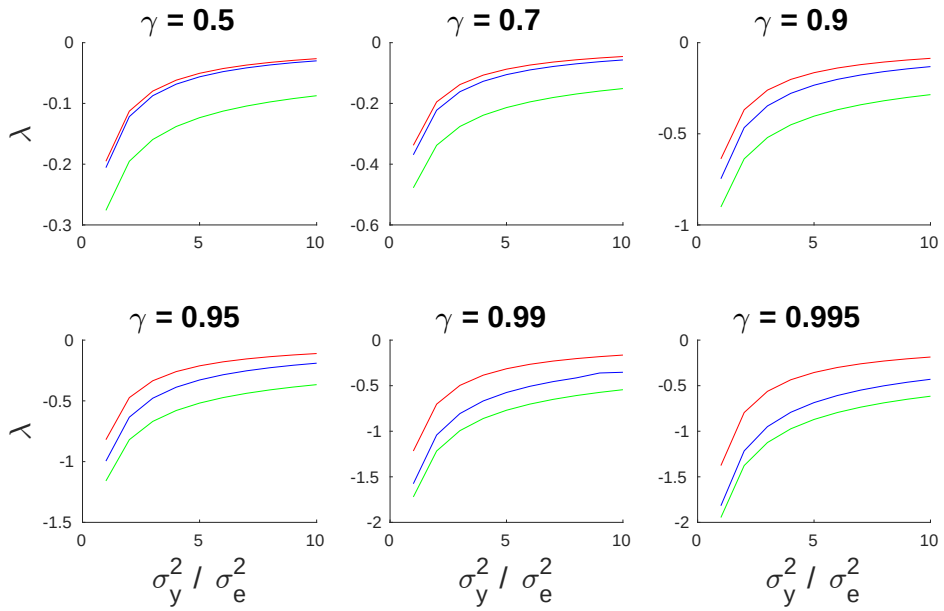


Fig. 4.1 ***Gittins index bounds.*** *Each blue line is the exact GI for infinite horizon for various discounting factors γ, each upper and lower bounded. Red shows the GI upper bound* (4.21) *resulting from pessimistic assumptions. Green shows the GI lower bound* (4.20) *resulting from optimistic assumptions. Values of the true GI are from Powell and Ryzhov (2012, table 6.3), and $\sigma_y = 1$.*

Both the upper and lower bounds of the true intractable GI^{true} are compared in Fig. 4.1, to visualise the quality of these bounds. The pessimistic upper bound (red) is a good approximation under severe discounting $\gamma = 0.5$, though not under light discounting of $\gamma = 0.995$. The reason is, our pessimistic approximation only considers short term information gain, which under severe discounting is accurate because only short term costs dominate the loss function. On the other hand, our optimistic lower bound (green) does the opposite: improving the larger γ is. Although we can see the green's accuracy additionally depends on the observation noise σ_y^2, especially accurate when the observation noise σ_y^2 is comparable with (or less than) the prior variance on C_e^{ψ} (becoming exact when $\sigma_y^2 = 0$). Being a good approximation under low observation noise should come as no surprise, since the approximation was to assume no observation noise. However, the improvement with larger γ (under fixed σ_y^2) can perhaps be explained by the fact that the uncertainty will indeed collapse *eventually* (given enough re-selections of the same bandit), which the loss function will consider if long term costs are not dominated by short terms costs, which they are not under light discounting of $\gamma = 0.995$.

An advantage of the lower bound instead of the upper bound GI is we do not need to know the actual value of y_e, we simply assumed y_e was zero. For this reason

we choose to use GI^{lower} in our experiments discussed later, and will abbreviate GI^{lower} as simply GI. Given the conditions under which GI^{lower} is accurate, we only advocate its usage for tasks with relatively small σ_y^2 and large $\hat{E}$ (γ close to one), ensuring GI^{lower} remains a tight bound to GI^{true}.

4.4 Directing Exploration with Cumulative-Cost Uncertainty

PILCO is a pure exploitation RL algorithm by optimising the cumulative-cost mean. Whilst PILCO does not explicitly conduct exploration, some 'indirect exploration' effects occur due to 1) inherent system stochasticity, 2) a tendency for the saturating cost functions to give lower expected costs for uncertain states when not near the goal (Deisenroth et al., 2015), and 3) the information from executing ones controller informs the performance for *all* controllers. PILCO's success was largely due a principled treatment of dynamics uncertainty using a probabilistic dynamics model to optimise controllers according to their expected cumulative cost, and the indirect exploration effects from a saturating cost function.

Our contribution in this chapter is to improve upon PILCO's existing data efficiency by using PILCO's existing probabilistic model to additionally compute the cumulative-cost *variance* to direct exploration towards testing controllers of uncertain cumulative-cost. We balance exploitation-vs-exploration by executing controllers of low mean cumulative-cost $\mu_e^{\mathcal{C}}$ (exploitation) and high variance $\Sigma_e^{\mathcal{C}}$ (exploration). As discussed previously (§ 4.3), we can strike a balance between the two using Bayesian Optimisation (BO), executing the controller of least heuristic output $BO(\mu_e^{\mathcal{C}}, \Sigma_e^{\mathcal{C}})$. The computation of $\mu_e^{\mathcal{C}}$ is straightforward, being the accumulation of mean-costs at each timestep, detailed in Deisenroth and Rasmussen (2011). However, PILCO did not compute $\Sigma_e^{\mathcal{C}}$. By doing so, we show we can achieve even greater data efficiency than the unprecedented performance of PILCO.

4.4.1 The Cumulative-Cost Distribution

We thus now outline computation of the cumulative-cost distribution:

$$\mathcal{C}_e^{\psi} \doteq \sum_{t=0}^{T} \text{cost}(X_t)|\psi \sim \mathcal{N}\left(\mu_e^{\mathcal{C}}, \Sigma_e^{\mathcal{C}}\right), \qquad \text{where} \tag{4.22}$$

$$\mu_e^{\mathcal{C}} = \sum_{t=0}^{T} \mathbb{E}_X\left[\text{cost}(X_t)|\psi\right], \tag{4.23}$$

$$\Sigma_e^{\mathcal{C}} = \sum_{t=0}^{T}\sum_{t'=0}^{T} \mathbb{C}_X\left[\text{cost}(X_t), \text{cost}(X_{t'})|\psi\right]. \tag{4.24}$$

The distribution of $\mathcal{C}_e^{\psi}$ is a function of the joint predictive state $p(X_0, ..., X_T)$, approximated as Gaussian. The original PILCO computed the marginal cost mean and variance for each timestep only. But here, each cost-cost correlation is a function of a corresponding state-state correlation $\mathbb{C}_X\left[X_t, X_{t'}|\psi\right]$, which we must now compute.

State-State Joint Distribution

PILCO already computes the expected states at each timestep, $\mathbb{E}_X\left[X_0\right], ..., \mathbb{E}_X\left[X_T\right]$, and each marginal variance $\mathbb{V}_X\left[X_0\right], ..., \mathbb{V}_X\left[X_T\right]$ (Deisenroth and Rasmussen, 2011). Computing the full state-state *joint* distribution (approximated as Gaussian) additionally requires covariances between arbitrary states, e.g. $\mathbb{C}_X\left[X_4, X_9\right]$. Before discussing covariance between arbitrary states, let's begin with successive states, X_t and X_{t+1}. First we define the linearisations that give rise to the moment-matched distributions of the control U_t and next state X_{t+1} given the current state X_t:

$$\begin{aligned}
C_t^{xy} &\doteq (\Sigma_t^x)^{-1}\mathbb{C}\left[X_t, Y_t\right] &&= I, \\
C_t^{xu} &\doteq (\Sigma_t^x)^{-1}\mathbb{C}\left[X_t, U_t\right] &&= C_t^{xy}C_t^{yu}, \\
C_t^{x\tilde{x}} &\doteq (\Sigma_t^x)^{-1}\mathbb{C}\left[X_t, \tilde{X}_t\right] &&= [I, C_t^{xu}], \\
C_t^{xx'} &\doteq C_t^{x\tilde{x}}C_t^{\tilde{x}x'} &&\approx (\Sigma_t^x)^{-1}\mathbb{C}\left[X_t, X_{t+1}\right],
\end{aligned} \tag{4.25}$$

where I is a $\mathrm{X} \times \mathrm{X}$ identity matrix, Y_t is a noisy observation of X_t, and $C_t^{\tilde{x}x'} \doteq (\Sigma_t^{\tilde{x}})^{-1}\mathbb{C}_{\tilde{X}}\left[\tilde{X}_t, X_{t+1}\right]$ is outputted by PILCO's input-output covariance from its GP dynamics model. Note the approximate equality in (4.25) is because between X_t and X_{t+1} there exist multiple nonlinear functions (first a controller function, second a dynamics model prediction). Even though PILCO's moment-matching is exact between individual nonlinear mappings, the moments are not exact through multiple sequential nonlinear mappings (see Appendix A.2.6). Continuing to use Appendix A.2.6,

we can approximately compute the covariance between successive states:

$$\mathbb{C}\left[X_t,\, X_{t+2}\right] \;\approx\; \mathbb{C}\left[X_t,\, X_{t+1}\right](\Sigma^x_{t+1})^{-1}\mathbb{C}\left[X_{t+1},\, X_{t+2}\right] \;=\; \Sigma^x_t C^{xx'}_t C^{xx'}_{t+1}. \tag{4.26}$$

And indeed any state-state covariance as a product (which is exact if using linear controllers and dynamics functions):

$$\mathbb{C}\left[X_t,\, X_{t+\tau}\right] \;\approx\; \Sigma^x_t \prod_{k=0}^{\tau-1} C^{xx'}_{t+k} \;=\; \Sigma^x_t C^{xx'}_t ... C^{xx'}_{t+\tau-1}, \qquad \forall\, \tau > t. \tag{4.27}$$

Using the above expression for state-state covariances we can now compute each and every cost-cost covariance required by (4.24), detailed next.

Cost-Cost Joint Distribution

We use the following saturating cost function of system state X, parameterised by goal state x^* and length scale matrix Λ_c, defined:

$$\text{cost}(X; x^*, \Lambda_c) \;\doteq\; 1 - \exp\left(-\tfrac{1}{2}(X - x^*)^\intercal \Lambda_c^{-1}(X - x^*)\right). \tag{4.28}$$

Here we define the joint distribution of the output of two costs functions:

$$p\left(\begin{bmatrix}\text{cost}(X_1; x^*, \Lambda_c)\\ \text{cost}(X_2; x^*, \Lambda_c)\end{bmatrix}\right) \;=\; \mathcal{N}\left(\begin{bmatrix}m^c_1\\ m^c_2\end{bmatrix}, \begin{bmatrix}V^c_1 & V^c_{12}\\ V^c_{21} & V^c_2\end{bmatrix}\right), \tag{4.29}$$

given the joint distribution of both input systems states:

$$p\left(\begin{bmatrix}X_1\\ X_2\end{bmatrix}\right) \;=\; \mathcal{N}\left(\begin{bmatrix}\mu^x_1\\ \mu^x_2\end{bmatrix}, \begin{bmatrix}\Sigma^x_1 & \Sigma^x_{12}\\ \Sigma^x_{21} & \Sigma^x_2\end{bmatrix}\right). \tag{4.30}$$

Previous work by Deisenroth and Rasmussen (2011) provides moments for *marginal* cost outputs, $i \in \{1, 2\}$:

$$\begin{aligned} m^c_i \;&\doteq\; \mathbb{E}_{X_i}\left[\text{cost}(X_i; x^*, \Lambda_c)\right] \\ &=\; 1 - \det(I + \Sigma^x_i \Lambda_c^{-1})^{-\frac{1}{2}} \\ &\quad\; \times \exp\Big(-\tfrac{1}{2}(\mu^x_i - x^*)^\intercal \Lambda_c^{-1}(I + \Sigma^x_i \Lambda_c^{-1})^{-1}(\mu^x_i - x^*)\Big), \end{aligned} \tag{4.31}$$

$$\begin{aligned} V^c_i \;&\doteq\; \mathbb{V}_{X_i}\left[\text{cost}(X_i; x^*, \Lambda_c)\right] \\ &=\; -(m^c_i - 1)^2 + \det(I + 2\Sigma^x_i \Lambda_c^{-1})^{-\frac{1}{2}} \\ &\quad\; \times \exp\Big(-(\mu^x_i - x^*)^\intercal \Lambda_c^{-1}(I + 2\Sigma^x_i \Lambda_c^{-1})^{-1}(\mu^x_i - x^*)\Big). \end{aligned} \tag{4.32}$$

Thus the only remaining term we must derive is V_{12}^c. Let us begin by defining

$$e_i \doteq -\tfrac{1}{2}(X_i - x^*)^\intercal \Lambda_c^{-1}(X_i - x^*), \tag{4.33}$$

noting

$$e_1 + e_2 = -\tfrac{1}{2}\begin{bmatrix} X_1 - x^* \\ X_2 - x^* \end{bmatrix}^\intercal \begin{bmatrix} \Lambda_c^{-1} & 0 \\ 0 & \Lambda_c^{-1} \end{bmatrix} \begin{bmatrix} X_1 - x^* \\ X_2 - x^* \end{bmatrix}. \tag{4.34}$$

Now we can compute V_{12}^c:

$$\begin{aligned} V_{12}^c &\doteq \mathbb{C}_X\left[\operatorname{cost}(X_1; x^*, \Lambda_c),\ \operatorname{cost}(X_2; x^*, \Lambda_c)\right] \\ &= \mathbb{E}_X\left[\operatorname{cost}(X_1; x^*, \Lambda_c) \cdot \operatorname{cost}(X_2; x^*, \Lambda_c)\right] - m_1^c m_2^c \\ &= \mathbb{E}_X\left[\big(1 - \exp(e_1)\big)\big(1 - \exp(e_2)\big)\right] - m_1^c m_2^c \\ &= \mathbb{E}_X\left[\exp(e_1 + e_2)\right] - (1 - m_1^c)(1 - m_2^c) \\ &= (1 - m^c) - (1 - m_1^c)(1 - m_2^c), \end{aligned} \tag{4.35}$$

where $V_{21}^c = (V_{12}^c)^\intercal$ and m^c is simply the expected cost of the concatenated input state (distributed by (4.30)) and augmented parameters:

$$m^c = \mathbb{E}_X\left[\operatorname{cost}\left(\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}; \begin{bmatrix} x^* \\ x^* \end{bmatrix}, \begin{bmatrix} \Lambda_c & 0 \\ 0 & \Lambda_c \end{bmatrix}\right)\right], \tag{4.36}$$

computed as per (4.31). This concludes how to compute the joint distribution of any cost-cost pair of outputs given the joint distribution of the state-state pair of inputs.

4.4.2 Experiments

We compare PILCO against the four aforementioned Bayesian optimisation exploration strategies using the cart double-pole swing-up task (Fig. 4.2). PILCO optimises the expected cumulative-cost $\mu_e^{\mathcal{C}}$ only, whereas the algorithms using UCB, PI, EI, and GI use the full distribution $\mu_e^{\mathcal{C}}$ and $\Sigma_e^{\mathcal{C}}$.

Looking at Fig. 4.2, the state of the system is $x = [\dot{x}_1, \dot{\theta}_2, \dot{\theta}_3, x_1, \theta_2, \theta_3]$, consisting of the horizontal position of the cart x_1, the two angles θ_2 and θ_3 (both measured anti-clockwise from vertically up) and the time derivatives of these three variables, $\dot{x}_1$, $\dot{\theta}_2$ and $\dot{\theta}_3$. To begin the task, the initial state is both poles hanging downwards: $x_{t=0} = [0, 0, 0, 0, \pi, \pi]$. To control the system, we define a controller's functional form as a mixture of 200 radial basis functions. The cart can move horizontally, with an applied external force $-20\text{N} \leq u \leq 20\text{N}$, and coefficient of

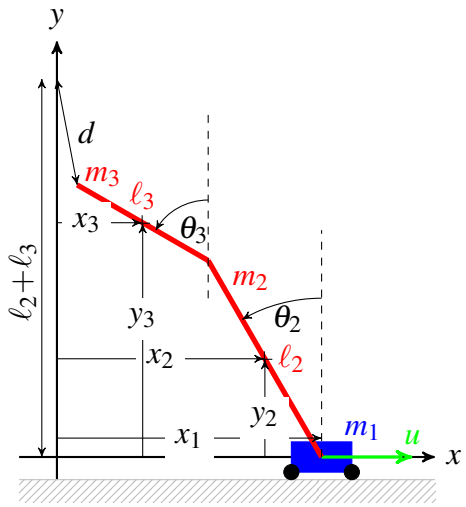


Fig. 4.2 ***The cart-double-pole swing-up task.*** *The cart and double pendulum dynamical system consists of a cart with mass m_1 and an attached double pendulum. The double pendulum consists of two pieces, the first of mass m_2 and length ℓ_2 and the second has mass m_3 and length ℓ_3. Both joints are frictionless and unactuated, the double pendulum can move freely in the vertical plane.*

friction $b = 0.1\text{Ns/m}$. The other physical values are $m_1 = m_2 = m_3 = 0.5\text{kg}$ and $\ell_2 = \ell_3 = 0.6\text{m}$. Thus, the moment of inertia around the midpoint of each part of the pendulum is $I_2 = I_3 = 0.015\text{kg}\cdot\text{m}^2$. The observation noise standard deviation is: $\text{diag}\big((\Sigma_y^\epsilon)^{\frac{1}{2}}\big) = [1\text{mm}/\Delta t, 1^\circ/10/\Delta t, 1^\circ/10/\Delta t, 1\text{mm}, 1^\circ/10, 1^\circ/10]$, where time is discretised as $\Delta t = 0.05\text{s}$. Each episode is time horizon of 1.5s (i.e. $T = 30$ timesteps), with a total number of $E = 20$ episodes per experiment. All parameter values are summarised Table D.3.

The cart double-pole system is a nonlinear dynamical system of six state dimensions, generally difficult to control in closed loop due to its complexity (a chaotic system), it's degree of underactuation, and the sensitivity of the stability of the system w.r.t. θ_3 in Fig. 4.2. As a well-known chaotic system, predicting long-term behaviour of the cart double-pole accurately in open-loop is very difficult. The system is also underactuated, with only a scalar control output to apply a bounded force u horizontally to the cart in which to control *six* state variables.

The task goal is to swing-up and then balance the double pendulum vertically above $x_1 = 0$, and the discrepancy is the distance d. We use the same saturation cost function as PILCO: $1 - \exp(-\frac{1}{2}d^2/\lambda_c^2)$ where $\lambda_c = 1.0\text{m}$ and d^2 is the squared Euclidean distance between the pendulum's end point and its goal $(0, l_2 + l_3)$. We execute 10 experiments per algorithm (with different random seeds). During each experiment, we evaluate the controller at each episode by averaging over 10 rollouts.

4.4.3 Results and Analysis

We initially test each BO heuristic using a system of standard Gaussian bandits, The main reason is to verify that the first approximation made in the GI heuristic discussed § 4.3.4, of a finite horizon approximated as infinite, is not too crude. Then, we move onto the more complicated cart double-pole swing-up task.

Gaussian Bandits Task

Consider a countably infinite number independent standard Gaussian bandits the robot can choose from. The i'th bandit returns a constant cost $\mathcal{C}^i$, where $\mathcal{C}^i$ is uncertain *a priori*. The robot's subjective prior on each cost $\mathcal{C}^i \;\; \forall\, i \in \mathbb{N}$ is independent and identically distributed: $\mathcal{C}^i \overset{iid}{\sim} \mathcal{N}(0,1)$. Assume the observation noise $\sigma_y = 0$ such that the uncertainty on $\mathcal{C}^i$ will immediately collapse after a single selection. Since our experimental setup with the cart double-pole will use 1 random rollout (Algorithm 1, line 2), followed by $E = 20$ controlled rollouts, totalling $\hat{E} = 21$ rollouts, we will use the same setting for our Gaussian bandits task. So, the robot only has $\hat{E} = 21$ iterations in which to select a new or previously played bandit i before the game is over. The goal, as usual, is to minimise the cumulative cost across $\hat{E}$-many bandit selections.

Results in Table 4.2 show all four algorithms, including PILCO which optimises the mean cumulative-cost only $BO(\mathcal{C}_e^{\Psi}) = \mathbb{E}\left[\mathcal{C}_e^{\Psi}\right] = \mu_e^{\mathcal{C}}$. Both the probability of improvement (PI) and expected improvement (EI) perform better than PILCO which had no explicit exploration strategy, whilst the approximate Gittins index (GI) obtains the best cumulative-cost averaged over the $\hat{E}$ episodes.

Table 4.2 ***Performance of each algorithm, a pure exploitation learner and four BO algorithms on standard Gaussian bandits.*** *The experiment lasts $\hat{E} = 21$ episodes. Costs are averaged over the $\hat{E}$ episodes and again averaged over 10^5 repeats of the experiment from scratch. Accuracy of the empirical results in the first row is approximately 10^{-3}, which is tends to zero in the infinite limit of experiments.*

$BO(\mathcal{C}_e^{\Psi})$ param.	random	PILCO	UCB $\beta=0.1$	UCB $\beta=0.3$	UCB $\beta=1.0$	PI	EI	GI
$\mathbb{E}\left[\frac{\mathbf{J}^{\text{empirical}}}{\hat{E}}\right]$	0.0006	-0.723	-0.775	-0.876	-1.081	-0.905	-0.864	**-1.093**
$\mathbb{E}\left[\frac{\mathbf{J}^{\text{predicted}}}{\hat{E}}\right]$	0	-0.722	-0.773	-0.874	-1.078		-0.564	**-1.177**

An advantage of some of these BO algorithms, is an ability to predict in advance how much each exploration strategy is expected to reduce the loss. For our baseline, random selection of bandits, we predict zero loss reduction, since each bandits' latent cumulative-cost is sampled form a standard Gaussian of zero mean. The UCB heuristics use the mean cumulative cost $\mu_e^{\mathcal{C}}$ and an additive $\beta\sigma_e^{\mathcal{C}}$ terms. We can predict the performance of PILCO using geometric series. Note, PILCO can be seen as a UCB heuristic also, with $\beta = 0$. To predict the loss, we first sum up the possible sequence of events, weighted by their probabilities, noting that by following UCB (optimising $\mu_e^{\mathcal{C}} - \beta\sigma_e^{\mathcal{C}}$, the method will continually re-execute the first standard bandit (i.e. controller) whose cost is observed to be less than $-\beta$. If after e many samples of positive costs, a cost below $-\beta$ is sampled, then that bandit will be resampled $\hat{E} - e$ many remaining iterations. We first note the probability of sampling less and greater than $-\beta$ is respectively $\Phi(-\beta)$ and $\Phi(\beta)$. Second, we note – using truncated Gaussian identities (Appendix A.2.5) – the expected value of sampling a standard Gaussian, conditioned on the sample being less and greater than $-\beta$, is respectively $\phi(\beta)/(\Phi(\beta)-1)$ and $\phi(\beta)/\Phi(\beta)$, where ϕ is the standard Gaussian density and Φ the cumulative density. So the predicted reduction in loss using PILCO's greedy strategy is predicted to be

$$\begin{aligned}\mathbb{E}\left[\mathbf{J}^{\text{pred.}}|UCB\right] &= \sum_{e=0}^{\hat{E}} \Phi(-\beta)\Phi(\beta)^e \left((\hat{E}-e)\frac{\phi(\beta)}{\Phi(\beta)-1} + e\frac{\phi(\beta)}{\Phi(\beta)}\right) + \Phi(\beta)^{\hat{E}}\hat{E}\frac{\phi(\beta)}{\Phi(\beta)} \\ &= \phi(\beta)\left[-\frac{\hat{E}+1}{\Phi(-\beta)} + \frac{1-\Phi(\beta)^{\hat{E}+1}}{\Phi(\beta)\Phi(-\beta)^2} + \frac{\hat{E}\Phi(\beta)^{\hat{E}}\Phi(-\beta)-1}{\Phi(\beta)}\right] \quad (4.37)\end{aligned}$$

which we use to compute some elements of the bottom row in Table 4.2. We could further use our analytical solution in (4.37) to optimise β, to avoid setting it as a free parameter. The optimal $\beta = 0.968$, yields an average predicted return of $\mathbb{E}\left[\mathbf{J}^{\text{predicted}}/\hat{E}\right] = -1.079$.

Since both PI and EI are exploration heuristics, and not derived on a set of principles on how to optimise the total loss $\mathbf{J}$ (indeed neither is a function of the number of trials remaining $\hat{E}$) there is not necessarily any obvious way to estimating how much each heuristic can reduce $\mathbf{J}$. Nevertheless EI outputs values in the same units of our objective function (opposed to PI, which outputs unitless probabilities) being the expected amount of improve for a single episode's loss. By computing the expected improvement on sampling one standard Gaussian, against a reference value $\mathcal{C}^*$ itself a standard Gaussian, we get the value -0.564. However, we again

note our approximate prediction of -0.564 is constant w.r.t. $\hat{E}$, whereas the empirical value grows with $\hat{E}$.

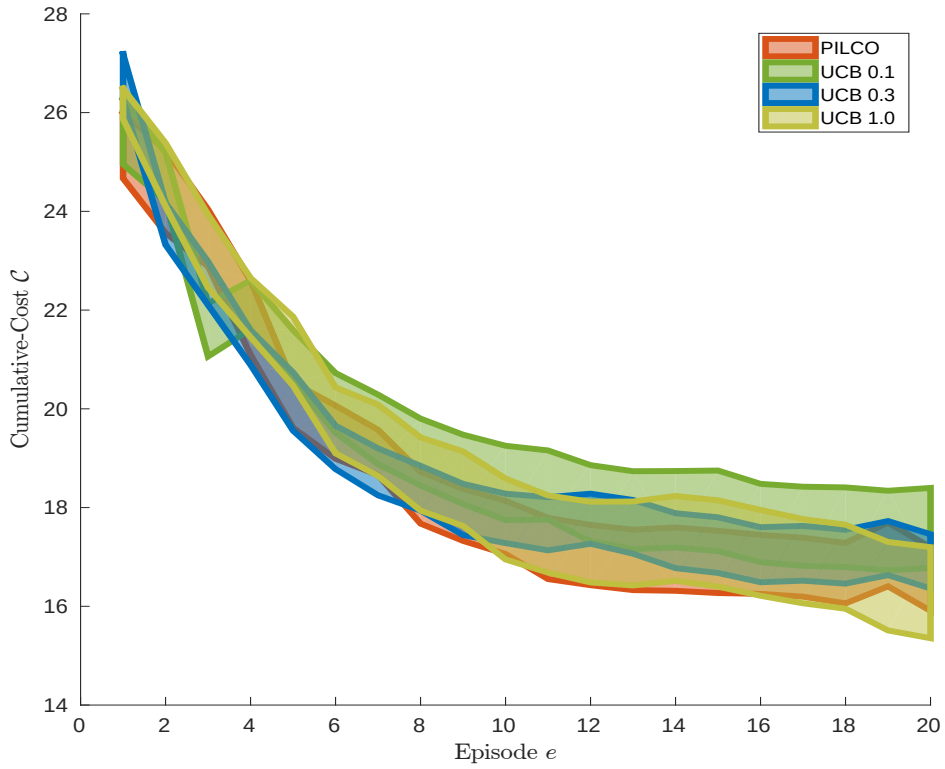
GI is instead a principled Bayes-optimal approach for selecting which bandit to execute. Using (4.20), our approximate upper bound is $GI^{\text{upper}}(\mu_e = 0, \sigma_e^{\mathcal{C}} = 1, \sigma_y = 0, \gamma = 1 - 1/\hat{E}) = -1.177$, where $E = 20$. The meaning of value -1.177 is the average predicted reduction of an episode's loss J_e (averaged over episodes). I.e. we expect the total loss $\mathbf{J}$ to decrease by up to $-1.177 \cdot \hat{E} = -24.71$ by following the approximate GI exploration strategy. Notice the *predicted* value -1.177 is not equal to the *empirical* value -1.093 in Table 4.2. As mentioned previously, in § 4.3.4, our GI^{upper} method corresponds to the true Gittins index under three assumptions: 1) a finite horizon may be approximated as an infinite horizon, using $\gamma = 1 - 1/\hat{E}$, 2) bandit costs are uncorrelated 3) bandit uncertainty is reduced to zero after one execution. In our bandits experiment, whose results are Table 4.2, we satisfied assumption 2 and 3 above by using independent bandits and observation noise $\sigma_y = 0$ respectively. However, assumption 1 remains violated, resulting in a discrepancy between predicted and empirical GI values.

CART DOUBLE-POLE SWING-UP TASK

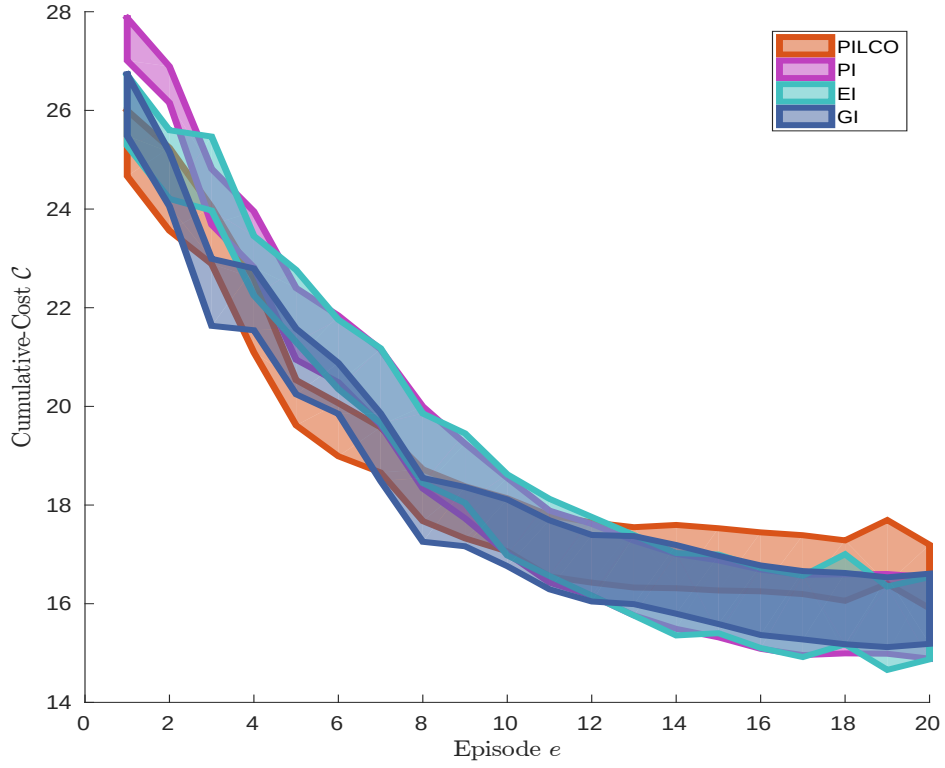
Each algorithm was then tested on the cart double-pole swing-up task 20 times. The results of the cumulative-cost per episode for each algorithm is shown Fig. 4.3. The goal was to minimise the *total* cumulative-cost, totalled across all $E = 20$ episodes, summarised by Table 4.3. The results show some exploration schemes increase PILCO's data efficiency. Interestingly PI performs *worse* than PILCO. However, PI is known to succumb to highly local search when the posterior belief over nearby inputs are highly correlated (Brochu et al., 2010). We discussed this pathological case of PI targeting high probabilities of *insignificant* improvements previously, § 4.3.2. Perhaps also for the reason of strong cost-correlation between controllers of similar

Table 4.3 ***Cart double-pole cumulative-cost performance, averaged over all episodes***

$BO(C_e^{\Psi})$ param.	PILCO	UCB $\beta=0.1$	UCB $\beta=0.3$	UCB $\beta=1.0$	PI	EI	GI
$\mathbb{E}_{\text{seeds}}\left[\frac{\mathbf{J}^{\text{empirical}}}{E}\right]$	18.82	19.48	18.97	19.07	19.11	19.00	**18.58**
standard error	0.46	0.60	0.42	0.65	0.58	0.63	0.55
$\mathbb{V}_{\text{seeds}}\left[\frac{\mathbf{J}^{\text{empirical}}}{E}\right]^{1/2}$	2.05	2.69	1.87	2.91	2.61	2.80	2.45



(a) PILCO *compared with UCB baselines.*



(b) PILCO *compared with Bayesian optimisation algorithms.*

Fig. 4.3 ***Cart double-pole swing-up performance per training episode.*** *Shown is the mean cumulative-cost performance ± the standard error (the standard deviation of performance divided by $\sqrt{\#\ seed}$). Each algorithm improves per episode from a maximum cumulative-cost of $T+1=31$, and asymptotes towards the final $E=20$ episode.*

parameterisations, our approximate GI strategy did not outperform the EI heuristic (the true GI is only valid under zero correlations). Both GI and EI outperform the original PILCO. Interestingly, GI had a much smaller loss variance than EI. Due to the limited number of trials, 20 repetitions for each type of exploration strategy, we cannot state with high statistical significance that the EI and GI exploration strategies outperform PILCO with high confidence. However, Table 4.3 still provides some indication that EI and GI assist PILCO to explore more initially, and find superior controllers after the fixed number of trials have ended, shown Fig. 4.3.

A limitation of this current exploration work is poor performance on systems with either nontrivial process noise ε_t^x or observation noise ε_t^y. The reason is our method cannot distinguish between *subjective* dynamics uncertainty in $p(f)$ from *inherent* sources of uncertainty that persist such as ε_t^x and ε_t^y. As our system cannot distinguish between *reducible* subjective forms of uncertainties and persistent sources, the 'exploration' strategy is easily misled into repeatedly executing some controller π^ψ with large cumulative-cost variance whose variance is dominated by persistent uncertainties. This is not the point of BO at all, the point is to target reducible forms of uncertainty which give the robot a chance to learn if a controller has *potentially* very small losses or not, information that will be gained only once a controller's cumulative-cost variance collapses. Thus, our work in the above section is only suitable when total uncertainty is dominated by subjective model uncertainties. We experimented above using deterministic dynamics because of this limitation, but in the next chapter we extend into exploration with stochastic dynamics.

4.5 *Distinguishing between Aleatoric and Epistemic Uncertainty

In the previous section, we discussed how controllers cumulative-cost have high variance are attractive to execute from the Bayesian optimisation point of view. This was because high variance indicated exploratory value: perhaps revealing that the controller π^ψ has much lower cumulative-cost $\mu_{e+1}^{\mathcal{C}}$ than currently believed (i.e. might be much lower than our current mean-posterior) $\mu_e^{\mathcal{C}}$. The benefit is that surprisingly-good controllers can be exploited *repeatedly* in future episodes, whereas surprisingly-poor controllers need not be executed again once revealed how poor they really are. However, a complication is that a high variance $\Sigma_e^{\mathcal{C}}$ is not always an indication that more can be learned from executing controller

π^{ψ}. For example, the cumulative-cost variance might be entirely due to inherent and persistent system stochasticities. In this § 4.5, we investigate how to detect if the variance $\Sigma_e^{\mathcal{C}}$ is reducible before executing the system, i.e. whether we can learn something potentially useful from our exploratory efforts. We present a new exploration strategy to the reader, an improvement over our previous strategy in § 4.4. However, this section nevertheless represents unfinished work as we have not yet conducted experimentation to validate our approach. As such we mark this section optional to the reader.

We begin by considering two distinct types of uncertainty – considered by other engineering fields concerned with risk (Der Kiureghian and Ditlevsen, 2009; Ferson et al., 2004) – each possibly contributing to $\Sigma_e^{\mathcal{C}}$:

- *Aleatoric* uncertainties: inherent randomness. For example, process noise ε_t^x, observation noise ε_t^y, random state initialisation ε^{x_0}. In our time-invariant control setting, we assume all aleatoric uncertainties are persistent and irreducible.

- *Epistemic* uncertainties: subjective uncertainty arising from limited knowledge about the true dynamics f. We quantify our subjective uncertainty of the dynamics with a probabilistic model $p(f)$. Since the robot improves its knowledge of the dynamics after each episode, epistemic uncertainties are reducible.

Since $\Sigma_e^{\mathcal{C}}$ is caused by both irreducible-aleatoric and reducible-epistemic uncertainties, we can distinguish how much $\Sigma_e^{\mathcal{C}}$ is caused by epistemic uncertainties if we can simulate how reducible $\Sigma_e^{\mathcal{C}}$ if we were to collect more data. Note the variance reduction ($\Sigma_{e+1}^{\mathcal{C}}$ - $\Sigma_e^{\mathcal{C}}$) can only be caused by model uncertainties being updated with more data. Thus, to develop an improved exploration strategy over those of Table 4.3 which use the form $BO(\mathcal{C}_e^{\psi})$, we will swap the random input $\mathcal{C}_e^{\psi}$ for another which only contains *reducible* variance. To get a handle on if variance is reducible, we fantasise about possible uncertain future data $\mathcal{D}_{e+1} = \{X_{e+1}, y_{e+1}\}$ and re-simulate, to understand if such data reduces variance (under expectation of uncertain future data). We compute the expected reduction in variance as $\Delta\Sigma^{\mathcal{C}} \doteq \Sigma_e^{\mathcal{C}} - \mathbb{E}_{\mathcal{D}_{e+1}}\left[\Sigma_{e+1}^{\mathcal{C}}\right]$. We again note that any variance reduction must be the result of improving system knowledge, since it cannot be a result of the unalterable system stochasticity. The uncertain future data for the next episode has a distribution, computed from the current predictive joint state distribution from § 4.4.1: $\mathcal{D}_{e+1} \sim p(X_{0:T}|\psi) = p(X_0, X_1, ..., X_T|\psi)$, given the dynamics model $p(f)$ at episode e. By law

of iterated expectation and variance:

$$\mu_e^{\mathcal{C}} = \mathbb{E}_{\mathcal{D}_{e+1}}\left[\mu_{e+1}^{\mathcal{C}}\right], \tag{4.38}$$

$$\Sigma_e^{\mathcal{C}} = \mathbb{E}_{\mathcal{D}_{e+1}}\left[\Sigma_{e+1}^{\mathcal{C}}\right] + \mathbb{V}_{\mathcal{D}_{e+1}}\left[\mu_{e+1}^{\mathcal{C}}\right], \tag{4.39}$$

and so the reducible variance

$$\Delta\Sigma^{\mathcal{C}} \doteq \Sigma_e^{\mathcal{C}} - \mathbb{E}_{\mathcal{D}_{e+1}}\left[\Sigma_{e+1}^{\mathcal{C}}\right] \tag{4.40}$$

$$= \mathbb{V}_{\mathcal{D}_{e+1}}\left[\mu_{e+1}^{\mathcal{C}}\right], \tag{4.41}$$

has an alternate interpretation. From (4.41) we see the reducible variance is equivalent to asking how might uncertain future data $\mathcal{D}_{e+1}$ affect the future cumulative-cost-mean? Often in control and RL, we are greedy about optimising the cumulative-cost-mean, as PILCO did. And when the dynamics f is known with certainty, the Bayes-optimal control does reduce to 'greedily' optimising $\mu_e^{\mathcal{C}} = \mu_{e+1}^{\mathcal{C}}$. However, under epistemic uncertainty, $\mu_e^{\mathcal{C}} \neq \mu_{e+1}^{\mathcal{C}}$ in general, complicating the search for a Bayes-optimal controller. Yet (4.41) relates our approximate Bayes-optimal solution to PILCO by similar optimisation of some $\mu^{\mathcal{C}}$. The only difference is we intend to optimise an uncertain $\mu_{e+1}^{\mathcal{C}}$, opposed to the certain $\mu_e^{\mathcal{C}}$. As before, we use BO for exploration. Instead of optimising $BO(\mathcal{C}_e^{\Psi})$ as we did previously § 4.4, this § 4.5 optimises $BO(\mu_{e+1}^{\mathcal{C}})$. Also note by considering only the following episode's potential data $\mathcal{D}_{e+1}$, and not $\mathcal{D}_{e+2}$, our BRL exploration algorithm remains myopic. Although, if we choose GI as our BO method, our algorithm is only partially-myopic: myopic by only considering learning one more timestep, yet non-myopic by being sensitive to the number of episodes remaining $\hat{E}$, unlike PI and EI. Indeed if $\sigma_y = 0$, then our 'myopic' algorithm GI^{lower} is equal to the 'non-myopic' GI^{true}.

4.5.1 Gaussian Process Variance Reduction

Now that we have decided our new objective function to minimise, $BO(\mu_{e+1}^{\mathcal{C}})$ (where BO could be PI, EI, or GI), we discuss how to compute the approximate distribution of $\mu_{e+1}^{\mathcal{C}}$, fitted as Gaussian. To compute $\Delta\Sigma^{\mathcal{C}}$ we must first inspect our current dynamics model $p(f)$ trained with a current dataset $\mathcal{D} = \mathcal{D}_{1:e}$, to understand how our model might update $p(f|\mathcal{D}_{e+1})$ and how predictions might be affected given additional currently-uncertain future data $\mathcal{D}_{e+1}$. This subsection addresses approximate computation of the expected variance reduction of a GP's prediction, a reduction caused by conditioning on uncertain future data $\mathcal{D}_{e+1}$. **The equations** (4.42) **–** (4.54) **within**

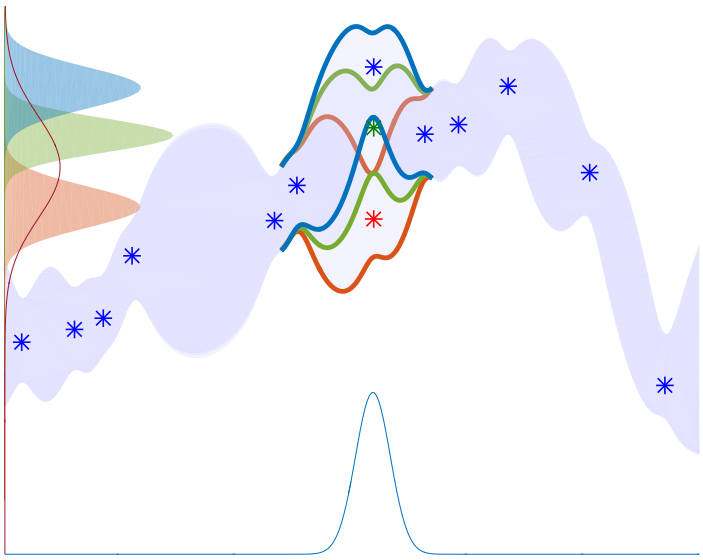


Fig. 4.4 ***Schematic of expected variance.*** *Input distribution below. Coloured (RGB) are GP posteriors conditioned on fantasy data and corresponding output distributions. Marginal output distribution in a line. This figure was produced by Mark van der Wilk.*

this subsection were not derived by myself, but by Mark van der Wilk during a collaboration.

For intuition into how our dynamics might change, let us consider Monte Carlo (MC) estimation. Say we sample from uncertain $\mathcal{D}_{e+1} \sim p(X_{0:T})$ a single fantasy datum with input $\hat{x}_t$ and output $\hat{x}_{t+1}$. Given $\{\hat{x}_t, \hat{x}_{t+1}\}$, we can update our model $p(f|\hat{x}_t, \hat{x}_{t+1})$ to understand the effects on individual state predictions. In Fig. 4.4 we show different ways a GP dynamics model posterior can changes in response to three different MC-sampled new datum $\{\hat{x}_t, \hat{x}_{t+1}\}$; red, green, and blue. All existing data is shown with blue star symbols, and the new data shown is possibly either red, green, or blue star. In general the input and output of the fantasy datum is uncertain, but as an approximation, we assume the fantasy-input $\hat{x}_t$ is certain (located at its mean prediction). Notice how the posterior of the GP changes in different ways, with red, green, blue outlines showing how the posterior would change depending on the fantasy-output value. Depending how the model posterior changes; red, green, or blue; the GP will predict differently. Our uncertain-input to the GP is given by the Gaussian on the horizontal axis, and the three possible predictions are given by the solid coloured Gaussian distributions on the vertical axis. The lone thin-red outline on the vertical axis represents the marginal variance output (before conditioning on a fantasy datum). Notice the marginal variance is larger than the *average*-variance of the red-green-blue distributions. This means we anticipate the

GP posterior and predictions will reduce their variance, we just do not yet know in what way.

MC samples may help to intuitively understand how GP predictions change but they are too computationally heavy for our purposes (the computational complexity of PILCO would increase a factor of the amount of MC samples). Instead, we seek a tractable analytic approximation, integrating out the unknown fantasy observation $\hat{x}_{t+1}$. We begin with some new notation:

$\tilde{x}_t$: input state,	x_{t+1} : output state,
$\hat{x}_t$: fantasy input,	$\hat{x}_{t+1}$: fantasy output,
X : current GP inputs,	y : current GP observations,
$\bar{X}$: X augmented with fantasy input,	$\bar{y}$: y augmented with fantasy output,
$\sigma^2_{\hat{x}_{t+1}}$: variance of fantasy output,	$\hat{\mathcal{D}}$: $\{\hat{x}_t, \hat{x}_{t+1}\}$ fantasy datum.

The rank-1 update of our GP dynamics model gram matrix K_{XX} given a *single* new fantasy datum $\hat{\mathcal{D}} = \{\hat{x}_t, \hat{x}_{t+1}\}$ is:

$$\begin{aligned} \bar{K}^{-1}_{\bar{X}\bar{X}} &= \begin{bmatrix} K_{XX} & \mathbf{k}_{\bar{X}\hat{x}_t} \\ \mathbf{k}^\intercal_{\bar{X}\hat{x}_t} & k_{\hat{x}_t\hat{x}_t} \end{bmatrix}^{-1} \\ &= \begin{bmatrix} K^{-1}_{XX} + K^{-1}_{XX}\mathbf{k}_{X\hat{x}_t}\mathbf{k}^\intercal_{X\hat{x}_t}K^{-1}_{XX}\sigma^{-2}_{\hat{x}_{t+1}|\hat{\mathcal{D}}} & -K^{-1}_{XX}\mathbf{k}_{X\hat{x}_t}\sigma^{-2}_{\hat{x}_{t+1}|\hat{\mathcal{D}}} \\ -\mathbf{k}^\intercal_{X\hat{x}_t}K^{-1}_{XX}\sigma^{-2}_{\hat{x}_{t+1}|\hat{\mathcal{D}}} & \sigma^{-2}_{\hat{x}_{t+1}|\hat{\mathcal{D}}} \end{bmatrix}, \end{aligned} \tag{4.42}$$

$$\sigma^{-2}_{\hat{x}_{t+1}|\hat{\mathcal{D}}} \doteq \left(k_{\hat{x}_t\hat{x}_t} + \sigma^2_{\hat{x}_{t+1}} - \mathbf{k}^\intercal_{X\hat{x}_t}K^{-1}_{XX}\mathbf{k}_{X\hat{x}_t}\right)^{-1}. \tag{4.43}$$

The term to estimate (detailed over the next 2 pages) is the expected variance of a predicted state conditioned on fantasy data:

$$\begin{aligned} \bar{\Sigma}^x_{t+1,e+1} &\approx \mathbb{E}_{\hat{x}_{t+1}}\left[\mathbb{V}_{\tilde{x}_t,f}\left[x_{t+1}|\tilde{x}_t,\hat{x}_{t+1}\right]\right] \\ &= \mathbb{E}_{\hat{x}_{t+1}}\left[\underbrace{\mathbb{E}_{\tilde{x}_t}\left[\mathbb{V}_f\left[x_{t+1}|\tilde{x}_t,\hat{x}_{t+1}\right]\right]}_{\text{Constant w.r.t. } \hat{x}_{t+1}} + \mathbb{V}_{\tilde{x}_t}\left[\mathbb{E}_f\left[x_{t+1}|\tilde{x}_t,\hat{x}_{t+1}\right]\right]\right], \end{aligned} \tag{4.44}$$

using an approximate equality since we assume $\Sigma^x_{t+1,e+1}$ is only affected by a single data-point within $\mathcal{D}_{e+1}$, even though $\mathcal{D}_{e+1}$ will contain T new data-points which will correlate and each affect every timestep dynamics prediction in general. We now discuss how to compute the first and second term of (4.44).

Expectation of the variance

For the first term in (4.44) we can ignore the expectation over $\hat{x}_{t+1}$:

$$\begin{aligned}
&\mathbb{E}_{\hat{x}_{t+1}}\left[\mathbb{E}_{\tilde{x}_t}\left[\mathbb{V}_f\left[x_{t+1}|\tilde{x}_t,\hat{x}_{t+1}\right]\right]\right] \\
&= \mathbb{E}_{\tilde{x}_t}\left[k_{\tilde{x}_t\tilde{x}_t} - \mathbf{k}_{\bar{X}\tilde{x}_t}^\intercal \bar{K}_{\bar{X}\bar{X}}^{-1}\mathbf{k}_{\bar{X}\tilde{x}_t}\right] \\
&= \mathbb{E}_{\tilde{x}_t}\left[k_{\tilde{x}_t\tilde{x}_t} - \begin{bmatrix}\mathbf{k}_{X\tilde{x}_t}\\ k_{\hat{x}_t\tilde{x}_t}\end{bmatrix}^\intercal \bar{K}_{\bar{X}\bar{X}}^{-1}\begin{bmatrix}\mathbf{k}_{X\tilde{x}_t}\\ k_{\hat{x}_t\tilde{x}_t}\end{bmatrix}\right] \\
&= \mathbb{E}_{\tilde{x}_t}\left[k_{\tilde{x}_t\tilde{x}_t} - \mathbf{k}_{X\tilde{x}_t}^\intercal K_{XX}^{-1}\mathbf{k}_{X\tilde{x}_t}\right] - \sigma_{\hat{x}_{t+1}|\hat{\mathcal{D}}}^{-2}\mathbb{E}_{\tilde{x}_t}\left[\left(\mathbf{k}_{X\hat{x}_t}^\intercal K_{XX}^{-1}\mathbf{k}_{X\tilde{x}_t} - k_{\hat{x}_t\tilde{x}_t}\right)^\intercal\left(\mathbf{k}_{X\hat{x}_t}^\intercal K_{XX}^{-1}\mathbf{k}_{X\tilde{x}_t} - k_{\hat{x}_t\tilde{x}_t}\right)\right] \\
&= \Sigma_{t+1}^x - \underbrace{\sigma_{\hat{x}_{t+1}|\hat{\mathcal{D}}}^{-2}[\mathbf{k}_{X\hat{x}_t}^\intercal K_{XX}^{-1}, -1]\mathbb{E}_{\tilde{x}_t}\left[\mathbf{k}_{\bar{X}\tilde{x}_t}\mathbf{k}_{\bar{X}\tilde{x}_t}^\intercal\right][\mathbf{k}_{X\hat{x}_t}^\intercal K_{XX}^{-1}, -1]^\intercal}_{\geq 0},
\end{aligned} \tag{4.45}$$

which is the original expected variance (before fantasy data) minus a non-negative reduction term.

Variance of the expectation

Now we compute the second term in (4.44). Starting with the variance of the expectation, without the expectation over the fantasy data:

$$\mathbb{V}_{\tilde{x}_t}\left[\mathbb{E}_f\left[x_{t+1}|\tilde{x}_t,\hat{x}_{t+1}\right]\right] = \mathbb{V}_{\tilde{x}_t}\left[\mathbf{k}_{\bar{X}\tilde{x}_t}^\intercal \bar{K}_{\bar{X}\bar{X}}^{-1}\bar{y}\right] = \mathrm{Tr}\left(\mathbb{V}_{\tilde{x}_t}\left[\mathbf{k}_{\bar{X}\tilde{x}_t}\right]\bar{K}_{\bar{X}\bar{X}}^{-1}\bar{y}\bar{y}^\intercal\bar{K}_{\bar{X}\bar{X}}^{-1}\right). \tag{4.46}$$

Now the expectation over fantasy output $\hat{x}_{t+1}$ is:

$$\mathbb{E}_{\hat{x}_{t+1}}\left[\mathbb{V}_{\tilde{x}_t}\left[\mathbb{E}_f\left[x_{t+1}|\tilde{x}_t,\hat{x}_{t+1}\right]\right]\right] = \mathrm{Tr}\left(\mathbb{V}_{\tilde{x}_t}\left[\mathbf{k}_{\bar{X}\tilde{x}_t}\right]\bar{K}_{\bar{X}\bar{X}}^{-1}\mathbb{E}_{\hat{x}_{t+1}}\left[\bar{y}\bar{y}^\intercal\right]\bar{K}_{\bar{X}\bar{X}}^{-1}\right), \tag{4.47}$$

where the expectation over the outer product is:

$$\mathbb{E}_{\hat{x}_{t+1}}\left[\bar{y}\bar{y}^\intercal\right] = \begin{bmatrix} yy^\intercal & y\mathbb{E}\left[\hat{x}_{t+1}\right] \\ y^\intercal\mathbb{E}\left[\hat{x}_{t+1}\right] & \mathbb{E}\left[\hat{x}_{t+1}^2\right]\end{bmatrix}, \tag{4.48}$$

$$\mathbb{E}\left[\hat{x}_{t+1}\right] = \mathbf{k}_{X\hat{x}_t}^\intercal K_{XX}^{-1}y, \tag{4.49}$$

$$\mathbb{E}\left[\hat{x}_{t+1}^2\right] = \sigma_{\hat{x}_{t+1}|\hat{\mathcal{D}}}^2 + \mathbb{E}\left[\hat{x}_{t+1}\right]^2. \tag{4.50}$$

$\mathbb{E}_{\hat{x}_{t+1}}\left[\bar{y}\bar{y}^\intercal\right]$ is not rank 1 anymore, but it can be decomposed into a $\mathbb{R}^{(N+1)\times 2}$ matrix:

$$\mathbb{E}_{\hat{x}_{t+1}}\left[\bar{y}\bar{y}^\intercal\right] = \tilde{y}\tilde{y}^\intercal, \qquad \tilde{y} = \begin{bmatrix} y & 0 \\ \mathbb{E}\left[\hat{x}_{t+1}\right] & \sigma_{\hat{x}_{t+1}|\hat{\mathcal{D}}}\end{bmatrix}. \tag{4.51}$$

Now we can rewrite $\mathbb{E}_{\hat{x}_{t+1}}\left[\mathbb{V}_{\tilde{x}_t}\left[\mathbb{E}_f\left[x_{t+1}|\tilde{x}_t,\hat{x}_{t+1}\right]\right]\right]$ as a trace with $\tilde{y}$ on the outside again, which allows for much easier evaluation:

$$\mathbb{E}_{\hat{x}_{t+1}}\left[\mathbb{V}_{\tilde{x}_t}\left[\mathbb{E}_f\left[x_{t+1}|\tilde{x}_t,\hat{x}_{t+1}\right]\right]\right] = \operatorname{Tr}\left(\tilde{y}^\intercal \bar{K}_{\bar{X}\bar{X}}^{-1}\mathbb{V}_{\tilde{x}_t}\left[\mathbf{k}_{\bar{X}\tilde{x}_t}\right]\bar{K}_{\bar{X}\bar{X}}^{-1}\tilde{y}\right), \tag{4.52}$$

$$\bar{K}_{\bar{X}\bar{X}}\tilde{y} = \begin{bmatrix} \bar{K}_{\bar{X}\bar{X}}^{-1}y & 0 \\ 0 & \sigma^{-1}_{\hat{x}_{t+1}|\hat{\mathcal{D}}} \end{bmatrix}, \tag{4.53}$$

$$\therefore \mathbb{E}_{\hat{x}_{t+1}}\left[\mathbb{V}_{\tilde{x}_t}\left[\mathbb{E}_f\left[x_{t+1}|\tilde{x}_t,\hat{x}_{t+1}\right]\right]\right] = y^\intercal K_{XX}^{-1}\mathbb{V}_{\tilde{x}_t}\left[\mathbf{k}_{X\tilde{x}_t}\right]K_{XX}^{-1}y + \mathbb{V}_{\tilde{x}_t}\left[k_{\tilde{x}_t\tilde{x}_t}\right]\sigma^{-2}_{\hat{x}_{t+1}|\hat{\mathcal{D}}} \tag{4.54}$$

We are now able to compute $\bar{\Sigma}^x_{t+1,e+1}$ from (4.44) as the sum of (4.45) and (4.54), using identities in Appendix B.1 to compute the final $\mathbb{E}_{\tilde{x}_t}[\cdot]$ and $\mathbb{V}_{\tilde{x}_t}[\cdot]$ terms. However, we have only approximately derived $\bar{\Sigma}^x_{t+1,e+1}$ for a 1-rank update, but in general $\bar{\Sigma}^x_{t+1,e+1}$ will update according to all T datum in $\mathcal{D}_{e+1}$ requiring a T-rank update.

4.5.2 System Simulation with Dual Uncertainties

In § 4.5.1 we outlined how the certainty of a GP's predictions is expected to change upon receiving a single additional datum $\hat{\mathcal{D}}$. However, we only discussed how such predictions change for a single timestep, from t to $t+1$. Given we need to compute how the *cumulative*-cost changes given more data, we must understand the relation between successive state prediction changes from time $t=0$ through to $t=T$. I.e. the state variance $\bar{\Sigma}^x_{t,e+1}$ is affected by new data $\mathcal{D}_{e+1}$ both directly, by affected the model $p(f|\mathcal{D}_{e+1})$ which outputs $\Sigma^x_{t,e+1}$, and by affecting an input to model $\Sigma^x_{t-1,e+1}$. In this subsection we discuss PILCO's system simulation phase with aleatoric and epistemic uncertainties, before discussing controller evaluation in § 4.5.3.

Similar to PILCO, we simulate forwards a sequence of uncertain states, from X_0 to X_T. The difference now is two types of uncertainty exist, not one. In PILCO each state distribution was distributed $X_t \sim \mathcal{N}(\mu^x_t, \Sigma^x_t)$, but now we keep distinct the contributions from aleatoric and epistemic random sources, which combined have variance Σ^x_t. As such, we 'divide up' Σ^x_t into two additive components, 1) the (expected) reduced variance $\bar{\Sigma}^x_{t,e+1}$ from (4.44), and 2) the amount of reduction $\Delta\Sigma^x_t = \Sigma^x_t - \bar{\Sigma}^x_{t,e+1}$. If the dynamics f were linear then simulation could proceed as PILCO did with the trivial change of exchanging Σ^x_t for $\bar{\Sigma}^x_{t,e+1}$ at each timestep t. Unfortunately, nonlinear dynamics bring two issues, 1) the certainty-equivalence principle no longer applies, so an exchange of variances would erroneously-alter the expected state mean trajectory, violating (4.38), and 2) both variance types are not easily decoupled, since $\Delta\Sigma^x_t$ affects $\bar{\Sigma}^x_{t+1,e+1}$ and $\bar{\Sigma}^x_{t,e+1}$ affects $\Delta\Sigma^x_{t+1}$. Nevertheless, in § 3.3 we developed a framework to forwards simulate hierarchical belief distributions.

Here, *hierarchical* is not as important as the fact that § 3.3 handles *two* variance components, $B \sim \mathcal{N}(M, V)$ where $M \sim \mathcal{N}(\mu^m, \Sigma^m)$, with Σ^m the 'variance of the mean' and V the 'mean of the variance'. Similarly, we can define the latent state to compose dual variance types. So instead of $X_t \sim \mathcal{N}(m^x, \Sigma_t^x)$, we can use:

$$X_t \sim \mathcal{N}\left(M^x, \bar{\Sigma}_{t,e+1}^x\right), \quad M^x \sim \mathcal{N}\left(\mu_t^x, \Delta\Sigma_t^x\right), \tag{4.55}$$

which we are able to simulate using the mathematics of GP predictions given hierarchically-distributed objects in Appendix B.4, and used previously in Chapter 3.

Now, to summarise what we have derived so far. We discussed: 1) how an uncertain state X_t can propagate forwards to a *hierarchically*-uncertain state X_{t+1} due to random data $\mathcal{D}_{e+1}$ in § 4.5.1, and 2) how a hierarchically-uncertain state X_t can propagate forwards to another hierarchically-uncertain state X_{t+1} without new data in § 3.3 and Appendix B.4. However, what remains is doing both: how a hierarchically-uncertain state X_t can propagate forwards to another hierarchically-uncertain state X_{t+1} under uncertain data $\mathcal{D}_{e+1}$, which we leave to future work.

4.5.3 Controller Evaluation with Dual Uncertainties

Here we discuss how to evaluate a controller given a simulation of dual uncertainties. Controller evaluation is necessary for controller optimisation as we saw in Algorithm 1. Because the cost function is nonlinear, we must continue to handle the hierarchical-nature of the state distributions with two distinct variances which, as noted in § 4.5.2, are not easily decoupled. Given (4.55) as input, the hierarchical cost moments can be computed by randomising the definition of the usual cost moments, cost-mean m^c (4.31) and cost-variance V^c (4.32) on page 112. The expected cost-mean, variance of the cost-mean, and expected cost-variance are respectively:

$$\begin{aligned}
\mu_t^c &\doteq \mathbb{E}_{M^x}[M^c] && (4.56)\\
&= \mathbb{E}\left[\text{cost}(\mathcal{N}\left(\mu_t^x, \Delta\Sigma_t^x + \bar{\Sigma}_{t,e+1}^x\right); x^*, \Lambda_c)\right], \\
\Sigma_t^c &\doteq \mathbb{V}_{M^x}[M^c] && (4.57)\\
&= \det(I + \bar{\Sigma}_{t,e+1}^x \Lambda_c^{-1})^{-1} \mathbb{V}_{M^x}\left[\text{cost}(M^x; x^*, (I + \bar{\Sigma}_{t,e+1}^x \Lambda_c^{-1})\Lambda_c)\right], \\
\bar{V}_t^c &\doteq \mathbb{E}_{M^x}[V^c] && (4.58)\\
&= \mathbb{V}\left[\text{cost}(\mathcal{N}\left(\mu_t^x, \Delta\Sigma_t^x + \bar{\Sigma}_{t,e+1}^x\right); x^*, \Lambda_c)\right] - \mathbb{V}\left[\text{cost}(\mathcal{N}\left(\mu_t^x, \Delta\Sigma_t^x\right); x^*, \Lambda_c)\right],
\end{aligned}$$

with derivations in Appendix E.5, function $\mathbb{E}[\text{cost}(\cdot)]$ is computed with (4.31), and $\mathbb{V}[\text{cost}(\cdot)]$ with (4.32).

Now we have most tools required to distinguish aleatoric from epistemic uncertainty in the cumulative-cost C^{ψ}, our goal from the beginning of § 4.5. One important tool we still lack is a hierarchical form of the cost-cost covariance, analogous to the non-hierarchical cost-cost covariance of § 4.4.1. However, if we approximate the joint cost-cost by ignoring all cross covariance terms (which as stated previously, contribute between 40-85% of the cumulative-cost variance), and focus on each cost's marginal hierarchical-distributions (4.56) – (4.58) then the *approximate* cumulative-cost uncertainties are

$$C_e^{\psi} \doteq \sum_{t=0}^{T} \text{cost}(X_t)|\psi \sim \mathcal{N}\left(\mu_e^{\mathcal{C}}, \Sigma_e^{\mathcal{C}} = \Sigma_e^{\mathcal{C},\text{aleatoric}} + \Sigma_e^{\mathcal{C},\text{epistemic}}\right), \quad (4.59)$$

$$\mu_e^{\mathcal{C}} = \sum_{t=0}^{T} \mathbb{E}_X\left[\text{cost}(X_t)|\psi\right], \quad (4.60)$$

$$\Sigma_e^{\mathcal{C},\text{aleatoric}} \approx \sum_{t=0}^{T} \bar{V}_t^c, \quad (4.61)$$

$$\Sigma_e^{\mathcal{C},\text{epistemic}} \doteq \Delta\Sigma^{\mathcal{C}} \approx \sum_{t=0}^{T} \Sigma_t^c, \quad (4.62)$$

where $\Delta\Sigma^{\mathcal{C}}$ is the expected change in cumulative-cost uncertainty, discussed previously (4.40), a reduction which can only result from improving our epistemic knowledge about the plausible set of dynamics $p(f)$. Thus our new suggested exploration strategy, accompanied with our previous strategies are:

1. Original PILCO from § 2.5, minimises $\mu_e^{\mathcal{C}}$, defined (4.23),
2. Modified PILCO that directs exploration with cumulative-cost uncertainty (§ 4.4) by minimising $BO(C_e^{\psi})$, where $C_e^{\psi} \sim \mathcal{N}\left(\mu_e^{\mathcal{C}}, \Sigma_e^{\mathcal{C}}\right)$, defined (4.22) – (4.24),
3. Modified PILCO that directs exploration with *epistemic* cumulative-cost uncertainty (§ 4.5) by minimising $BO(\mu_{e+1}^{\mathcal{C}})$, where $\mu_{e+1}^{\mathcal{C}} \sim \mathcal{N}\left(\mu_e^{\mathcal{C}}, \Sigma_e^{\mathcal{C},\text{epistemic}}\right)$, from (4.62).

We were able to show in § 4.4 than the second algorithm used exploration to yield improved data efficiency on the first algorithm above – the original PILCO. We think the third algorithm holds even greater promise, although work remains to 1) complete the system simulation in § 4.5.2, 2) improve the accuracy of the controller evaluation in § 4.5.3 by considering cost-cost covariance under dual uncertainties, and 3) experimental validation.

4.6 Discussion and Future Work

In this chapter on exploration we discussed the problem of dual control: how to control a system well whilst we are simultaneously learning about the system dynamics. In RL, this problem is often interpreted as a dilemma, between optimising system performance w.r.t. current system knowledge (exploitation), and testing different control outputs intended to improve our system knowledge (exploration). In the context of episodic PILCO, exploitation seeks to minimise the loss of the current episode, whereas exploratory control outputs might minimise the loss of multiple future episodes. Often in the RL literature, such exploration is *undirected*, where robots decide exploratory control outputs independent of the robot's uncertainties (e.g. Boltzmann exploration), or worse: completely independent of system knowledge altogether (e.g. ε-greedy exploration) discussed in § 4.1. By contrast we argued *directed* exploration is preferable, exploring according to the robot's system uncertainties in § 4.2. Exploration is much more data efficient if the robot specifically targets its *known unknowns* of the system. Reducing uncertainties in the dynamics is certainly preferable to undirected exploration, yet only really matters if it results in a reduction in loss uncertainty – the only objective we care about. By focusing on reducing dynamics uncertainties only, the robot is susceptible to situations where much can be learned about a dynamical system – none of which improves the loss function. Whilst developing a more certain dynamics model is interesting, the *value* of that information could be zero. Thus, we would rather focus our dynamics learning on what helps the robot optimise the loss function and reduce loss uncertainty. We characterised how to balance minimising expected loss against loss uncertainty by a reduction in *total* loss: the cumulative loss across *all* episodes (4.3).

To compute cumulative-cost uncertainties, we used PILCO's probabilistic dynamics model. First we computed the a Gaussian approximation of the full joint distribution over states $X_0, \ldots, X_T$ in § 4.4.1, from which we computed the full joint distribution over costs at each points in time (§ 4.4.1), approximated as Gaussian. Doing so gave us the ability to compute both the cumulative-cost-mean (as the original PILCO did), but additionally the cumulative-cost-variance (part of our extension of PILCO). Using the uncertainty over the cumulative cost, we devised an exploration strategy in § 4.4, balancing exploitation (via cumulative-cost-mean) with exploration (via cumulative-cost-variance) using Bayesian Optimisation (BO).

We investigated four BO algorithms in § 4.3 to balance exploration and exploitation, including a simplified version of upper confidence bounds (UCB), probability

of improvement (PI), expected improvement (EI), and Gittins index (GI). 'Bayes optimal' control would be preferable to using BO, since Bayes optimal control optimises the total loss (summed over all episodes) w.r.t. the robot's belief function and number of episodes remaining. However, Bayes optimal control is intractable in general. Thus, we investigated the use of myopic and tractable BO methods for exploration instead. Both PI and EI are common exploration strategies in the BO community. By contrast, GI originates from the bandits literature and is perhaps less familiar to the BO community. Nevertheless GI is an important BO algorithm, a principled method that, under certain conditions, yields Bayes-optimal control. Being a Bayes-optimal controller, Gittins index considers discounting factors according to a deep lookahead. By contrast, both PI and EI are myopic heuristics and do not consider discounting factors (which we approximately transformed into the number of episodes remaining). As such, when experimenting with Gaussian bandits in § 4.4.3, the GI heuristic outperformed both PI and EI. The GI is only Bayes optimal under the following assumptions: 1) the horizon is infinite, 2) cumulative costs between controllers are independent, 3) a controller's cumulative-cost-variance reduces to zero after the next episode's execution, 4) the search space over controller parameterisations is convex. All the above assumptions are violated in the case of PILCO, yet the consequences of violating these assumptions are not necessarily drastic. As such our approximate lower bound of the true GI still performed strongly. For future work, we could satisfy the first assumption by simply changing our experimental tasks to minimising total losses over infinite horizons instead of finite. In regards to the fourth assumption, most control algorithms settle for locally optimal controllers, as does PILCO, since the optimisation surface for nonlinear control is non-convex in general. Apart from BO, a wider variety of Bayesian reinforcement Learning (BRL) algorithms were also discussed. For a recent survey for BRL algorithms see Ghavamzadeh et al. (2016).

Our experimental results on the cart double-pole swing-up task in § 4.4.3 suggested that exploration is an important addition to improving PILCO's data efficiency. Although, each experiment could take several hours to conduct, so only 20 experiments per seven algorithms were conducted. Future work could redo such experiments many more times before statistically significant claims can be made about the benefit of EI or GI exploration over the purely-exploitative PILCO. However, § 4.4.3 did indicate that both EI and GI are probably a benefit to PILCO's data efficiency. Interestingly we saw that the PI strategy performs worse than PILCO. PI only gauges the probability of improvement a controller has of previously tested controllers, not how much better the controller might be. Consequently, PI is known

to suffer very local search (even more local than PILCO): only searching in very close proximity to what was tested before (i.e. very similar controller parameterisation). For future work, we would consider a higher-dimensional control task such as the unicycle used in PILCO with 10 state dimensions. More complex control tasks would require more episodes to learn control, hopefully separating the performances of each exploration strategy more, since the value of exploration is greater with more episodes in which to learn.

Several questions remain for future work. The most important is whether the method for distinguishing epistemic uncertainty from aleatoric in the cumulative-cost in § 4.5 is accurate or effective, which would require experimental validation. Other improvements to § 4.5 could also be made to improve an estimate of the epistemic uncertainty including:

- Using a full rank-T update of the gram matrix K considering the next dataset $\mathcal{D}_{e+1}$ will contain T many transition data. Currently we only conduct rank-1 updates at each timestep, which neglects how all T new datapoints effects the prediction of a single state $p(X_t)$.
- Considering how the *joint*-effects on how new data effects predictions $p(X_t)$ and $p(X_{t+1})$, not each in isolation.
- Complete our derivation on the recursive nature of predictive changes. I.e. changes to $p(X_t)$ will effect $p(X_{t+1})$ directly, since $p(X_t)$ causes $p(X_{t+1})$.
- Consider fantasy input uncertainty
- Derive the *hierarchical* cost-cost joint representation.

Other future work follows. First, much correlation exists between controllers nearby in controller-parameters space, and executing one controller informs us about most other controllers. Currently we have no way to evaluate the information gained about any other controller than the controller we intend to execute. Whilst our assumption of independence is not ideal, it is only made for tractability. Second, our BRL extension of PILCO is myopic. By only considering the next episode's potential data $\mathcal{D}_{e+1}$, and not $\mathcal{D}_{e+2}$, we are using a myopic belief lookahead RL algorithm. We could potentially deepen our lookahead at the expense of additional computation, although it is unclear how tractable this would be.

Chapter 5

Data Efficient Deep Reinforcement Learning

All work in Chapter 5 was done in an equal-effort collaboration with Yarin Gal. This final research-based chapter investigates data efficiency in an emerging subfield of reinforcement learning: *deep* reinforcement learning. Deep learning has previously proven useful in supervised learning tasks, adept at learning rich features to extract from high dimensional inputs. Recently, interest has increased for applying deep methods to reinforcement learning tasks too, due to an ability to train controllers with high dimensional observations, such as pixels. To date, most deep RL methods are *model free*: using deep architectures for the action value function (Q function) or policy.

Unfortunately, model free methods (deep or not) suffer from data inefficiency, explained in § 2.3.1. As a result, many state-of-the-art deep RL methods (being model free) require thousands of trials, equivalent to hundreds of thousands of observed transitions, to learn simple tasks, even when the observation dimensionality is low (Lillicrap et al., 2015). When real-world trials involve real cost, time and resources, data efficiency is critical.

Data efficiency is improved if we model the dynamics – termed model based RL – discussed in § 2.4. Model based methods discover better controllers with fewer data since dynamics models can better generalise the system dynamics knowledge gained to unobserved states. Probabilistic modelling especially offered even greater data efficiency under dynamics uncertainty (typical of low-data regimes, when data-efficiency is a concern), by considering dynamics uncertainty throughout planning and prediction. PILCO for instance (§ 2.5) used non-deep probabilistic Gaussian Process (GP) models to achieve state-of-the-art data efficiency on low dimensional

control tasks including the cart-pole. PILCO learns the cart-pole swing-up task in just 6-7 episodes (not thousands) by analytically propagating uncertain state distributions through a GP dynamics model. This allows the robot to consider the longterm consequences (loss) of a particular controller parameterisation w.r.t. all plausible dynamics models. We saw dynamics uncertainty play a crucial role in PILCO, with MAP estimation shown to fail in Chapter 3.

Recent work on model-based deep RL still rely on large amounts of data for unsupervised training of auto-encoders (Stadie et al., 2015; Wahlström et al., 2015), before a controller is learned. Work by Wahlström et al. (2015) in particular used a method similar to PILCO, but without considering model uncertainty. Instead, (deterministic) model predictive control (MPC) simulated state trajectories, leaving the approach susceptible to the problem of model bias.

In this chapter, we instead propose a *probabilistic* deep model-based RL to achieve data efficient deep RL. We achieve much greater data efficiency than other state-of-the-art deep RL methods. Our framework is similar to PILCO, replacing PILCO's probabilistic GP model with a Bayesian probabilistic Neural Network (BNN). Whilst we have not applied our system to tasks with high dimensional observations, we show Bayesian deep RL is much more data efficient in low dimensions using the cart-pole swing-up task. Conceivably the same holds true for higher dimensional observation tasks, which is why we are interested in deep methods to begin with, but this is left for future work.

5.1 DEEP PILCO

We now describe our method – *Deep* PILCO – for data efficient deep RL. Our method is similar to PILCO: both methods follow Algorithm 1. The main difference of Deep PILCO is its dynamics model. PILCO uses a GP which can model the dynamics' output uncertainty, does not scale to high dimensional inputs or size of training data (see Table 2.1). In contrast, Deep PILCO uses a deep neural network, capable of scaling to with data (only if required by the complexity of the task – noting our goal is still data efficiency) and potentially capable of scaling to high dimensional observations spaces. Like PILCO, our policy-search algorithm alternates between fitting a dynamics model to observed transition data, evaluating the controller using dynamics model predictions of future states and costs, and then improving the controller.

Replacing PILCO's GP with a deep network is not trivial, since we wish our dynamics model to maintain its probabilistic nature, capturing output uncertainty and also input uncertainty.

5.1.1 OUTPUT UNCERTAINTY

First, we require output uncertainty from our dynamics model, critical to PILCO's data efficiency. Typically, NNs are trained by optimising deterministic weights to minimise some loss function. Such non-probabilistic inference methods do not capture subjective uncertainty in system dynamics to yield output model uncertainty. Whilst NN models are clearly powerful, such inference method limit their applicability, especially in low data settings. Instead, we use Bayesian (probabilistic) Neural Networks (BNN). BNNs represent model uncertainty with the use of a posterior distribution over the weights of the NN (MacKay, 1992).

Unfortunately, the true posterior of a BNN is intractably complex. However, several approximations exist. Hamiltonian Monte Carlo is perferred for low-dimensional NNs, although not effective in high dimensions (Gal, 2016b, chapter 2). Whilst we only experiment with a low-dimensional system in this chapter, our intention is to work towards a methods capable of high-dimensional deep RL. Variational autoencoders provide another approximate solution, but require much data, contravening our data efficiency objective. Variational inference finds a distribution from a tractable family that minimises the Kullback-Leibler (KL) divergence to the true posterior. Another variational approximation we instead use is from Gal and Ghahramani (2015). Gal and Ghahramani (2015) show that dropout can be interpreted as a variational Bayesian approximation, where the approximating distribution is a mixture of two Gaussians with small variances and the mean of one of the Gaussians fixed at zero. The uncertainty in the weights induces prediction uncertainty by marginalising over the approximate posterior using Monte Carlo integration. This amounts to the regular dropout procedure only with dropout also applied at test time, giving us output uncertainty from our dynamics model.

This approach also offers insights into the use of NNs with small data. Gal and Ghahramani (2015) show that the network's weight decay can be parameterised as a function of dataset size, dropout probability, and observation noise. Together with adaptive learning-rate optimisation techniques, the number of parameters requiring tuning becomes negligible.

5.1.2 INPUT UNCERTAINTY

A second difficulty with NN dynamics models is handling *input* uncertainty. To plan under dynamics uncertainty, PILCO analytically propagates state distributions approximately through the dynamics model (Algorithm 1, line 5). To do so, the dynamics model must pass uncertain dynamics outputs at time t as the uncertain input into the dynamics model at time $t+1$. As far as theauthor is aware, no analytical method is known to approximately transform input uncertainty through a NN that closely resembles the true (intractible) transformed distribution. To feed a distribution into the dynamics model, we thus resort to particle methods (Algorithm 2). This involves sampling a set of particles from the input distribution (step 1 in Algorithm 2), passing these particles through the BNN dynamics model (and sampling the uncertain output, step 7 in Algorithm 2), which yields an output distribution of particles.

Algorithm 2 Step 5 of Algorithm 1: *Simulate* system trajectories from $p(X_0)$ to $p(X_T)$

1: *Initialise* set of P particles $x_0^p \overset{iid}{\sim} p(X_0)$.
2: **for** $p = 1$ to P **do**
3: Sample BNN dynamics model weights W^p.
4: **end for**
5: **for** time $t = 0$ to T **do**
6: **for** each particle x_t^1 to x_t^{P} **do**
7: Evaluate BNN with weights W^p and input particle $\tilde{x}_t^p$, obtain output z_t^p.
8: **end for**
9: Calculate mean μ_t and variance Σ_t of $\{z_t^1, ..., z_t^{\mathrm{P}}\}$.
10: Sample set of P particles $x_{t+1}^p \sim \mathcal{N}(\mu_t, \Sigma_t)$.
11: **end for**

A similar approach using a particle filter for dynamics model training was attempted unsuccessfully in the past with PILCO. McHutchon (2014, section 3.7) encountered several problems optimising hyperparameters of a model with particle methods, the main problem being the abundance of local optima in the optimisation surface, impeding their BFGS optimisation method, which PILCO also uses to optimise the controller. McHutchon (2014) suggested that this may be due to the finite number of particles P used and their deterministic optimisation. To avoid these issues when optimising the controller, we randomly re-sample a new set of particles at each optimisation step, giving us an unbiased estimator for the objective (Algorithm 1, line 6). In conjunction we use a stochastic optimisation procedure Adam (Kingma and Ba, 2014) instead of BFGS.

We found that fitting a Gaussian distribution to the output state distribution at each timestep, as PILCO does, is of crucial importance (steps 9-10 in Algorithm 2). Moment matching avoids multi-modalities in predictive state distributions. Fitting a multi-modal distribution with a (wide) Gaussian causes the cost function to span many high-cost states, often resulting in a larger expected cost for the unimodal fitted distribution than the original multi-model distribution (Deisenroth et al., 2015). By forcing a unimodal fit, the algorithm penalises controllers that cause the predictive states to bifurcate, often a precursor to a loss of control. In this sense, the Gaussian fit is a form of reward-shaping (Ng et al., 1999) where the cost function altered in a way that improves learning speed. Using a smooth unimodal fit (Gaussian) to the dynamics outputs, combined with our smooth cost function, results in smooth loss gradients (reduced local optima and discontinuities), helpful for controller optimisation via gradient decent (this is explained further, with examples, in the experiments section). We hypothesised this to be an important modelling choice used by PILCO and assessed this assumption in our experiments.

5.1.3 SAMPLING FUNCTIONS FROM THE DYNAMICS MODEL

Unlike the original PILCO's GP model, Deep PILCO can sample individual plausible dynamics functions from our probabilistic dynamics model and following a single function throughout an entire episode. Two advantages of sampling from a probabilistic model are 1) a directed exploration strategy is easily implemented using Thompson sampling (discussed § 4.2.1), and 2) to consider temporal correlation in model uncertainty. Temporal correlations reflect that – whilst the robot is (epistemically) uncertain about the dynamics function – whatever the true dynamics function is, it is consistent accross time! The true function is not 're-sampled' from the robot's uncertainty at each point in time. PILCO does not consider such temporal correlation in model uncertainty between successive state transitions, but rather treats each timestep as independence. This independence probably results in PILCO underestimating state uncertainty at future timesteps (Deisenroth et al., 2015). Fig. 5.3 (explained in detail below) shows episodes obtained with a fixed controller and sampled dynamics model functions for the cart-pole swing-up task. They are generated by sampling particles from the initial distribution, and sampling and fixing a dropout mask throughout the episode for each particle.

5.2 EXPERIMENTS

This section describes the experimental setup we used to compare our Deep-PILCO algorithm against the original PILCO, using the cart-pole swing-up task as a benchmark. We follow with an analysis of the experiment results in § 5.3. The cart-pole swing-up task (Fig. D.1 on page 177) is a standard benchmark for nonlinear control due to the nonlinearity in the dynamics, and the requirement for nonlinear controllers to successfully swing up and balance the pendulum.

The cart-pole swing-up is a continuous state, continuous controls, discrete time task. In Chapter 3 we used the cart-pole with significant observation noise, however, here we will now assume negligible observation noise. Our goal is again to balance the pendulum upright. The system state x comprises the cart position, pendulum angle, and their time derivatives $x = [x_c, \theta, \dot{x}_c, \dot{\theta}]^\intercal$. Task parameters (summarised Table D.2) used are pendulum length $l = 0.6$m, cart mass $m_c = 0.5$kg, pendulum mass $m_p = 0.5$kg, time horizon $T = 2.5$s, time discretisation $\Delta t = 0.1$s, and acceleration due to gravity $g = 9.82\text{m/s}^2$. In addition, friction resists the cart's motion with a damping coefficient $b = 0.1\text{Ns/m}$. The cart-pole's motion is described with the differential equation:

$$\dot{x} = \left[\dot{x}_c, \dot{\theta}, \frac{-2m_p l\dot{\theta}^2 s + 3m_p g s c + 4u - 4b\dot{x}_c}{4(m_c + m_p) - 3m_p c^2}, \frac{-3m_p l\dot{\theta}^2 s c + 6(m_c + m_p)gs + 6(u - b\dot{x}_c)c}{4l(m_c + m_p) - 3m_p l c^2}\right],$$

using shorthand $s = \sin\theta$ and $c = \cos\theta$. Both the initial latent state and initial belief are assumed to be iid: $X_0 \overset{iid}{\sim} \mathcal{N}\left(\mu_0^x, \Sigma_0^x\right)$ where $\mu_0^x \sim \delta([0\text{m}, \pi\text{rad}, 0\text{m/s}, 0\text{rad/s}]^\intercal)$ and $(\Sigma_0^x)^{\frac{1}{2}} = \text{diag}([0.2\text{m}, 0.2\text{rad}, 0.2\text{m/s}, 0.2\text{rad/s}])$.

We use a saturating cost function as PILCO did: $1 - \exp\left(-\frac{1}{2}d^2/\lambda_c^2\right)$ where $\lambda_c = 0.25$m and d^2 is the squared Euclidean distance between the pendulum's end point (x_p, y_p) and its goal $(0, l)$.

For our deep dynamics model we experimented with dropout probabilities $p_{\text{dropout}} = 0$, 0.05, 0.1, and 0.2. We found that $p_{\text{dropout}} = 0.05$ performed best and used this in our comparison to PILCO. As per Algorithm 1 we alternate between fitting a dynamics model and optimising the controller. To fit the dynamics model we use 5×10^3 optimisation steps, each step using 100 particles (batch size). To optimise the controller we use 10^3 steps, each step with batch size of 10. Weight decay of the NN dynamics model is set to 10^{-4}. The dynamics model architecture has 200 units with 2 hidden layers and sigmoid activation functions. Our controller

is a RBF network with R = 50 units. Like Lillicrap et al. (2015), we use a "replay buffer" of finite size (the most recent 10 episodes), discarding older episodes of data.

As per Algorithm 1 line 2, we first generate a single random episode when experimenting with each method, before iterating over Algorithm 1's main loop for 100 episodes (40 for PILCO due to its time complexity). At each iteration a single episode of data is acquired by executing the cart-pole for 2.5s, generating 25 transition datum per episode. We evaluate each method (for visualisation, not optimisation) at each iteration by calculating the controller's cost averaged over 50 randomly sampled initial states from $p(X_0)$. Both PILCO and Deep-PILCO use the same physical simulator and cost function settings summarised Table D.2. Thus, the online time for each algorithm is 2.5 seconds per episode. Comparing offline time, PILCO averages 2.3 minutes per episode for the first 40 episodes using CPUs[1] whereas Deep-PILCO requires 8.0 minutes per episode on a GPU. Thus, Deep-PILCO is slower to converge than PILCO, yet our main goal is data efficiency,

Fig. 5.1 compares the average cost (y-axis) per episode (x-axis) of PILCO, Deep-PILCO, and other deep-RL algorithms to help visualise data efficiency. We can see PILCO converges after 7-8 episodes, whereas Deep-PILCO requires about 100 before reaching an average cost slightly worse than PILCO. However, Deep-PILCO does learn to achieve the task within 20-25 episodes only, not as data-efficicent as PILCO, but much more than competiting deep-RL algorithms shown by vertical bars in Fig. 5.1. We see in Fig. 5.2 the progression of deep PILCO's fitting as more data is collected. Slightly more than 20 episodes of data are required, to control the task. Between 10-20 episodes are required to model the dynamics well using a BNN.

Our model can be seen as a Bayesian approach to data efficient deep RL. We compare to recent deep RL algorithms (Lillicrap et al. (2015) and Gu et al. (2016)). Lillicrap et al. (2015) use an actor-critic model-free algorithm based on deterministic policy gradients. Gu et al. (2016) train a continuous version of model-free deep Q-learning using imagined episodes generated with a learnt model. For their *low dimensional* cart-pole swing-up task Lillicrap et al. (2015) require approximately 2.5×10^5 steps to achieve good results. This is equivalent to approximately 2.5×10^3 episodes of data, based on Figure 2 in Lillicrap et al. (2015) (note that Lillicrap et al. (2015) used time horizon 2s and time discretisation $\Delta t = 0.02$s, slightly different from ours; they also normalised their reward, which does not allow us to compare to their converged reward directly). Gu et al. (2016) require approximately 400 episodes for model convergence. These two results are marked with vertical lines in Fig. 5.1 (as

[1] PILCO is easily parallelised when training X GP models for each output dimension

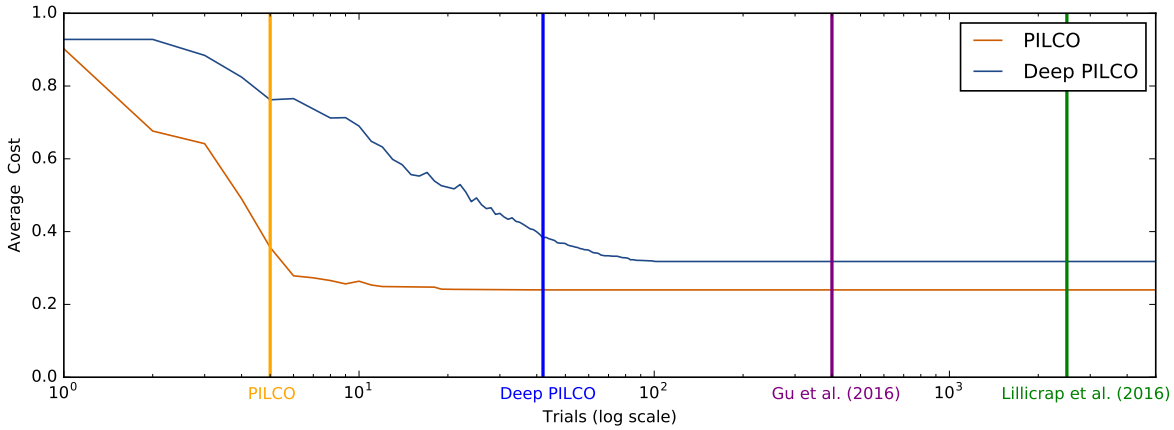
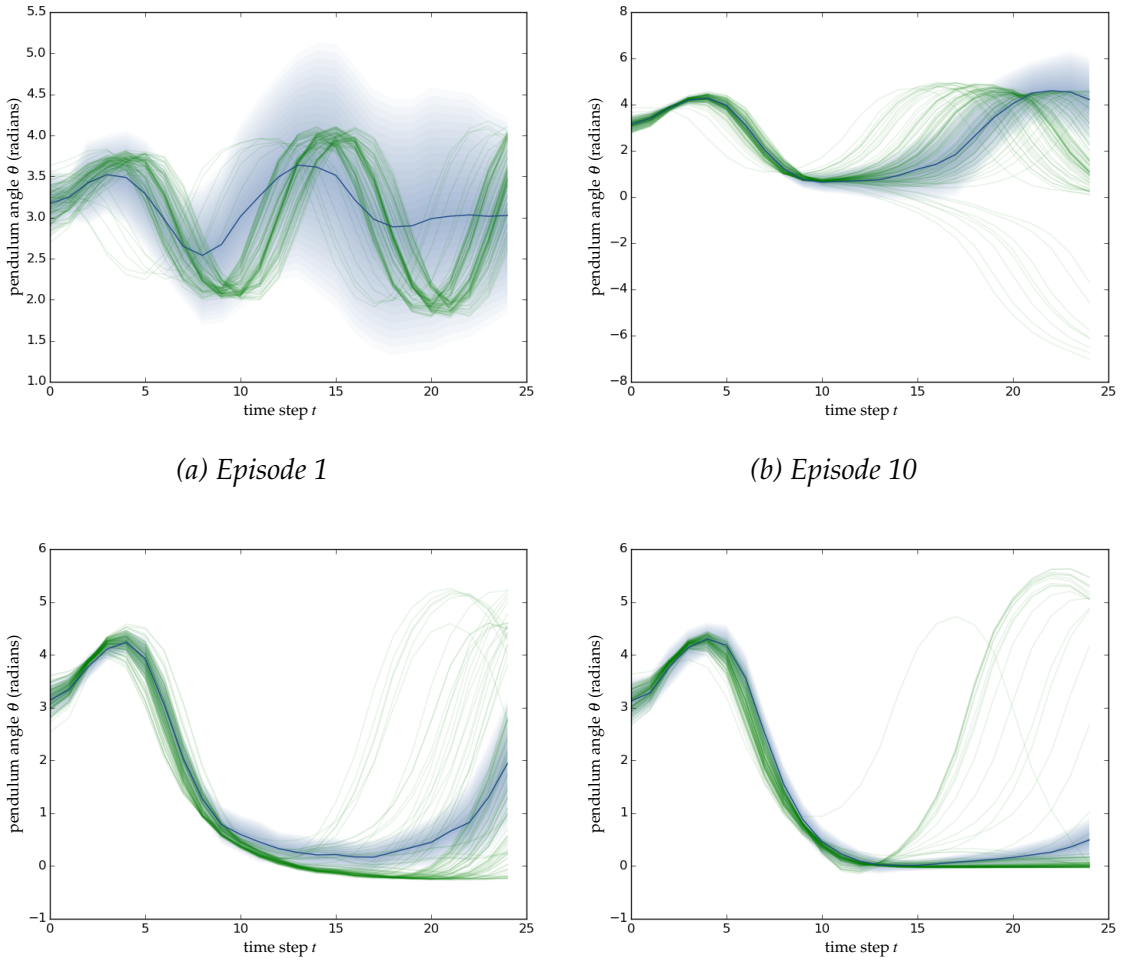


Fig. 5.1 ***Average costs per trial and convergence comparison on cart-pole swing-up task.*** *Shading shows one and two standard deviations of the average-costs' standard error given 40 experimental repeats for both* PILCO *and Deep-*PILCO*. Vertical lines show estimates of number of episodes required for model convergence (judged by successful balancing of the pendulum most experimental repeats) for* PILCO *(5 trials), Deep-*PILCO *(42 trials), Gu et al. (2016) (*$\sim$ *400 trials), and Lillicrap et al. (2015) (*$\sim 2,500$ *trials).*

the respective papers do not provide high resolution episode-cost plots) obtained from personal communication with the authors of both papers, after adjusting to their different time horizon and discretisation into account. Both Gu et al. (2016) and Lillicrap et al. (2015) used different cost functions, which is hard to compare with, but even if they reach lower costs using a common utility function, Fig. 5.1 shows their approaches require orders of magnitudes more episodes to solve the same task, and are much less data efficient deep RL-algorithms than Deep PILCO.

Lastly, we report model run time for both Deep PILCO as well as PILCO. Deep PILCO can leverage GPUs, and took an average of 5.3 hours total offline over 40 episodes. Time complexity for simulation (with gradient information) is technically constant w.r.t. the data set size N for our experiments, since the dynamics model is only trained on the 10 most recent episodes (so that the NN would give higher weighting to newer information more likely useful in training a controller than older data). However, as tasks become more complex the model will require more data, likely scaling linearly with N. In addition, time complexity is cubic w.r.t. input dimensionality X. Yet using the GPU and an ability to parallelise the operation in a NN, the cubic effect is unnoticed for $X < 100$. PILCO's simulation by contrast took 1.5 hours for $E = 40$ episodes running on the CPU, with time complexity: $\mathcal{O}(N^2X^2(X+U)^2)$ (see Table 2.1). With more episodes collecting more data PILCO slows down more and more between each episode. Consequently, PILCO is unsuited for tasks requiring a large number of episodes, nor high-dimensional state tasks.



(a) Episode 1 *(b) Episode 10*

(c) Episode 20 *(d) Episode 30*

Fig. 5.2 ***Progression of model fitting and controller optimisation as more episodes of data are collected.*** *We show the pendulum angle θ (see Fig. 3.6) per timestep to visualise whether the system is learning to balance the pole. The goal is to swing the pendulum up such that $mod(\theta, 2\pi) \approx 0$. The green lines are samples from our (latent) physical simulator of the cart-pole. The blue distribution is our Gaussian-fitted predictive distribution of states at each timestep.* ***(a)*** *After the first episode the model fit (blue) does not yet have enough data to accurately capture the true dynamics (green). Thus the controller performs poorly: the pendulum remains downwards swinging between 2π and 4π.* ***(b)*** *After 10 episodes, the model fit (blue) predicts very well for the fist 13 timesteps before separating from the true rollouts (green). The controller has stabilised the pendulum at 0π for about 10 timesteps (1 second).* ***(c)*** *After 20 episodes the model fit and controller are slightly improved.* ***(d)*** *From episode 30 onward, the dynamics model successfully captured the true dynamics and the controller successfully stabilises the pendulum upright at 0π radians most episodes.*

5.3 Results and Analysis

We next analyse the results from the previous section, providing insights based on the different setups we experimented with. First we note that Deep PILCO is more data inefficient than PILCO – but not by much. Deep-PILCO requires more data than PILCO and a 3.5 times more offline to train the dynamics model (a figure, which continually improves with more episodes than 40 since PILCO's dominant time scaling follows a quadratic relationship with the amount of data observed) and optimise a controller. One of PILCO's limiting factors at the moment is its lack of support of GPUs. We demonstrate this deficiency by giving GPU run time for our model, which is supported out of the box.

5.3.1 Dynamics Models of Finite Capacity

PILCO's dynamics model is a GP, a nonparametric model of infinite capacity, effectively able to memorise all observed transition data. As more data are observed, the model's flexibility increases to fit the additional data. NNs on the other hand are parametric with finite capacity, and must "smooth" over different data points when fitting to the data. This poses a difficulty in our setting since when fitting a NN model to a sequence of episodes, the same importance would be given to new episodes as old ones. But new episode's worth of data are much more important for the robot to learn, and with many old episodes a NN might not model the new ones well.

One possible solution would be to use a larger NN with each iteration, making inference slower as we get more data. Another solution is to downweigh older episodes, and use a weighted Euclidean loss (which can be seen as a naive form of "experience replay"). We experimented with an exponential decay, such that data from the oldest episode either had weight 0.01, 0.1 or 0.4. A disadvantage of this approach is slower optimisation. It requires several passes over the training dataset with most observations being inconsequential to the optimisation and did not work well in practice for any decay rate. We suspect this to be due to the finite capacity of the NN, where the function complexity keeps increasing but the model complexity does not.

Our solution was instead to keep the last 10 episodes of data only, similar to Lillicrap et al. (2015)'s "replay buffer". A disadvantage of this approach is that the robot 'forgets' bad episodes, and will continuously attempt to explore these even with a good controller. The effect is that – even after 50 episodes – the controller

would fail to stabilise the pole every 4th or 5th episode thereafter. Such behaviour degrades Deep PILCO's performance, and one of the main factors explaining its inferior performance to PILCO (Fig. 5.1). A better solution, that retains all training data is no doubt possible, left for future work.

5.3.2 PARTICLE METHODS AND MOMENT MATCHING

In our method we moment-matched the output distribution at each timestep before propagating it to the next timestep. This forces the state distribution to be unimodal, avoiding bifurcations. We hypothesised in § 5.1 that this is an important modelling choice in PILCO. To assess this hypothesis, we experimented with an identical experiment setup to the above, but without moment matching. Instead, we pass the particles unchanged from the output of one timestep to the next.

Fig. 5.3 shows the dynamics model fit (in a similar plot to Fig. 5.2). The robot is able to swing the pendulum up and balance the pendulum inverted for 0.5s, but then loses control and has to swing it up again. Notice the state trajectories bifurcating at the origin, with half of the states swinging clockwise, and the other half counter-clockwise. The controller's parameterisation in Fig. 5.3 is at a local optimum (improving the cost of clockwise trajectories can be at the expense of counter-clockwise trajectories), with the controller optimisation procedure unable to escape this bifurcating behaviour after 100 episodes (regardless of whether the 10 episode memory restriction is in place or not).

Imposing a moment matching modelling assumption, the robot incurs much higher cost by averaging over a Gaussian fit at time $t = 9$ in Fig. 5.3 than by averaging over the true bimodal distribution at time $t = 9$, whose values of pendulum angle θ are 0π and 2π, both of which indicate an inverted pendulum near its goal, each incurring little cost. Instead of having half of the trajectories at 2π and the other half at 0π (resulting in overall low cost), a Gaussian fit to these trajectories placing most probability mass in the centre at $\theta = \pi$ which entails large cost, even though we see there is very little probability the pendulum will have $\theta = \pi$ at $t = 9$. In addition, the state variance is predicted to be large, which in this case causes overestimation of the predicted cost variance (by covering low cost $\theta = 0\pi, 2\pi$, and high cost $\theta = \pi$) interfering with directed exploration strategies discussed Chapter 4. By sampling new particles from a Gaussian fitted distribution at $t = 9$, almost all particles have high cost. For timesteps after such bifurcation with unimodal fitting and saturating cost function, then the Gaussian is stretched so wide that state distributions have small *partial* derivatives w.r.t. the controller parameters, however have large *full*

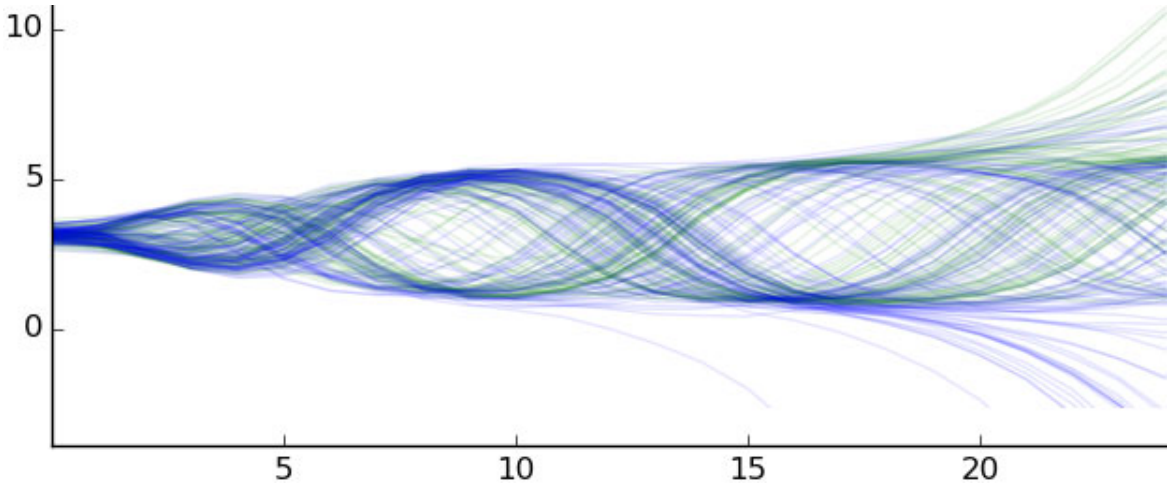
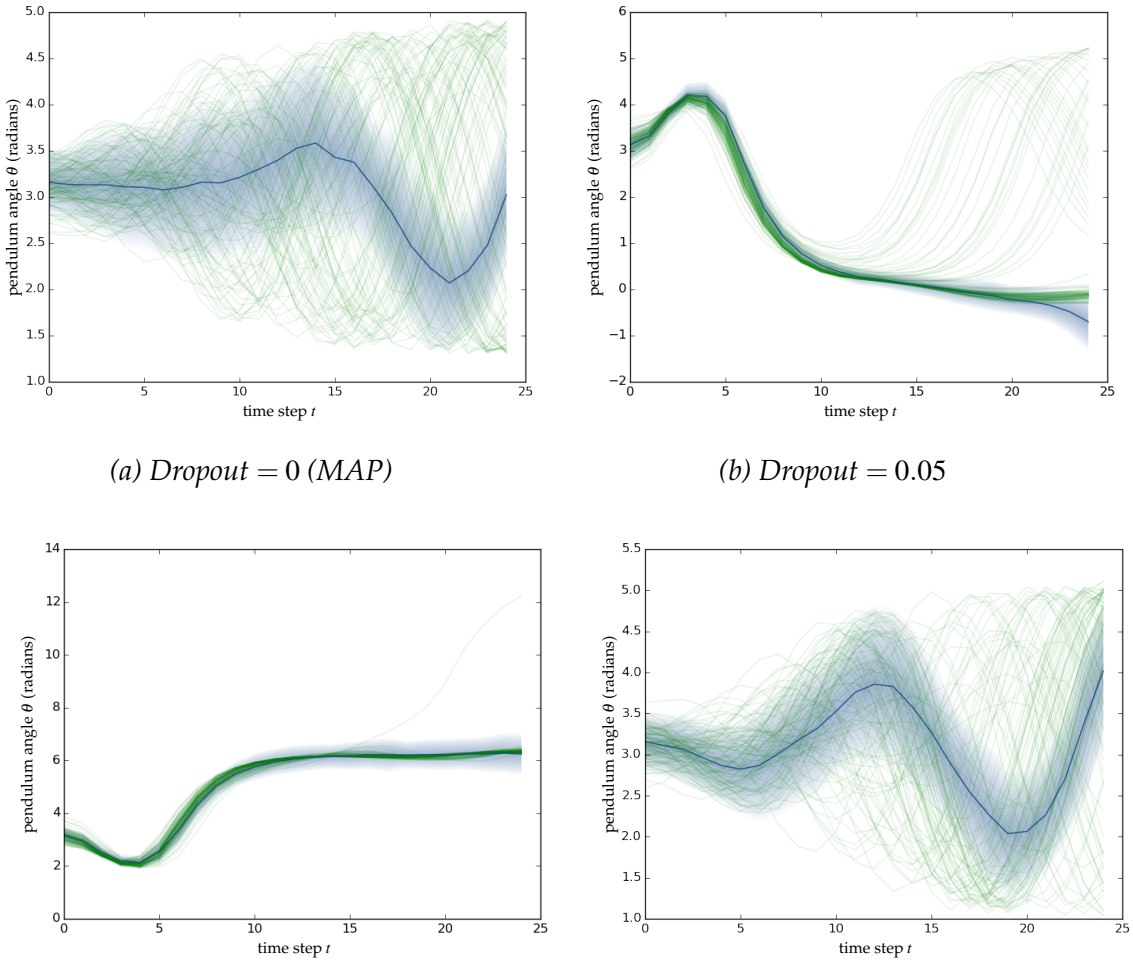


Fig. 5.3 ***Pendulum angle θ (from Fig. 3.6) as a function of timesteps t.*** *Each trajectory corresponds to a single particle sampled from the initial distribution at $t = 0$. A controller is optimised using Deep* PILCO *and fixed. Blue trajectories are pendulum angle following the learnt dynamics model with the fixed controller, and green trajectories are pendulum angle following the system (true) dynamics with the same controller. Each blue trajectory follows a single sampled function from the dynamics model (applying the same dropout mask at each timestep). The learnt dynamics model matches the system dynamics, and the particle distribution successfully captures the multi-modal predictive distribution of future states. However, without moment matching we could not optimise a good controller from these multi-modal distributions.*

derivatives w.r.t. the controller parameters due to the chain rule: whatever happened in previous timesteps that led up the bifurcation. This directs controller optimisation towards robot behaviours of low cost that do not bifurcate.

5.3.3 IMPORTANCE OF UNCERTAINTY IN DYNAMICS MODELLING

We assessed the importance of a probabilistic dynamics model in our setting. This can easily be done by setting the dropout probability to $p_{\text{dropout}} = 0$, which results in a MAP estimate. As can be seen in Fig. 5.4a, even after 75 episodes the dynamics model did not converge to anything sensible. This suggests that input uncertainty, originating solely from the initial state variance Σ_0^x is insufficient, and propagating the input distribution with a MAP estimate will not necessarily avoid model bias. Similar behaviour is observed when the dropout probability is too large (Fig. 5.4d) presumably as the model underfits. Dropout probabilities of 0.05 and 0.1 seem to work best (Fig. 5.4b and Fig. 5.4c). Note though that suitable dropout probabilities are dependent on model size. Larger NNs may necessitate larger dropout probabilities to adequately capture output uncertainty.



(a) Dropout $= 0$ *(MAP)*

(b) Dropout $= 0.05$

(c) Dropout $= 0.1$

(d) Dropout $= 0.2$

Fig. 5.4 ***Effects of dropout probabilities on the dynamics model after 75 episodes.*** *Each x-axis is timestep t, and each y-axis is pendulum angle* θ*. MAP estimate fails to capture the dynamics as it offers no probabilistic output (the depicted uncertainty is that of the propagated input distribution). Too high dropout probability (0.2) does not allow the model to learn the system dynamics.*

5.4 Discussion and Future Work

In this chapter we laid some foundations to apply the PILCO framework to higher state-dimensional control tasks. The time complexity of learning to control with PILCO with GP prediction (with gradients) scales $\mathcal{O}(X^4)$ and difficult to parallelise. Our new algorithm presented this chapter, Deep PILCO, instead uses Bayesian Neural Networks (BNN), scaling only $\mathcal{O}(X^3)$ and it is easily parallelised. Although our experiments in § 5.2 were still low dimensional, we have made a step towards high-dimensional learning of control with PILCO by replacing PILCO's GP dynamics model with a BNN, a nontrivial task. Neural networks are commonly used for image processing, with pixel inputs of thousands of dimensions. BNNs are not typically used for such tasks which is why we opted for a scalable BNN inference using Gal and Ghahramani (2015). A remaining challenge for high-dimensional PILCO is to be able to compress a high dimensional input space into a much smaller latent space, which a dynamics model can use for forwards simulation, propagating uncertain state inputs and producing uncertain state outputs for the purpose of controller evaluation. We plan to build on these foundations to scale the model to high X in future work.

We demonstrated that BNNs can mitigate model bias when learning control with a small amount of data by following PILCO's framework. PILCO's framework used probabilistic dynamics models, using probability theory as a principled method for handling subjective uncertainties a robot has over the set of plausible dynamics functions that might exist. In addition, we compared Deep PILCO against three similar algorithms by averaging the performance over multiple runs of a 100 episode cart-pole swing-up task, Fig. 5.1. However, Deep-PILCO discovered converged to higher-cost controllers on average, and required 3.5 times the offline processing time, and required 8-9 times more data to invert the pendulum in the cart-pole task that PILCO required. Thus whilst PILCO outperformed Deep PILCO in each of the above three metrics on the cart-pole setup, we hope successful utilisation of a scalable BNN dynamics model will be scalable to pixels in future work, which PILCOcannot scale to.

By replacing PILCO's GP model with a BNN, several choice for BNN parameters existed which required tuning. In this chapter, we investigated 1) automatic discovery of how much data to retain in our 'replay buffer', 2) how the model fit improves per episode by projecting the simulation of state trajectories along only a single dimension, the pendulum angle (Fig. 5.2), 3) analysed particle methods

vs moment-matching (Fig. 5.3), and 4) compared dropout parameter values and inference settings (Fig. 5.4).

An exciting set of options also exist for future work:

1. **Observation noise**: We can easily extend the dynamics model to consider observation noise – even heteroscedastic observation noise (Gal, 2016a). For example, heteroscedastic observation noise exists in using a camera to observe the position of moving objects. The speed of the pendulum increases uncertainty in observing the pendulum's position due to blurring of the image. Capturing observation noise can be challenging with the standard PILCO framework, but would be a simple extension in ours.

2. **Exploration**: We can increase data efficiency further by updating Deep PILCO using directed-exploration via Thompson sampling. A NN is easy to sample from by probabilistically drawing a random dropout mask, which specifies which nodes are 'on' and 'off' in proportion to p_{dropout}. The controller could be optimised w.r.t. the sampled dynamics model, and executed for one episode, before sampling another NN from the BNN for the following episode.

3. **Higher dimensions**: Several compression techniques remain available to learn in high dimensional spaces (e.g. pixels) by compressing down to low dimensional state representations. Ideally one can avoid the use of data inefficient autoencoders which are unsupervised w.r.t. the cost function, e.g. Wahlström et al. (2015). Better would be to use a supervised compression, supervised in the sense that the robot can still predict future costs, required for controller optimisation.

4. **Faster controller optimisation**: Gradient smoothness can be increased to better facilitate the controller optimisation process, smoothing the particles at each time-point with Gaussian bumps.

5. **Multi-modality**: Ideally Deep PILCO should handle multi-modal distributions. Unfortunately, we could not successfully train a cart-pole swing-up controller without moment-matching at each timestep.

Chapter 6

Conclusions and Outlook

This thesis addressed several avenues to autonomously learn control of nonlinear systems *efficiently*. By 'efficient' we mean data efficient: minimal system interaction. Data efficiency is a goal of this thesis, critical when systems are slow, expensive to operate, or prone to wear and tear. A previous benchmark of data efficient learning of control was PILCO by Deisenroth and Rasmussen (2011). Until now, PILCO had been an unbeaten benchmark of data efficiency, specifically of autonomously learning black-box, nonlinear, time-invariant, stochastic, continuous-state-action dynamical systems with real-time episodic tasks. We extended the PILCO framework in three principle ways: 1) evaluating controllers w.r.t. to a filtering process to learn effectively under significant observation noise (Chapter 3); 2) balancing exploration and exploitation to achieve greater data efficiency than PILCO (Chapter 4); and 3) using Bayesian Neural Network (BNN) dynamics models efficiently (Chapter 5), to develop a framework for data efficient deep Reinforcement Learning (RL) for high dimensional applications. By contrast, the original PILCO did not filter, was purely-exploitative, and limited to Gaussian Process (GP) dynamics models only.

To successfully extend PILCO in various ways, we needed to retain what made PILCO so successful in the first place. It should go without saying, that data efficient methods must be able to learn effectively in small-data regimes. Yet small-data regimes are notorious for causing highly flexible models to overfit. The first tenet to PILCO's success we used was the Bayesian probabilistic dynamics modelling. Bayesian models are critical to overcoming issues of overfitting, by considering a distribution of plausible dynamics model rather than a single model throughout prediction, simulation, and controller evaluation. Probabilistic prediction helps controller evaluation to focus on what will *probably* improve the controller (according to the set of plausible models). Deterministic prediction by contrast has no ability

to distinguish probable-vs-improbable improvements, instead trusting a single dynamics model with *arbitrary* values when far from data, easily leading controller optimisation astray. Bayesian nonparametric models in particular resist overfitting and underfitting the data, and thus we continued to use a GP dynamics model in most chapters. One exception was Chapter 5, where we instead use a BNN. Being Bayesian, our BNN resisted overfitting, however, being parametric we did need to select the number of nodes carefully. Probabilistic simulations required predictions of uncertain inputs, which we approximated using MC sampling with our BNN.

A second tenet behind PILCO we continued to follow was: simulations should be faithful to reality. For accurate controller evaluation, the entire robot should act in simulation exactly as it acts in reality. For example, PILCO simulated the probabilistic process of closed-loop control, precisely because it executes closed-loop control. Similarly, we faithfully simulated closed-loop *filtered* control, precisely because we execute closed-loop filtered control. By doing so, we optimised a controller for the exact context in which it is used and consequently outperformed other methods which did not, § 3.3.6.

A third tenet is dual control. To learn to control in a data efficient way, a robot must have a chance to learn *actively*: to be able to influence which data it collects and learns from next. Many approaches to control first consider system identification (dynamics modelling), and only after system identification is complete is a controller designed. Data-efficient learning requires both system identification and control design be done simultaneously. PILCO iterates between them, seen Algorithm 1, which we continued to do.

A forth tenet was the analytic framework for simulating control using Gaussian-fitted distributions for tractability. An analytic framework has the advantage of smooth and analytically computable gradient information useful for controller optimisation. An advantage of also using GPs is that output moments are analytically and exactly computable, given Gaussian distributions as input and squared exponential covariance functions. We continued to use an analytic framework to tractably simulate control, retaining PILCO's *linear* time-complexity w.r.t. time horizon, not exponential as many BRL methods have. We also continued to use bounded cost functions, not quadratic, to avoid issues of cost of *arbitrary*-magnitude, which can dominate the controller optimisation process. Following the four tenets above resulted in our successful development of various data efficient algorithm. By comparison, for the problems we tested, many other algorithms will find near-optimal

controllers eventually, but require much data to do so. We conclude with a summary contributions and an outlook of interesting avenues to extend this research.

6.1 Summary of Contributions

Six novel contributions this thesis has made to data efficient automatic learning of control are:

1. A probabilistic framework to simulate a process of filtered control for, for accurate controller evaluation under significant observation noise (§ 3.3).
2. Gaussian process prediction derivations given *hierarchically*-uncertain inputs, for an analytic framework of distributions over Gaussian-belief-distributions (Chapter 3, Appendix B.4).
3. A Markov process, *latent-variable belief MDP*, which is more general than POMDPs being able to predict consequences of incorrect prior beliefs and to accurately evaluate a filtered process which must rely on inaccurate, real-time dynamics models (§ 3.4).
4. Compute the predictive cumulative-cost *distribution* of a controller, used for exploration (§ 4.4).
5. A new Bayesian Optimisation (BO) technique: an approximate lower-bound of Gittins index for Gaussian bandits (§ 4.3.4).
6. Developed a framework for deep Bayesian RL for data efficient control with high-dimensional states (Chapter 5).

6.2 Outlook

Here we discuss further interesting avenues of research, what could be done next to extend the research of this thesis. We discuss some big ideas for improvements first, and progress to smaller, more specific ideas.

Our experiments with a filtered version of PILCO in § 3.3 demonstrated how faithful simulators benefit controller performance, since controllers thus designed for the specific circumstance in which they are then used. A deficiency of our simulation in § 3.3 is that our uncertain belief-states $B \sim \mathcal{N}(M, V)$ considers distributions on the

belief-mean M but not the belief-variance V. A more accurate simulation would be to also randomise the variance V matrix too, to analytically simulate distributions of beliefs states with varying degrees of certainty. Such an approach may be possible using Wishart distributions over PSD variance matrices.

We showed that exploration and exploitation can improve data efficiency of PILCO using cumulative-cost distributions and BO exploration heuristics in § 4.4. However, an important extension yet to be completed is distinguishing aleatoric and epistemic uncertainties. Aleatoric uncertainty, represents inherent random system noises, which contribute to cumulative-cost variance, yet offer no exploratory value (since their randomness is persistent and irreducible). Epistemic uncertainty on the other hand, represents subjective uncertainty of the system dynamics, which also contributes to cumulative-cost variance, yet does offer exploratory value (being a reducible uncertainty). In § 4.4, both forms of uncertainty are confounded in a single cumulative-cost variance term, and ideally we wish to separate these confounding factors, to target epistemic uncertainty only. Work in progress for doing so was discussed § 4.5.

Another exciting avenue for continued research lies in deep RL. In Chapter 4 we developed a framework for data efficient deep RL. The great promise of deep RL is learning control of tasks with high dimensional states, observations and control outputs (e.g. millions of dimensions). However, we have only shown how data efficient deep RL is possible in smaller, four dimensional state tasks. Future work could be to incorporate both dimensional-reduction of pixel-space observations and probabilistic simulation of smaller-dimensional states. Ideally the dimensionality reduction would be supervised by an ability to predict future simulated costs (critical for controller optimisation), not unsupervised (e.g. auto-encoders) as some other methods do (Stadie et al., 2015; Wahlström et al., 2015). The result would be the first data efficient high-dimensional RL algorithm.

Now we examine some smaller scale improvements to our work. We revisit the equations of control on page 151, to compare: 1) the general equations of control (6.1) – (6.6) (copied from § 2.1 for reader convenience), 2) the restricted set of equations we considered throughout this thesis (6.7) – (6.13) (copied from § 2.6 for reader convenience), and 3) a potential set of equations we consider for future work: (6.14) – (6.20).

Observations that can be modelled as the latent state with additive Gaussian noise is one of our greatest restrictions in this thesis. In (6.9) our sensor model is much less general than (6.3). Learning a nonlinear observation function is too

General equations of discrete-time control:

$$
\begin{aligned}
x_0 &= \varepsilon^{x_0} && \in \mathbb{R}^{\mathrm{X}}, && (6.1)\\
x_{t+1} &= f(x_t, u_t, t, \varepsilon_t^x) && \in \mathbb{R}^{\mathrm{X}}, && (6.2)\\
y_t &= g(x_t, u_t, t, \varepsilon_t^y) && \in \mathbb{R}^{\mathrm{Y}}, && (6.3)\\
u_t &= \pi(y_{0:t}, u_{0:t-1}, t, \varepsilon_t^u) && \in \mathbb{R}^{\mathrm{U}}, && (6.4)\\
c_t &= \mathrm{cost}(x_t, u_t, x_{t+1}, t, \varepsilon_t^c) && \in \mathbb{R}, && (6.5)\\
J_t &= \textstyle\sum_{\tau=t}^{T} \gamma^{\tau-t} \mathbb{E}\left[c_\tau\right] && \in \mathbb{R}. && (6.6)
\end{aligned}
$$

Control equations this thesis considered:

$$
\begin{aligned}
x_0 &= \varepsilon^{x_0} && \in \mathbb{R}^{\mathrm{X}} && , \ \varepsilon^{x_0} \overset{iid}{\sim} \mathcal{N}(\mu_0^x, \Sigma_0^x), && (6.7)\\
x_{t+1} &= f(x_t, u_t) + \varepsilon_t^x && \in \mathbb{R}^{\mathrm{X}} && , \ \varepsilon_t^x \overset{iid}{\sim} \mathcal{N}(0, \Sigma_x^\varepsilon), && (6.8)\\
y_t &= x_t + \varepsilon_t^y && \in \mathbb{R}^{\mathrm{X}} && , \ \varepsilon_t^y \overset{iid}{\sim} \mathcal{N}\left(0, \Sigma_y^\varepsilon\right), && (6.9)\\
b_{t|t} &= p(x_t | y_{0:t}, u_{0:t-1}) && \in \mathcal{N}^{\mathrm{X}} && , && (6.10)\\
u_t &= \pi(\mathbb{E}\left[b_{t|t}\right]) && \in \mathbb{R}^{\mathrm{U}} && , && (6.11)\\
c_t &= \mathrm{cost}(x_t) && \in [0,1] && , && (6.12)\\
J_t &= \textstyle\sum_{\tau=t}^{T} \gamma^{\tau-t} \mathbb{E}\left[c_\tau\right] && \in [0, T-t+1] && . && (6.13)
\end{aligned}
$$

Control equations for future work:

$$
\begin{aligned}
x_0 &= \varepsilon^{x_0} && \in \mathbb{R}^{\mathrm{X}} && , \ \varepsilon^{x_0} \overset{iid}{\sim} \mathcal{N}(\mu_0^x, \Sigma_0^x), && (6.14)\\
x_{t+1} &= f(x_t, u_t) + \varepsilon_t^x && \in \mathbb{R}^{\mathrm{X}} && , \ \varepsilon_t^x \overset{iid}{\sim} \mathcal{N}(0, \Sigma_x^\varepsilon), && (6.15)\\
y_t &= Cx_t + Du_t + \varepsilon_t^y && \in \mathbb{R}^{\mathrm{Y}} && , \ \varepsilon_t^y \overset{iid}{\sim} \mathcal{N}\left(0, \Sigma_y^\varepsilon\right), && (6.16)\\
b_{t|t} &= p(x_t | y_{0:t}, u_{0:t-1}) && \in \mathcal{N}^{\mathrm{X}} && , && (6.17)\\
u_t &= \pi(\mathbb{E}\left[b_{t|t}\right], \mathbb{V}\left[b_{t|t}\right], t) + \varepsilon_t^u && \in \mathbb{R}^{\mathrm{U}} && , \ \varepsilon_t^u \overset{iid}{\sim} \mathcal{N}(0, \Sigma_u^\varepsilon), && (6.18)\\
c_t &= \mathrm{cost}(x_t, u_t, x_{t+1}, t) + \varepsilon_t^c && \in \mathbb{R}, && , \ \varepsilon_t^c \overset{iid}{\sim} p(\cdot), && (6.19)\\
J_t &= \textstyle\sum_{\tau=t}^{T} \gamma^{\tau-t} \mathbb{E}\left[c_\tau\right] && \in \mathbb{R} && . && (6.20)
\end{aligned}
$$

complex a topic for this thesis. However, linear control theory is more complete and mature, and an easier extension is to use a *linear* observation function seen (6.16).

In Chapter 3 we changed the controller's input from observation y_t to a function of the belief-mean in (6.11). A better approach would be to decide controls based on the full belief-distribution. Since we fit belief distributions as Gaussian, an easy extension of our work is a controller that is a function of both belief-mean and belief-variance, seen (6.18). Such an extension may improve performance significantly, giving a controller the flexibility to act 'cautiously' under significant uncertainty about the system state (large belief-variance), otherwise aggressively for a particular state x if certain (small belief-variance). Additionally inputting time t would be a trivial change, although the time-invariant dynamical system we consider, the benefits are limited. Additive iid Gaussian controller noise ε_t^u seen (6.18) would also be an easy extension to our analytic Gaussian framework.

We also considered a generic cost function, (6.12), as a function of the state only. Since our cost function is already arbitrary, the user has reasonable freedom in specifying the objective 'undesirability' of particular systems states x_t, opposed to LQ control methods (see § 2.2.3) which must use quadratic cost functions, regardless of user preferences. Additionally, we found saturating cost functions useful, since they *bound* the cost function (e.g. within $[0, 1]$), useful for systems whose performance is evaluated with a soft binary of 'works well' or 'is not working'. Quadratic-costs by contrast can evaluate states as *arbitrarily*-bad, and can completely dominate the controller optimisation process as the expense of improving the probably that a system can be controlled 'well enough'.

Our cost function could easily be generalised to (6.19), almost as general as (6.5) except with additive noise. Since the loss function uses expected costs, and cost-noise represents only aleatoric not epistemic uncertainty (see § 4.5), we can safely ignore additive iid cost noise from arbitrary probability distributions. The other cost inputs in (6.19), the control u_t and following state x_{t+1} are straightforward to integrate into simulation (the complete sequence of states, controls, and associated cross-covariances required for simulation were already derived in § 4.4.1). Inputting time is also a trivial change to the cost function, since it is 'known' in advance - no need to simulate it!

Thus, several open questions and avenues for continued research exist, which can hopefully extend the field of data efficient learning of control even further.

APPENDIX A

MATHEMATICAL IDENTITIES

A.1 BASICS OF PROBABILITY

A.1.1 OPERATORS OF RANDOM VARIABLES

Notation: Random variables and matrices are capitalised.

Let X and Y each be a random column vector.

EXPECTATION

$$\mathbb{E}_X[f(X)] \doteq \int f(x)p(X=x)\mathrm{d}x. \tag{A.1}$$

COVARIANCE

$$\begin{aligned}\mathbb{C}_{X,Y}[X, Y] &\doteq \mathbb{E}_{X,Y}[(X-\mathbb{E}_X[X])(Y-\mathbb{E}_Y[Y])^\intercal] \\ &= \mathbb{E}_{X,Y}[XY^\intercal]-\mathbb{E}_X[X]\mathbb{E}_Y[Y]^\intercal.\end{aligned} \tag{A.2}$$

VARIANCE

$$\mathbb{V}_X[X] \doteq \mathbb{C}_X[X, X]. \tag{A.3}$$

INDEPENDENCE

$X_\perp$ and $Y_\perp$ are independent if and only if:

$$p(X_\perp, Y_\perp) = p(X_\perp)p(Y_\perp). \tag{A.4}$$

Independence is a stronger property than zero-covariance: independence implies zero-covariance, however, zero-covariance does not imply independence.

A.1.2 ALGEBRA OF RANDOM VARIABLES

Let X and Y be (possibly 'intra' and/or 'inter' dependent) random row vectors, and A and B be matrices.

Expectation:

$$\mathbb{E}\left[AX+BY+c\right] = A\mathbb{E}\left[X\right]+B\mathbb{E}\left[Y\right]+c. \tag{A.5}$$

Variance:

$$\begin{aligned}
\mathbb{V}\left[AX+BY+c\right], &= \mathbb{V}\left[[A,B]\begin{bmatrix}X\\Y\end{bmatrix}+c\right] \\
&= [A,B]\mathbb{V}\begin{bmatrix}X\\Y\end{bmatrix}[A,B]^\intercal \\
&= [A,B]\Big(\mathbb{E}\left[\begin{bmatrix}X\\Y\end{bmatrix}\begin{bmatrix}X\\Y\end{bmatrix}^\intercal\right]-\mathbb{E}\begin{bmatrix}X\\Y\end{bmatrix}\mathbb{E}\begin{bmatrix}X\\Y\end{bmatrix}^\intercal\Big)[A,B]^\intercal \\
&= A\mathbb{V}\left[X\right]A^\intercal+A\mathbb{C}\left[X,\,Y\right]B^\intercal+B\mathbb{C}_X\left[Y,\,A\right]^\intercal+B\mathbb{V}\left[Y\right]B^\intercal. \qquad \text{(A.6)}
\end{aligned}$$

Covariance:

$$\mathbb{C}\left[AX+BY+a,\,CZ+c\right] = A\mathbb{C}\left[X,\,Z\right]C^\intercal+B\mathbb{C}\left[Y,\,Z\right]C^\intercal. \tag{A.7}$$

A.2 GAUSSIANS

A.2.1 GAUSSIAN DISTRIBUTION

$$p(X=x) \;=\; \mathcal{N}(x;a,A) \;=\; (2\pi)^{-\frac{d}{2}}|A|^{-\frac{1}{2}}\exp\left[-\tfrac{1}{2}(x-a)^\intercal A^{-1}(x-a)\right]. \tag{A.8}$$

A.2.2 GAUSSIAN PRODUCT

$$\mathcal{N}(X;a,A)\mathcal{N}(X;b,B) = \eta\mathcal{N}(X;c,C), \tag{A.9}$$

where

$$\begin{aligned} C &= (A^{-1}+B^{-1})^{-1} \\ &= A(A+B)^{-1}B \\ &= B(A+B)^{-1}A, && \text{(A.10)} \\ c &= C(A^{-1}a+B^{-1}b) \\ &= B(A+B)^{-1}a+A(A+B)^{-1}b, && \text{(A.11)} \\ \eta &= \mathcal{N}(a;b,A+B). && \text{(A.12)} \end{aligned}$$

More generally:

$$\prod_i \mathcal{N}(X;\mu_i,\Sigma_i) \propto \mathcal{N}(X;\mu,\Sigma), \tag{A.13}$$

where

$$\begin{aligned} \Sigma^{-1} &= \sum_i \Sigma_i^{-1}, && \text{(A.14)} \\ \Sigma^{-1}\mu &= \sum_i \Sigma_i^{-1}\mu_i. && \text{(A.15)} \end{aligned}$$

Another generalisation, with square matrix $\mathcal{A}$, and general matrix $\mathcal{B}$,

$$\mathcal{N}(\mathcal{A}X;a,A)\mathcal{N}(\mathcal{B}X;b,B) \propto \mathcal{N}(X;c,C), \tag{A.16}$$

where

$$\begin{aligned} C &= (\mathcal{A}^\top A^{-1}\mathcal{A}+\mathcal{B}^\top B^{-1}\mathcal{B})^{-1} \\ &= \hat{A}-\underbrace{\hat{A}\mathcal{B}^\top(B+\mathcal{B}\hat{A}\mathcal{B}^\top)^{-1}}_{K}\mathcal{B}\hat{A}, \qquad \text{(using (A.81))}, && \text{(A.17)} \\ c &= C(\mathcal{A}^\top A^{-1}a+\mathcal{B}^\top B^{-1}b) && \text{(A.18)} \\ &= (I-K\mathcal{B})\mathcal{A}^\top(\mathcal{A}\mathcal{A}^\top)^{-1}a+Kb, && \text{(A.19)} \\ \hat{A}^{-1} &\doteq \mathcal{A}^\top A^{-1}\mathcal{A}. && \text{(A.20)} \end{aligned}$$

A.2.3 Gaussian Marginalisation of a Mean Parameter

$$\int \mathcal{N}(X;AY,B)\mathcal{N}(Y;c,C)\,\mathrm{d}Y \quad = \quad \mathcal{N}(X;Ac,ACA^\intercal+B)\,. \tag{A.21}$$

A.2.4 Gaussian Marginals and Conditionals

Let:

$$\begin{aligned}\begin{bmatrix}X\\Y\end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix}a\\b\end{bmatrix},\begin{bmatrix}A & C\\C^\intercal & B\end{bmatrix}\right)\\ &\sim \mathcal{N}\left(\begin{bmatrix}a\\b\end{bmatrix},\begin{bmatrix}\hat{A} & \hat{C}\\\hat{C}^\intercal & \hat{B}\end{bmatrix}^{-1}\right),\end{aligned} \tag{A.22}$$

then

$$X \sim \mathcal{N}(a,A)\,, \tag{A.23}$$

$$Y \sim \mathcal{N}(b,B)\,, \tag{A.24}$$

$$\begin{aligned}X|Y &\sim \mathcal{N}\left(a+CB^{-1}(Y-b),A-CB^{-1}C^\intercal\right)\\ &\sim \mathcal{N}\left(a-\hat{A}^{-1}\hat{C}(Y-b),\hat{A}^{-1}\right),\end{aligned} \tag{A.25}$$

$$\begin{aligned}Y|X &\sim \mathcal{N}\left(b+C^\intercal A^{-1}(X-a),B-C^\intercal A^{-1}C\right)\\ &\sim \mathcal{N}\left(b-\hat{B}^{-1}\hat{C}^\intercal(X-a),\hat{B}^{-1}\right).\end{aligned} \tag{A.26}$$

A.2.5 Moments of Functions of Gaussians

Linear and Quadratic Functions of Gaussians

Let $X \sim \mathcal{N}(\mu,\Sigma)$, and *inter*-independent $X_\perp \sim \mathcal{N}(\mu_x,\Sigma_x)$, $Y_\perp \sim \mathcal{N}(\mu_y,\Sigma_y)$, be Gaussian random vectors,

$$\mathbb{E}[AX+a] = A\mu+a, \tag{A.27}$$

$$\mathbb{V}[AX+a] = A\Sigma A^\intercal, \tag{A.28}$$

$$\mathbb{E}[XX^\intercal] = \Sigma+\mu\mu^\intercal, \tag{A.29}$$

$$\mathbb{E}[(AX-a)^\intercal B(AX-a)] = \mathrm{Tr}[A^\intercal BA\Sigma]+(A\mu-a)^\intercal B(A\mu-a), \tag{A.30}$$

$$\mathbb{V}[X^\intercal BX] = \mathrm{Tr}[B\Sigma(B+B^\intercal)\Sigma]+\mu^\intercal(B+B^\intercal)\Sigma(B+B^\intercal)\mu, \tag{A.31}$$

$$\mathbb{E}[X_\perp^\intercal Y_\perp] = \mu_x^\intercal\mu_y, \tag{A.32}$$

$$\mathbb{V}[X_\perp^\intercal Y_\perp] = \mathrm{Tr}[\Sigma_x\Sigma_y]+\mu_x^\intercal\Sigma_y\mu_x+\mu_y^\intercal\Sigma_x\mu_y. \tag{A.33}$$

Truncated Univariate Gaussian

Let $X \sim \mathcal{N}\left(\mu, \sigma^2\right)$, $-\infty \leq a < b \leq \infty$, $\phi(\cdot)$ be the standard normal distribution, $\Phi(\cdot)$ its cumulative distribution function,

$$\begin{aligned}
\alpha &= (a-\mu)/\sigma, && \text{(A.34)}\\
\beta &= (b-\mu)/\sigma, && \text{(A.35)}\\
\mathbb{E}_X\left[X|a<X<b\right] &= \mu + \sigma\frac{\phi(\alpha)-\phi(\beta)}{\Phi(\beta)-\Phi(\alpha)}, && \text{(A.36)}\\
\mathbb{V}_X\left[X|a<X<b\right] &= \sigma^2\left[1+\frac{\alpha\phi(\alpha)-\beta\phi(\beta)}{\Phi(\beta)-\Phi(\alpha)}-\left(\frac{\phi(\alpha)-\phi(\beta)}{\Phi(\beta)-\Phi(\alpha)}\right)^2\right]. && \text{(A.37)}
\end{aligned}$$

Exponential Functions of Gaussians

Let $X \sim \mathcal{N}(\mu, \Sigma)$ be a Gaussian random vector, where $\mu \in \mathbb{R}^D$, $\Sigma \in \mathbb{R}^{D\times D}$ (symmetric), and $a \in \mathbb{R}$, $b \in \mathbb{R}^D$, $B \in \mathbb{R}^{D\times D}$ (symmetric), and $Y = \exp\left(-\frac{1}{2}(X-b)^\intercal B(X-b)\right)$,

$$\begin{aligned}
\mathbb{E}_X\left[\exp(aX_i)\right] &= \exp(a\mu_i + a^2\Sigma_{ii}/2), && \text{(A.38)}\\
\mathbb{E}_X\left[Y\right] &= \det(I+\Sigma B)^{-1/2}\exp\Big(-\tfrac{1}{2}(\mu-b)^\intercal B(I+\Sigma B)^{-1}(\mu-b)\Big), && \text{(A.39)}\\
\mathbb{V}_X\left[Y\right] &= \det(I+2\Sigma B)^{-1/2}\exp\Big(-(\mu-b)^\intercal B(I+2\Sigma B)^{-1}(\mu-b)\Big) \\
&\quad -\mathbb{E}_X\left[Y\right]^2, && \text{(A.40)}\\
\mathbb{C}_X\left[X,\, Y\right] &= \Sigma\mathbb{E}_X\left[Y\right]\Big(Bb - B(I+\Sigma B)^{-1}(\Sigma Bb+\mu)\Big). && \text{(A.41)}
\end{aligned}$$

Proofs available Appendix E.1.

TRIGONOMETRIC FUNCTIONS OF GAUSSIANS

Let $X \sim \mathcal{N}(\mu, \Sigma)$ be a Gaussian random vector, where $\mu \in \mathbb{R}^D$, $\Sigma \in \mathbb{R}^{D \times D}$ (symmetric), and $a \in \mathbb{R}$:

$$\mathbb{E}_X[\sin(aX_i)] = \exp(-a^2\Sigma_{ii}/2)\sin(a\mu_i), \tag{A.42}$$

$$\mathbb{E}_X[\cos(aX_i)] = \exp(-a^2\Sigma_{ii}/2)\cos(a\mu_i), \tag{A.43}$$

$$\mathbb{V}_X[\sin(aX_i)] = \tfrac{1}{2}\Big(1+\exp(-a^2\Sigma_{ii})\cos(2a\mu_i)\Big)\Big(1-\exp(-a^2\Sigma_{ii})\Big), \tag{A.44}$$

$$\mathbb{V}_X[\cos(aX_i)] = \tfrac{1}{2}\Big(1-\exp(-a^2\Sigma_{ii})\cos(2a\mu_i)\Big)\Big(1-\exp(-a^2\Sigma_{ii})\Big), \tag{A.45}$$

$$\mathbb{C}_X[\sin(aX_i), \cos(aX_i)] = -\tfrac{1}{2}\exp(-a^2\Sigma_{ii})\sin(2a\mu_i)\Big(1-\exp(-a^2\Sigma_{ii})\Big), \tag{A.46}$$

$$\mathbb{C}_X\left[X_i, \sin(X_j)\right] = \Sigma_{ij}\exp(-\Sigma_{jj}/2)\cos(\mu_j), \tag{A.47}$$

$$\mathbb{C}_X\left[X_i, \cos(X_j)\right] = -\Sigma_{ij}\exp(-\Sigma_{jj}/2)\sin(\mu_j). \tag{A.48}$$

Proofs available Appendix E.2.

A.2.6 APPROXIMATE COVARIANCE OF FUNCTIONS OF GAUSSIANS

Let $X \sim \mathcal{N}(\mu, \Sigma)$ be a Gaussian-distributed vector, and f and g be two nonlinear functions:

$$\mathbb{C}_X[f(X), g(X)] \approx \mathbb{C}_X[f(X), X]\Sigma^{-1}\mathbb{C}_X[X, g(X)] \tag{A.49}$$

$$= F^\intercal \Sigma G, \tag{A.50}$$

$$F \doteq \Sigma^{-1}\mathbb{C}_X[X, f(X)], \tag{A.51}$$

$$G \doteq \Sigma^{-1}\mathbb{C}_X[X, g(X)]. \tag{A.52}$$

Explanation: even though the distribution of $f(X)$ is intractable and non-Gaussian, f is often of the form where we can often compute $F = \Sigma^{-1}\mathbb{C}_X[X, f(X)]$ analytically. If so, then the matrix F is useful since the linear transformation $\bar{f}(X) = F^\intercal X$ is Gaussian-distributed and preserves the input-output covariance of X and $f(X)$, i.e. $\mathbb{C}_X\left[X, \bar{f}(X)\right] = \mathbb{C}_X[X, F^\intercal X] = \Sigma F = \mathbb{C}_X[X, f(X)]$. We can use this linearisation of function f (a linearisation which considers both μ and Σ) to approximate other quantities, i.e. : $\mathbb{C}_X[f(X), g(X)] \approx \mathbb{C}_X\left[\bar{f}(X), \bar{g}(X)\right] = \mathbb{C}_X[F^\intercal X, G^\intercal X] = F^\intercal \Sigma G$. Note the approximation in (A.49) becomes an exact equality if either f or g is linear.

Consider another, similar problem of computing $\mathbb{C}_X[X,\, h(f(X))]$ approximately, given two nonlinear functions f and h. Then using (A.49) we have:

$$\begin{aligned}
\mathbb{C}_X[X,\, h(f(X))] &\approx \mathbb{C}_X[X,\, f(X)]\,\mathbb{V}_X[f(X)]^{-1}\,\mathbb{C}_X[f(X),\, h(f(X))] && \text{(A.53)}\\
&= \Sigma FH, && \text{(A.54)}\\
F &\doteq \Sigma^{-1}\mathbb{C}_X[X,\, f(X)], && \text{(A.55)}\\
H &\doteq \mathbb{V}_X[f(X)]^{-1}\,\mathbb{C}_X[f(X),\, h(f(X))], && \text{(A.56)}
\end{aligned}$$

applicable to situations where F and H are computable. Note a nesting of functions in (A.53) correspond to multiplication of linearisation terms in (A.54). The relationship holds for deeper nesting too. Intuitively this makes sense for linear functions (where the approximate equality becomes exact) since nested linear functions $h(f(X))$ can be collapsed to a single linear function $(FH)^\intercal X$. Viewing FH as a single linear transform, the relationship above is obvious: $\mathbb{C}_X[X,\, (FH)^\intercal X] = \Sigma(FH)$.

A.2.7 Gaussian Derivatives

The following holds for *symmetric* A, where $\odot$ is the element wise product:

$$\begin{aligned}
\frac{\partial}{\partial x}\log\mathcal{N}(x;a,A) &= \frac{\partial}{\partial x}\big(-\tfrac{1}{2}(x-a)^\intercal A^{-1}(x-a)\big)\\
&= -A^{-1}(x-a), && \text{(A.57)}\\
\frac{\partial}{\partial a}\log\mathcal{N}(x;a,A) &= A^{-1}(x-a), && \text{(A.58)}\\
\frac{\partial}{\partial A}\log\mathcal{N}(x;a,A) &= \frac{\partial}{\partial A}\big(-\tfrac{1}{2}\log(|A|) - \tfrac{1}{2}(x-a)^\intercal A^{-1}(x-a)\big)\\
&= -\tfrac{1}{2}A^{-1} + \tfrac{1}{2}A^{-1}(x-a)(x-a)^\intercal A^{-1}, && \text{(A.59)}\\
\partial\mathcal{N}(\cdot) &= \mathcal{N}(\cdot)\odot\partial\log\mathcal{N}(\cdot). && \text{(A.60)}
\end{aligned}$$

A.2.8 Gaussian Integrals

Let $\phi(\cdot)$ be the standard normal distribution, $\Phi(\cdot)$ its cumulative distribution function,

$$\int z\phi(z)dz = -\phi(z)+C, \tag{A.61}$$

$$\int z\phi(a+bz)dz = -b^{-2}(\phi(a+bz)+a\Phi(a+bz))+C, \tag{A.62}$$

$$\int x\mathcal{N}\left(x;\mu,\sigma^2\right)dx = -\sigma\phi(\frac{x-\mu}{\sigma})+\mu\Phi(\frac{x-\mu}{\sigma})+C. \tag{A.63}$$

A.3 MATRICES

A.3.1 TRACE

The following reduces a $O(n^3)$ operation to $O(n^2)$:

$$\text{Tr}(X^\intercal Y) = \sum_{ij}(X\odot Y)_{ij}. \tag{A.64}$$

A.3.2 DETERMINANT

The determinate has a geometric interpretation. Consider matrices $A\in\mathbb{R}^{n\times n}$, $B\in\mathbb{R}^{n\times n}$, $C\in\mathbb{R}^{n\times m}$, $D\in\mathbb{R}^{m\times m}$, $E\in\mathbb{R}^{m\times n}$, and $a\in\mathbb{R}$. If we separate the rows (or columns) of A into n many n-dimensional vectors, then intuitive for the determinate is that $\det(A)$ equals the volume of the n-dimensional parallelepiped the vectors define.

$$\begin{aligned}
\det(A) &= \sum_{i=1}^{n}(-1)^{i+j}a_{ij}M_{ij} \\
&= \sum_{j=1}^{n}(-1)^{i+j}a_{ij}M_{ij}, && \text{(A.65)}\\
\det(A) &= \prod_i \text{eig}_i(A), && \text{(A.66)}\\
\det(A^\intercal) &= \det(A), && \text{(A.67)}\\
\det(aA) &= a^n\det(A), \quad \text{where } a\in\mathbb{R}, A\in\mathbb{R}^{n\times n}, && \text{(A.68)}\\
\det(AB) &= \det(A)\det(B), && \text{(A.69)}\\
\det(A^{-1}) &= 1/\det(A), && \text{(A.70)}\\
\det(aI_n \pm CD^{-1}E) &= a^{n-m}\det(aD\pm EC)/\det(D), && \text{(A.71)}
\end{aligned}$$

where M_{ij} is the minor matrix (removing the i'th row and j'th column of matrix A.)

A.3.3 ADJUNCT

$$
\begin{aligned}
A^{-1} &= \tfrac{1}{\det(A)}\mathrm{adj}(A), && \text{(A.72)}\\
\mathrm{adj}(A^\intercal) &= \mathrm{adj}(A)^\intercal, && \text{(A.73)}\\
\mathrm{adj}(aA) &= a^{n-1}\mathrm{adj}(A), && \text{(A.74)}\\
\mathrm{adj}(AB) &= \mathrm{adj}(B)\mathrm{adj}(A). && \text{(A.75)}
\end{aligned}
$$

A.3.4 EIGENVALUES

Let matrix $A \in \mathbb{R}^{n\times n}$ have unsorted eigenvalues $\mathrm{eig}(A) = [\lambda_1, ..., \lambda_n]$.

$$
\begin{aligned}
\det(A) &= \prod_i \lambda_i, && \text{(A.76)}\\
\mathrm{Tr}(A) &= \sum_i \lambda_i, && \text{(A.77)}\\
\mathrm{eig}(A^\intercal) &= \mathrm{eig}(A), && \text{(A.78)}\\
\mathrm{eig}(A^m) &= [\lambda_1^m, ..., \lambda_n^m], && \text{(A.79)}\\
\mathrm{eig}(A^{-1}) &= [1/\lambda_1, ..., 1/\lambda_n]. && \text{(A.80)}
\end{aligned}
$$

A.3.5 INVERSE

MATRIX INVERSION LEMMA

Let A, C and $C^{-1} + DA^{-1}B$ be non-singular square matrices, then:

$$
(A+BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}. \quad \text{(A.81)}
$$

BLOCK-MATRIX INVERSE

In general:

$$
\begin{aligned}
\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} (A-BD^{-1}C)^{-1} & -A^{-1}B(D-CA^{-1}B)^{-1} \\ -D^{-1}C(A-BD^{-1}C)^{-1} & (D-CA^{-1}B)^{-1} \end{bmatrix}\\
&= \begin{bmatrix} (A-BD^{-1}C)^{-1} & -(A-BD^{-1}C)^{-1}BD^{-1} \\ -(D-CA^{-1}B)^{-1}CA^{-1} & (D-CA^{-1}B)^{-1} \end{bmatrix}. \quad \text{(A.82)}
\end{aligned}
$$

A special case is if the full matrix are symmetric **and** all individual block matrices are themselves symmetric:

$$\begin{bmatrix} A & B \\ B & D \end{bmatrix}^{-1} = \begin{bmatrix} (A-BD^{-1}B)^{-1} & (B-DB^{-1}A)^{-1} \\ (B-AB^{-1}D)^{-1} & (D-BA^{-1}B)^{-1} \end{bmatrix}. \tag{A.83}$$

INVERSE WITH AN INFINITE ELEMENT

The inverse of a matrix A whose element $A_{ij} = \infty$, is such that $(A^{-1})_{kl} = 0$ if $k = j$ or $l = i$, otherwise populated by values of submatrix $(M_{I\setminus\{i\},J\setminus\{j\}})^{-1}$. I.e. let

$$\begin{aligned} A &= \begin{bmatrix} a & \infty & c \\ d & e & f \\ g & h & i \end{bmatrix}, \quad \text{with submatrix} \\ M_{12} &= \begin{bmatrix} d & f \\ g & i \end{bmatrix} \quad \text{and} \\ M_{12}^{-1} &= \begin{bmatrix} \hat{d} & \hat{f} \\ \hat{g} & \hat{i} \end{bmatrix}. \quad \text{Then} \\ A^{-1} &= \begin{bmatrix} 0 & \hat{d} & \hat{f} \\ 0 & 0 & 0 \\ 0 & \hat{g} & \hat{i} \end{bmatrix}. \end{aligned}$$

A.4 MATRIX DERIVATIVES

A.4.1 BASIC IDENTITIES

$$\begin{aligned} \partial X^{-1} &= -X^{-1}(\partial X)X^{-1}, && \text{(A.84)} \\ \partial(XY) &= \partial(X)Y + X\partial(Y), && \text{(A.85)} \\ \partial \operatorname{Tr}(X) &= \operatorname{Tr}(\partial X), && \text{(A.86)} \\ \partial \det(X) &= \det(X) \operatorname{Tr}\left(X^{-1}\partial X\right), && \text{(A.87)} \\ \partial \log\det(X) &= \operatorname{Tr}\left(X^{-1}\partial X\right), && \text{(A.88)} \end{aligned}$$

with some proofs available in Appendix E.3. Thus,

$$\frac{\partial}{\partial X}\det(X) = \det(X)X^{-\intercal}, \tag{A.89}$$

$$\frac{\partial}{\partial X}\log\det(X) = X^{-\intercal}, \tag{A.90}$$

$$\frac{\partial}{\partial X}\det(X^{-1}) = -\det(X^{-1})X^{-\intercal}, \tag{A.91}$$

$$\frac{\partial}{\partial X_{ij}}(X^{-1})_{kl} = -(X^{-1})_{ki}(X^{-1})_{jl}, \tag{A.92}$$

$$\frac{\partial}{\partial (X^{-1})}f(X) = -X^{\intercal}\Big(\frac{\partial}{\partial X}f(X)\Big)X^{\intercal}, \qquad f:\mathbb{R}^{n\times n}\to\mathbb{R}, \tag{A.93}$$

$$\frac{\partial}{\partial (X^{-1})_{ij}}f(X)_{kl} = -X_{jl}\Big(\frac{\partial}{\partial X_{kl}}f(X)_{kl}\Big)X_{ki}, \qquad f:\mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}. \tag{A.94}$$

A.4.2 Scalar-Numerator Identities

For general matrices X, A, B, C, $D_i\forall i$, columns vectors a, b, functions $U_i(X)\forall i$, and letting:

- $Y = AXB+C$,
- $Z = A\sum_i\left[(X+D_i)^{-1}\right]B+C$,
- $U = \prod_i U_i(X)$,

we have:

$$\frac{\partial}{\partial X}a^{\intercal}Xb = ab^{\intercal}, \tag{A.95}$$

$$\frac{\partial}{\partial X}\log\det(Y) = A^{\intercal}Y^{-\intercal}B^{\intercal}, \tag{A.96}$$

$$\frac{\partial}{\partial X}\log\det(Z) = -\sum_i\Big[(X+D_i)^{-\intercal}A^{\intercal}Z^{-\intercal}B^{\intercal}(X+D_i)^{-\intercal}\Big], \tag{A.97}$$

$$\frac{\partial}{\partial X}a^{\intercal}Y^{-1}b = -A^{\intercal}Y^{-\intercal}ab^{\intercal}Y^{-\intercal}B^{\intercal}, \tag{A.98}$$

$$\frac{\partial}{\partial X}a^{\intercal}Z^{-1}b = \sum_i\Big[(X+D_i)^{-\intercal}A^{\intercal}Z^{-\intercal}ab^{\intercal}Z^{-\intercal}B^{\intercal}(X+D_i)^{-\intercal}\Big], \tag{A.99}$$

$$\frac{\partial}{\partial X}U = U\odot\sum_i\Big[\frac{\partial}{\partial X}\log U_i\Big], \qquad \text{(product rule)}. \tag{A.100}$$

A.4.3 MATRIX-NUMERATOR IDENTITIES

Let vectors a and b be instead $e_i = [0,..., \underbrace{1}_{i\text{'th element}},...0]$ and $e_j = [0,..., \underbrace{1}_{j\text{'th element}},...0]$, we can rewrite the above expressions as:

$$\frac{\partial}{\partial X}(\cdot)_{ij} = \frac{\partial}{\partial X} e_i^\intercal (\cdot) e_j = I e_i e_j^\intercal J. \tag{A.101}$$

Iterating over i and j creates a 4-dimensional tensor, whose first two dimensions correspond to elements in the matrix-numerator $(\cdot)$, and last two dimensions correspond to elements in the matrix-denominator X. We transform 4D tensors by partially unwrapping into 2D matrices, whose rows are unwrapped numerators, columns unwrapped denominators. Let $\text{vec}(\cdot)$ a function that unwraps matrices into vectors in column-major order (enumerating down columns first, then rows):

$$\begin{aligned} \frac{\partial}{\partial \text{vec}(X)} \text{vec}(\cdot) &= \text{kron}(J, I^\intercal) \\ &= \begin{bmatrix} J_{11}I^\intercal & J_{12}I^\intercal & \cdots \\ J_{21}I^\intercal & J_{22}I^\intercal & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}. \end{aligned} \tag{A.102}$$

Example 1:

$$\begin{aligned} \text{Let } \; Y &= AXB + C, \\ \frac{\partial}{\partial X} e_i^\intercal D Y^{-1} E e_j, &= \underbrace{(-A^\intercal Y^{-\intercal} D^\intercal)}_{I} e_i e_j^\intercal \underbrace{(E^\intercal Y^{-\intercal} B^\intercal)}_{J}, \\ \therefore \frac{\partial}{\partial \text{vec}(X)} \text{vec}(DY^{-1}E) &= \text{kron}(J, I^\intercal) \\ &= \text{kron}(E^\intercal Y^{-\intercal} B^\intercal \; , \; -DY^{-1}A). \end{aligned}$$

Example 2:

$$\begin{aligned} \frac{\partial}{\partial X} e_i^\intercal I^\intercal X J^\intercal e_j &= I e_i e_j^\intercal J, \\ \therefore \frac{\partial}{\partial \text{vec}(X)} \text{vec}(I^\intercal X J^\intercal) &= \text{kron}(J, I^\intercal). \end{aligned}$$

Note, kron terms can be collected using

$$\text{kron}(A+B, C+D) = \text{kron}(A,C) + \text{kron}(A,D) + \text{kron}(B,C) + \text{kron}(B,D). \tag{A.103}$$

APPENDIX B

GAUSSIAN PROCESS PREDICTION

B.1 SOME IDENTITIES

The two functions

$$\begin{aligned} q(x,x',\Lambda,V) &\doteq |\Lambda^{-1}V+I|^{-1/2}\exp\left(-\tfrac{1}{2}(x-x')[\Lambda+V]^{-1}(x-x')\right) \\ &= (2\pi)^{D/2}|\Lambda|^{1/2}\mathcal{N}\left(x;x',\Lambda+V\right), \qquad \text{(B.1)} \\ Q(x,x',\Lambda_a,\Lambda_b,V,\mu,\Sigma) &\doteq c_1\exp\left(-\tfrac{1}{2}(x-x')^{\mathsf{T}}[\Lambda_a+\Lambda_b+2V]^{-1}(x-x')\right) \\ &\quad \times\exp\left(-\tfrac{1}{2}(z-\mu)^{\mathsf{T}}\left[\left((\Lambda_a+V)^{-1}+(\Lambda_b+V)^{-1}\right)^{-1}\right.\right. \\ &\quad \left.\left.+\Sigma\right]^{-1}(z-\mu)\right), \qquad \text{(B.2)} \\ &= c_2\, q(x,\mu,\Lambda_a,V)\, q(\mu,x'\Lambda_b,V) \\ &\quad \times\exp\left(\tfrac{1}{2}r^{\mathsf{T}}\left[(\Lambda_a+V)^{-1}+(\Lambda_b+V)^{-1}+\Sigma^{-1}\right]^{-1}r\right) \qquad \text{(B.3)} \end{aligned}$$

where

$$\begin{aligned} z &\doteq (\Lambda_b+V)(\Lambda_a+\Lambda_b+2V)^{-1}x+(\Lambda_a+V)(\Lambda_a+\Lambda_b+2V)^{-1}x', \qquad \text{(B.4)} \\ r &\doteq (\Lambda_a+V)^{-1}(x-\mu)+(\Lambda_b+V)^{-1}(x'-\mu), \qquad \text{(B.5)} \\ c_1 &\doteq \left|(\Lambda_a+V)(\Lambda_b+V)+(\Lambda_a+\Lambda_b+2V)\Sigma\right|^{-1/2}\left|\Lambda_a\Lambda_b\right|^{1/2}, \qquad \text{(B.6)} \\ c_2 &\doteq \left|\left((\Lambda_a+V)^{-1}+(\Lambda_b+V)^{-1}\right)\Sigma+I\right|^{-1/2}, \qquad \text{(B.7)} \end{aligned}$$

have the following Gaussian integrals

$$\int q(x,t,\Lambda,V)\mathcal{N}(t;\mu,\Sigma)\,\mathrm{d}t = q(x,\mu,\Lambda,\Sigma+V), \tag{B.8}$$

$$\int (t-\mu)q(x,t,\Lambda,V)\mathcal{N}(t;\mu,\Sigma)\,\mathrm{d}t = \Sigma(\Lambda+\Sigma+V)^{-1}(x-\mu) \times q(x,\mu,\Lambda,\Sigma+V), \tag{B.9}$$

$$\int (x-t)q(x,t,\Lambda,V)\mathcal{N}(t;\mu,\Sigma)\,\mathrm{d}t = (\Lambda+V)(\Lambda+\Sigma+V)^{-1}(x-\mu) \times q(x,\mu,\Lambda,\Sigma+V), \tag{B.10}$$

$$\int q(x,t,\Lambda_a,V)\,q(t,x',\Lambda_b,V)\mathcal{N}(t;\mu,\Sigma)\,\mathrm{d}t = Q(x,x',\Lambda_a,\Lambda_b,V,\mu,\Sigma), \tag{B.11}$$

$$\int Q(x,x',\Lambda_a,\Lambda_b,0,t,V)\mathcal{N}(t;\mu,\Sigma)\,\mathrm{d}t = Q(x,x',\Lambda_a,\Lambda_b,0,\mu,\Sigma+V), \tag{B.12}$$

with some proofs available in Appendix E.4.

B.1.1 Derivatives

For symmetric Λ and V and Σ:

$$\frac{\partial \ln q(x,x',\Lambda,V)}{\partial x} = -(\Lambda+V)^{-1}(x-x') = -(\Lambda^{-1}V+I)^{-1}\Lambda^{-1}(x-x'), \tag{B.13}$$

$$\frac{\partial \ln q(x,x',\Lambda,V)}{\partial x'} = (\Lambda+V)^{-1}(x-x'), \tag{B.14}$$

$$\frac{\partial \ln q(x,x',\Lambda,V)}{\partial V} = -\tfrac{1}{2}(\Lambda+V)^{-1} + \tfrac{1}{2}(\Lambda+V)^{-1}(x-x')(x-x')^{\mathsf{T}}(\Lambda+V)^{-1}. \tag{B.15}$$

Let $L = (\Lambda_a+V)^{-1} + (\Lambda_b+V)^{-1}$, $S = \Sigma L + I$, $R = S^{-1}\Sigma = \left[L+\Sigma^{-1}\right]^{-1}$, then

$$\partial Q(x,x',\Lambda_a,\Lambda_b,V,\mu,\Sigma) = Q \odot \partial\Big(\ln c_2 + \ln q(x,\mu,\Lambda_a,V), + \ln q(\mu,x'\Lambda_b,V) + \tfrac{1}{2}r^{\mathsf{T}}Rr\Big), \tag{B.16}$$

where

$$\begin{aligned}
\tfrac{1}{2}\frac{\partial r^{\mathsf{T}}Rr}{\partial \mu} &= r^{\mathsf{T}}R\frac{\partial r}{\partial \mu} = -r^{\mathsf{T}}RL, && \text{(B.17)}\\
\frac{\partial \ln c_2}{\partial \Sigma} &= -\tfrac{1}{2}\frac{\partial \ln|L\Sigma+I|}{\partial \Sigma} = -\tfrac{1}{2}L^{\mathsf{T}}(L\Sigma+I)^{-\mathsf{T}} = -\tfrac{1}{2}LS^{-1}, && \text{(B.18)}\\
\frac{\partial r^{\mathsf{T}}Rr}{\partial \Sigma} &= \Sigma^{-\mathsf{T}}R^{\mathsf{T}}rr^{\mathsf{T}}R^{\mathsf{T}}\Sigma^{-\mathsf{T}} = S^{-\mathsf{T}}rr^{\mathsf{T}}S^{-1}, && \text{(B.19)}\\
\frac{\partial \ln c_2}{\partial V} &= -\tfrac{1}{2}\frac{\partial \ln|L\Sigma+I|}{\partial V} = -\tfrac{1}{2}\frac{\partial \ln|\sum_i\left[(\Lambda_i+V)^{-1}\right]\Sigma+I|}{\partial V}\\
&= \tfrac{1}{2}\sum_i\left[(\Lambda_i+V)^{-\mathsf{T}}\Big(\sum_j\left[(\Lambda_j+V)^{-1}\right]\Sigma+I\Big)^{-\mathsf{T}}\Sigma^{\mathsf{T}}(\Lambda_i+V)^{-\mathsf{T}}\right]\\
&= \tfrac{1}{2}\sum_i\left[(\Lambda_i+V)^{-1}R(\Lambda_i+V)^{-1}\right], && \text{(B.20)}\\
\frac{\partial r^{\mathsf{T}}Rr}{\partial V} &= r^{\mathsf{T}}\frac{\partial R}{\partial V}r+\frac{\partial r^{\mathsf{T}}}{\partial V}Rr+r^{\mathsf{T}}R\frac{\partial r}{\partial V} = \sum_i\left[(\Lambda_i+V)^{-1}R^{\mathsf{T}}rr^{\mathsf{T}}R^{\mathsf{T}}(\Lambda_i+V)^{-1}\right]\\
&\quad -\sum_i\left[(\Lambda_i+V)^{-1}(x_{n_i}-\mu)(Rr)^{\mathsf{T}}(\Lambda_i+V)^{-1}\right]\\
&\quad -\sum_i\left[(\Lambda_i+V)^{-1}(r^{\mathsf{T}}R)^{\mathsf{T}}(x_{n_i}-\mu)^{\mathsf{T}}(\Lambda_i+V)^{-1}\right]. && \text{(B.21)}
\end{aligned}$$

B.2 Predictions from Deterministic Inputs

Consider modelling data comprising vector-input $\tilde{x}\in\mathbb{R}^{(\mathrm{X}+\mathrm{U})}$, vector-output $f(\tilde{x})\in\mathbb{R}^{\mathrm{X}}$ with separate combinations of linear models and GPs to make predictions, $a=1,\ldots,\mathrm{X}$:

$$\begin{aligned}
f_a(\tilde{x}) &\sim \mathcal{N}\left(\mathbb{E}_f\left[f_a(\tilde{x})\right],\mathbb{V}_f\left[f_a(\tilde{x})\right]\right), && \text{(B.22)}\\
\mathbb{E}_f\left[f_a(\tilde{x})\right] &= \beta_a^{\mathsf{T}}k_a(X,\tilde{x})+\phi_a^{\mathsf{T}}\tilde{x}, && \text{(B.23)}\\
\mathbb{V}_f\left[f_a(\tilde{x})\right] &= k_a(\tilde{x},\tilde{x})-k_a(\tilde{x},X)(K_a+\Sigma_a^{\varepsilon})^{-1}k_a(X,\tilde{x}), && \text{(B.24)}
\end{aligned}$$

where the X squared exponential covariance functions are

$$k_a(x,x') = s_a^2 q(x,x',\Lambda_a,0), \text{ where } a=1,\ldots,\mathrm{X}, \qquad \text{(B.25)}$$

and s_a^2 are the signal variances and Λ_a is a diagonal matrix of squared length scales for GP number a. The noise variances Σ_a^{ε} is the identity matrix multiplied by the observation noise of the a'th GP. The inputs are X and the outputs y_a and we define $\beta_a=(K_a+\Sigma_a^{\varepsilon})^{-1}(y_a-\phi_a^{\mathsf{T}}X)$, where K_a is the Gram matrix.

B.3 Predictions from Uncertain Inputs

Prediction from uncertain inputs has been discussed previously in Deisenroth and Rasmussen (2011) and also used for filtered control and dynamics predictions during the system-execution phase. Consider making predictions from $a = 1, \ldots, \mathrm{X}$ GPs at uncertain test input $\tilde{X}$ with specification

$$p(\tilde{X}) \sim \mathcal{N}(m, V). \tag{B.26}$$

We have the following expressions for the predictive mean $\mathbf{m}$, variances $\mathbf{V}$ and input-output covariances $\mathbf{C}$ using the law of iterated expectations and variances:

$$f_a(\tilde{X}) \sim \mathcal{N}(\mathbf{m}, \mathbf{V}). \tag{B.27}$$

B.3.1 Mean:

The scalar expectation of the a'th element of predictive output vector $f(\tilde{X})$ w.r.t. random input vector $\tilde{X}$ is:

$$\begin{aligned} \mathbf{m}^a &\doteq \mathbb{E}_{\tilde{X}}\left[\mathbb{E}_f\left[f_a(\tilde{X})\right]\right] \\ &= \int \left(s_a^2 \beta_a^\intercal q(X, \tilde{X}, \Lambda_a, 0) + \phi_a^\intercal \tilde{X}\right) \mathcal{N}\left(\tilde{X}; m, V\right) \mathrm{d}\tilde{X} \\ &= s_a^2 \beta_a^\intercal q_a + \phi_a^\intercal m, \end{aligned} \tag{B.28}$$

where

$$\beta_a \doteq (K_a + \Sigma_a^\varepsilon)^{-1}(y_a - \phi_a^\intercal X), \tag{B.29}$$

$$q_a^i \doteq q(X_i, m, \Lambda_a, V). \tag{B.30}$$

B.3.2 Covariance:

Let us first define, using identity (B.9):

$$\begin{aligned} \mathbf{C}_a &\doteq V^{-1} \mathbb{C}_{\tilde{X}}\left[\tilde{X},\, s_a^2 \beta_a^\intercal q(X, \tilde{X}, \Lambda_a, 0)\right] \\ &= V^{-1} \int (\tilde{X} - m) s_a^2 \beta_a^\intercal q(X, \tilde{X}, \Lambda_a, 0) \mathcal{N}\left(\tilde{X}; m, V\right) \mathrm{d}\tilde{X} \\ &= s_a^2 (\Lambda_a + V)^{-1} (X - m) \beta_a^\intercal q_a. \end{aligned} \tag{B.31}$$

The covariance vector of the input vector $\tilde{X}$ with the a'th element of predictive output vector $f(\tilde{X})$ is thus:

$$\begin{aligned}\mathbb{C}_{\tilde{X}}\left[\tilde{X},\, f_a(\tilde{X})\right] &= \mathbb{C}_{\tilde{X}}\left[\tilde{X},\, s_a^2\beta_a^\intercal q(X,\tilde{X},\Lambda_a,0)+\phi_a^\intercal\tilde{X}\right]\\ &= V\mathbf{C}_a+V\phi_a. \end{aligned} \tag{B.32}$$

B.3.3 Variance:

The scalar covariance of the a'th and b'th elements of predictive output vector $f(\tilde{X})$ is:

$$\begin{aligned}\mathbf{V}^{ab} &\doteq \mathbb{C}_{\tilde{X}}\left[f_a(\tilde{X}),\, f_b(\tilde{X})\right]\\ &= \mathbb{C}_{\tilde{X}}\left[\mathbb{E}_f\left[f_a(\tilde{X})\right],\, \mathbb{E}_f\left[f_b(\tilde{X})\right]\right]+\mathbb{E}_{\tilde{X}}\left[\mathbb{C}_f\left[f_a(\tilde{X}),\, f_b(\tilde{X})\right]\right]\\ &= \mathbb{C}_{\tilde{X}}\left[s_a^2\beta_a^\intercal q(X,\tilde{X},\Lambda_a,0)+\phi_a^\intercal\tilde{X},\, s_b^2\beta_b^\intercal q(X,\tilde{X},\Lambda_b,0)+\phi_b^\intercal\tilde{X}\right]+\\ &\quad \delta_{ab}\mathbb{E}_{\tilde{X}}\left[s_a^2-k_a(\tilde{X},X)(K_a+\Sigma_a^\varepsilon)^{-1}k_a(X,\tilde{X})\right]\\ &= s_a^2s_b^2\left[\beta_a^\intercal(Q_{ab}-q_aq_b^\intercal)\beta_b+\right.\\ &\quad \left.\delta_{ab}\left(s_a^{-2}-\mathrm{tr}((K_a+\Sigma_a^\varepsilon)^{-1}Q^{aa})\right)\right]+\mathbf{C}_a^\intercal V\phi_b+\phi_a^\intercal V\mathbf{C}_b+\phi_a^\intercal V\phi_b, \end{aligned} \tag{B.33}$$

where

$$Q_{ab}^{ij} \doteq Q\left(X_i,X_j,\Lambda_a,\Lambda_b,0,m,V\right), \tag{B.34}$$

and training inputs are X, outputs are y_a and K_a is a Gram matrix.

B.4 Predictions from Hierarchically Uncertain Inputs

Prediction from hierarchically-uncertain inputs is useful for filtered control and dynamics predictions for the random belief state during system-prediction. Consider making predictions from $a=1,\ldots,\mathrm{X}$ GPs at $\tilde{X}$ with *hierarchical* specification,

$$p(\tilde{X}) \sim \mathcal{N}(M,V), \text{ and } M \sim \mathcal{N}(\mu,\Sigma), \tag{B.35}$$

or equivalently the joint

$$p\left(\begin{bmatrix}\tilde{X}\\ M\end{bmatrix}\right) \sim \mathcal{N}\left(\begin{bmatrix}\mu\\ \mu\end{bmatrix},\begin{bmatrix}\Sigma+V & \Sigma\\ \Sigma & \Sigma\end{bmatrix}\right). \tag{B.36}$$

Considering the input distribution can be expressed as a hierarchy of uncertainties, it should be little surprise that the resultant predictive output distribution can also:

$$f_a(\tilde{X}) \sim \mathcal{N}\left(\mathbf{M}, \bar{\mathbf{V}}\right), \text{ and } \mathbf{M} \sim \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right). \tag{B.37}$$

B.4.1 Mean of the Mean:

The scalar expectation of the a'th element of predictive output mean vector $\mathbf{M}$ (defined (B.28)) w.r.t. the random input mean vector M is:

$$\begin{aligned}
\boldsymbol{\mu}^a &\doteq \mathbb{E}_M[\mathbf{M}^a] \\
&= \int \mathbf{M}^a \mathcal{N}(M;\mu,\Sigma)\,\mathrm{d}M \\
&= s_a^2 \beta_a^\intercal \int q(X,M,\Lambda_a,V)\mathcal{N}(M;\mu,\Sigma)\,\mathrm{d}M + \phi_a^\intercal \mu \\
&= s_a^2 \beta_a^\intercal \hat{q}_a + \phi_a^\intercal \mu, \qquad \text{(B.38)} \\
\hat{q}_a^i &\doteq q(X_i,\mu,\Lambda_a,\Sigma+V). \qquad \text{(B.39)}
\end{aligned}$$

B.4.2 Covariance of the Mean:

Let us first define, using identity (B.9):

$$\begin{aligned}
\bar{\mathbf{C}}_a &\doteq \Sigma^{-1}\mathbb{C}_M\left[M,\, s_a^2\beta_a^\intercal q_a\right] \\
&= \Sigma^{-1}\int (M-\mu) s_a^2 \beta_a^\intercal q(X,M,\Lambda_a,V)\mathcal{N}(M;\mu,\Sigma)\,\mathrm{d}M \\
&= s_a^2(\Lambda_a+\Sigma+V)^{-1}(X-\mu)\beta_a^\intercal \hat{q}_a. \qquad \text{(B.40)}
\end{aligned}$$

The covariance vector of the input mean vector M with the a'th element of predictive output mean vector $\mathbf{M}$ (defined (B.28)) is thus:

$$\begin{aligned}
\mathbb{C}_M[M,\, \mathbf{M}^a] &= \mathbb{C}_M\left[M,\, s_a^2\beta_a^\intercal q_a + \phi_a^\intercal M\right] \\
&= \Sigma\bar{\mathbf{C}}_a + \Sigma\phi_a. \qquad \text{(B.41)}
\end{aligned}$$

B.4.3 Variance of the Mean:

The scalar covariance of the a'th and b'th elements of predictive output mean vector $\mathbf{M}$ (defined (B.28)) is:

$$\begin{aligned}
\boldsymbol{\Sigma}^{ab} &\doteq \mathbb{C}_M\left[\mathbf{M}^a,\, \mathbf{M}^b\right] \\
&= \int (\mathbf{M}^a - \boldsymbol{\mu}^a)(\mathbf{M}^b - \boldsymbol{\mu}^b)\mathcal{N}(M;\mu,\Sigma)\,\mathrm{d}M \\
&= \int \left(s_a^2\beta_a^\intercal(q_a - \hat{q}_a) + \phi_a^\intercal(M-\mu)\right)\left(s_b^2(q_b^\intercal - \hat{q}_b^\intercal)\beta_b + (M^\intercal - \mu^\intercal)\phi_b\right) \\
&\quad \times\mathcal{N}(M;\mu,\Sigma)\,\mathrm{d}M \\
&= s_a^2 s_b^2 \beta_a^\intercal(\hat{Q}_{ab} - \hat{q}_a\hat{q}_b^\intercal)\beta_b + \bar{\mathbf{C}}_a^\intercal\Sigma\phi_b + \phi_a^\intercal\Sigma\bar{\mathbf{C}}_b + \phi_a^\intercal\Sigma\phi_b, \qquad \text{(B.42)} \\
\hat{Q}_{ab}^{ij} &\doteq Q(X_i, X_j, \Lambda_a, \Lambda_b, V, \mu, \Sigma). \qquad \text{(B.43)}
\end{aligned}$$

B.4.4 Mean of the Covariance:

First, let us derive the expectation of (B.31) w.r.t. the random input mean vector M, and using identity (B.10):

$$\begin{aligned}
\mathbb{E}_M[\mathbf{C}_a] &= \int \mathbf{C}_a\mathcal{N}(M;\mu,\Sigma)\,\mathrm{d}M \\
&= \int s_a^2(\Lambda_a + V)^{-1}(X-M)\beta_a^\intercal q(X,M,\Lambda_a,V)\mathcal{N}(M;\mu,\Sigma)\,\mathrm{d}M \\
&= s_a^2(\Lambda_a + \Sigma + V)^{-1}(X-\mu)\beta_a^\intercal\hat{q}_a \\
&= \bar{\mathbf{C}}_a. \qquad \text{(B.44)}
\end{aligned}$$

Now, the expectation of the covariance of input vector $\tilde{X}$ with the a'th element of predictive output vector $f(\tilde{X})$ (defined (B.31)), w.r.t. the random input mean vector M, is thus:

$$\begin{aligned}
\mathbb{E}_M\left[\mathbb{C}_{\tilde{X}}\left[\tilde{X},\, f_a(\tilde{X})\right]\right] &= \mathbb{E}_M[V\mathbf{C}_a + V\phi_a] \\
&= V\bar{\mathbf{C}}_a + V\phi_a. \qquad \text{(B.45)}
\end{aligned}$$

B.4.5 MEAN OF THE VARIANCE:

The scalar expectation of the belief-variance $\mathbf{V}_{ab}$ defined (B.33), w.r.t. the random input mean vector M, using (B.12), is:

$$\begin{aligned}
\bar{\mathbf{V}}^{ab} &\doteq \mathbb{E}_M[\mathbf{V}_{ab}] \\
&= \int \mathbf{V}_{ab}\mathcal{N}(M;\mu,\Sigma)\,\mathrm{d}M \\
&= \int \Big(s_a^2 s_b^2 \big[\beta_a^\intercal (Q_{ab} - q_a q_b^\intercal)\beta_b + \delta_{ab}\big(s_a^{-2} - \mathrm{tr}((K_a + \Sigma_a^\varepsilon)^{-1} Q_{aa})\big)\big] \\
&\quad + \mathbf{C}_a^\intercal V \phi_b + \phi_a^\intercal V \mathbf{C}_b + \phi_a^\intercal V \phi_b \Big) \mathcal{N}(M;\mu,\Sigma)\,\mathrm{d}M \\
&= s_a^2 s_b^2 \big[\beta_a^\intercal (\tilde{Q}_{ab} - \hat{Q}_{ab})\beta_b + \delta_{ab}\big(s_a^{-2} - \mathrm{tr}((K_a + \Sigma_a^\varepsilon)^{-1} \tilde{Q}_{aa})\big)\big] \\
&\quad + \bar{\mathbf{C}}_a^\intercal V \phi_b + \phi_a^\intercal V \bar{\mathbf{C}}_b + \phi_a^\intercal V \phi_b, \qquad \text{(B.46)} \\
\tilde{Q}_{ab}^{ij} &= Q(X_i, X_j, \Lambda_a, \Lambda_b, 0, \mu, \Sigma + V). \qquad \text{(B.47)}
\end{aligned}$$

APPENDIX C

BAYESIAN OPTIMISATION

C.1 PROBABILITY OF IMPROVEMENT

The probability of improvement, a Bayesian optimisation algorithm discussed § 4.3.2, is defined here with accompanying derivatives. Let the (possibly uncertain) cumulative-cost of the best performing controller executed so far be parameterised:

$$\mathcal{C}^* \sim \mathcal{N}\left(\mu_*^{\mathcal{C}}, \Sigma_*^{\mathcal{C}}\right). \tag{C.1}$$

We would like to choose a new parameterisation, ψ, in such a way that it maximises the *probability of improvement* (PI) of the cumulative-cost. However, for consistency of *minimising* objective functions, we shall instead minimise the negative of the PI. For arbitrary ψ we have cumulative-cost distribution $\mathcal{C}^\psi \sim \mathcal{N}\left(\mu^{\mathcal{C}}, \Sigma^{\mathcal{C}}\right)$. What is the probability of improvement $P(\mathcal{C}^\psi < \mathcal{C}^*)$? Let $\Delta\mathcal{C} \doteq \mathcal{C}^\psi - \mathcal{C}^*$. Note:

$$\Delta\mathcal{C} \sim \mathcal{N}\left(\mu^{\mathcal{C}} - \mu_*^{\mathcal{C}},\ \Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}} - 2c\right), \tag{C.2}$$

where c is the covariance between $\mathcal{C}^\psi$ and $\mathcal{C}^*$. Let's approximate $c = 0$ for simplicity. So now the probability of improvement, by changing parameterisation from $*$ to ψ

is:

$$\begin{aligned} PI(\mathcal{C}^\psi) &= -p\left(\mathcal{C}^\psi < \mathcal{C}^*\right) \\ &= -p\left(\Delta\mathcal{C} < 0\right) \\ &= -\int_{-\infty}^{0} \mathcal{N}\left(x; \mu^{\mathcal{C}} - \mu_*^{\mathcal{C}}, \Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}\right) dx \\ &= \Phi(z) - 1, \qquad \text{where} \qquad \text{(C.3)} \\ z &\doteq \frac{\mu^{\mathcal{C}} - \mu_*^{\mathcal{C}}}{\sqrt{\Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}}}, \qquad \text{(C.4)} \end{aligned}$$

where $\Phi(\cdot)$ is the cumulative standard normal function. The gradients are

$$\begin{aligned} \frac{\mathrm{d}PI}{\mathrm{d}\mu^{\mathcal{C}}} &= \frac{1}{\sqrt{\Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}}} \phi(z), \qquad \text{(C.5)} \\ \frac{\mathrm{d}PI}{\mathrm{d}\Sigma^{\mathcal{C}}} &= -\frac{1}{2(\Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}})} z\phi(z), \qquad \text{(C.6)} \end{aligned}$$

where $\phi(\cdot)$ is the standard normal distribution.

C.2 Expected Improvement

As before, let the cumulative-cost of the best performing controller executed so far be distributed:

$$\mathcal{C}^* \sim \mathcal{N}\left(\mu_*^{\mathcal{C}}, \Sigma_*^{\mathcal{C}}\right). \qquad \text{(C.7)}$$

We would like to choose a new parameterisation, ψ, in such a way that it optimises the *expected improvement* (EI) of the cumulative-cost. An 'improvement' means a decrease in cost. For arbitrary ψ we again have cumulative-cost distribution $\mathcal{C}^\psi \sim \mathcal{N}\left(\mu^{\mathcal{C}}, \Sigma^{\mathcal{C}}\right)$, and improvement $\Delta\mathcal{C} \doteq \mathcal{C}^\psi - \mathcal{C}^*$ where

$$\Delta\mathcal{C} \sim \mathcal{N}\left(\mu^{\mathcal{C}} - \mu_*^{\mathcal{C}}, \Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}} - 2c\right), \qquad \text{(C.8)}$$

with c again approximated as zero. So now the expected improvement if we were to change a controller's parameterisation from $*$ to ψ would be:

$$\begin{aligned} EI(\mathcal{C}^{\psi}) &= \mathbb{E}_{\mathcal{C}^{\psi}}\left[\min(\Delta\mathcal{C},\, 0)\right] \\ &= \int_{-\infty}^{0} x\mathcal{N}\left(x; \mu^{\mathcal{C}} - \mu_*^{\mathcal{C}}, \Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}\right) dx \\ &= \Phi(-z)(\mu^{\mathcal{C}} - \mu_*^{\mathcal{C}}) - \phi(z)\sqrt{\Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}}, \qquad \text{where} \qquad \text{(C.9)} \\ z &\doteq \frac{\mu^{\mathcal{C}} - \mu_*^{\mathcal{C}}}{\sqrt{\Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}}}. \qquad \text{(C.10)} \end{aligned}$$

where $\phi(\cdot)$ is the standard normal distribution, $\Phi(\cdot)$ its cumulative standard normal function. The gradients are:

$$\begin{aligned} \frac{\mathrm{d}EI}{\mathrm{d}\mu^{\mathcal{C}}} &= -\frac{\partial z}{\partial \mu^{\mathcal{C}}}\phi(z)(\mu^{\mathcal{C}} - \mu_*^{\mathcal{C}}) + \Phi(-z) + \frac{\partial z}{\partial \mu^{\mathcal{C}}} z\phi(z)\sqrt{\Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}} \\ &= \Phi(-z), \qquad \text{(C.11)} \\ \frac{\mathrm{d}EI}{\mathrm{d}\Sigma^{\mathcal{C}}} &= -\frac{\partial z}{\partial \Sigma^{\mathcal{C}}}\phi(z)(\mu^{\mathcal{C}} - \mu_*^{\mathcal{C}}) + \frac{\partial z}{\partial \Sigma^{\mathcal{C}}} z\phi(z)\sqrt{\Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}} - \frac{\phi(z)}{2\sqrt{\Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}}} \\ &= -\frac{\phi(z)}{2\sqrt{\Sigma^{\mathcal{C}} + \Sigma_*^{\mathcal{C}}}}. \qquad \text{(C.12)} \end{aligned}$$

APPENDIX D

EXPERIMENTAL SETUPS

D.1 CART POLE SWING UP

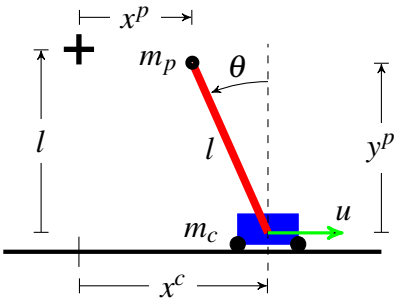


Fig. D.1 The cartpole swing-up task.

For the cartpole swing-up task, the ground truth equations of motion are:

$$\dot{x} \doteq \frac{\mathrm{d}x}{\mathrm{d}t} = \begin{bmatrix} \dot{x^c} \\ \dot{\theta} \\ \frac{-2m_p l\dot{\theta}^2 s + 3m_p gsc + 4u - 4b\dot{x^c}}{4(m_c+m_p) - 3m_p c^2} \\ \frac{-3m_p l\dot{\theta}^2 sc + 6(m_c+m_p)gs + 6(u-b\dot{x^c})c}{4l(m_c+m_p) - 3m_p lc^2} \end{bmatrix}, \tag{D.1}$$

where s and c are shorthand for $\sin(\theta)$ and $\cos(\theta)$ respectively.

A saturating cost function is used: $1 - \exp\left(-\frac{1}{2}d^2/\lambda_c^2\right)$ where d^2 is the squared Euclidean distance between the pendulum's end point (x_p, y_p) and its goal $(0, l)$.

In Chapter 3 our state includes the previous control: $x_t = [u_{t-1}, x_t^c, \theta_t, \dot{x}_t^c, \dot{\theta}_t]^\intercal$.

Table D.1 The cartpole parameter values, Chapter 3

Parameter	Symbol	Value
friction	b	0.1Nsm^{-1}
number episodes per exp.	E	10
gravity	g	9.82ms^{-2}
pole length	l	0.2m
cart mass	m_c	0.5kg
pole mass	m_p	0.5kg
# controller RBF centroids	R	100
time discretisation	Δt	0.0333s
time horizon (# timesteps)	T	60
time horizon (seconds)	(none)	2.0s
max force	$u_{\max}$	10N
min force	$u_{\min}$	-10N
# control variables	U	1
# state variables	X	5
discount factor	γ	1
mean initial state	μ_0^x	$[0\text{m}, \pi\text{rad}, 0\text{m/s}, 0\text{rad/s}]^\intercal$
std. dev. initial state	$(\Sigma_0^x)^{1/2}$	diag([0.2m,0.2rad,0.2m/s,0.2rad/s])
std. dev. observation	$(\Sigma_y^\varepsilon)^{1/2}$	diag([0.03m,0.03rad,0.9m/s,0.9rad/s])
cost lengthscale	λ_c	0.25m
sensor lag	τ_{lag}	0.005s

In Chapter 5 we use the state representation: $x = [x_t^c, \theta_t, \dot{x}_t^c, \dot{\theta}_t]^\top$.

Table D.2 The cartpole parameter values, Chapter 5

Parameter	Symbol	Value
friction	b	0.1Nsm^{-1}
number episodes per exp.	E	100
gravity	g	9.82ms^{-2}
pole length	l	0.6m
cart mass	m_c	0.5kg
pole mass	m_p	0.5kg
# particles (fit f)	P	100
# particles (opt. π)	P	10
# controller RBF centroids	R	50
time discretisation	Δt	0.1s
time horizon (# timesteps)	T	25
time horizon (seconds)	(none)	2.5s
max force	$u_{\max}$	10N
min force	$u_{\min}$	-10N
# control variables	U	1
# state variables	X	4
discount factor	γ	1
mean initial state	μ_0^x	$[0\text{m}, \pi\text{rad}, 0\text{m/s}, 0\text{rad/s}]^\top$
std. dev. initial state	$(\Sigma_0^x)^{1/2}$	diag([0.2m,0.2rad,0.2m/s,0.2rad/s])
std. dev. observation	$(\Sigma_y^\varepsilon)^{1/2}$	diag([0.001m,0.001rad,0.001m/s,0.001rad/s])
cost lengthscale	λ_c	0.25m
sensor lag	τ_{lag}	0s
dropout probability	p_{dropout}	0.05

D.2 Cart Double-Pole Swing Up

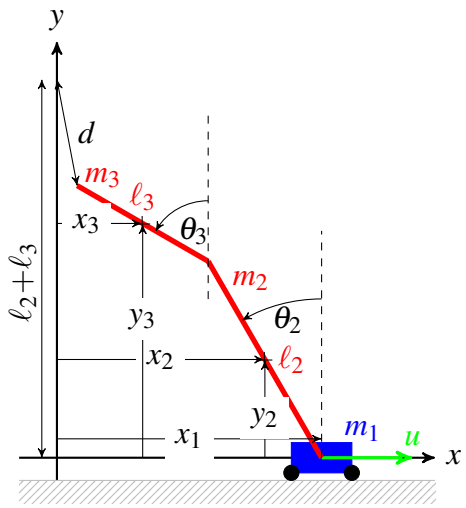


Fig. D.2 The cart-double-pole swing-up task.

In Chapter 4 we use the state: $x^\intercal = [\dot{x}^c, \dot{\theta}_2, \dot{\theta}_3, x^c, \theta_2, \theta_3]$. The equations of motion for the cart double pole are:

$$\dot{x} \doteq \frac{\mathrm{d}x}{\mathrm{d}t} = [\ddot{x}_c, \ddot{\theta}_2, \ddot{\theta}_3, \dot{x}_c, \dot{\theta}_2, \dot{\theta}_3]^\intercal, \tag{D.2}$$

where

$$\begin{bmatrix} \ddot{x}_c \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} = \begin{bmatrix} 2(m_1+m_2+m_3) & -(m_2+2m_3)l_2\cos(\theta_2) & -m_3 l_3 \cos(\theta_3) \\ -(3m_2+6m_3)\cos(\theta_2) & (2m_2+6m_3)l_2 & 3m_3 l_3 \cos(\theta_2-\theta_3) \\ -3\cos(\theta_3) & 3l_2\cos(\theta_2-\theta_3) & 2l_3 \end{bmatrix}^{-1} \times \begin{bmatrix} 2u - 2b\dot{x}^c - (m_2+2m_3)l_2\dot{\theta}_2^2\sin(\theta_2) - m_3 l_3 \dot{\theta}_3^2 \sin(\theta_3) \\ (3m_2+6m_3)g\sin(\theta_2) - 3m_3 l_3 \dot{\theta}_3^2 \sin(\theta_2-\theta_3) \\ 3l_2\dot{\theta}_2^2\sin(\theta_2-\theta_3) + 3g\sin(\theta_3) \end{bmatrix}. \tag{D.3}$$

We again use a saturating cost function: $1-\exp(-\frac{1}{2}d^2/\lambda_c^2)$ except with $\lambda_c = 1.0m$ and d^2 is the squared Euclidean distance between the pendulum's end point and its goal $(0, l_2+l_3)$.

Table D.3 The cart-double-pole parameter values, Chapter 4

Parameter	Symbol	Value
friction	b	0.1Ns/m
number episodes per exp.	E	20
gravity	g	9.82ms^{-2}
moment inertia first pole	I_2	0.015 kg m^2
moment inertia first pole	I_3	0.015 kg m^2
first pole length	l_2	0.6m
second pole length	l_3	0.6m
cart mass	m_1	0.5kg
first pole mass	m_2	0.5kg
second pole mass	m_3	0.5kg
# controller RBF centroids	R	200
time discretisation	Δt	0.05s
time horizon (# timesteps)	T	30
time horizon (seconds)	(none)	1.5s
max force	$u_{\max}$	20N
min force	$u_{\min}$	-20N
# control variables	U	1
# state variables	X	6
discount factor	γ	1
mean initial state	μ_0^x	$[0\text{m/s}, 0\text{rad/s}, 0\text{rad/s}, 0\text{m}, \pi\text{rad}, \pi\text{rad}]^\intercal$
std. dev. initial state	$(\Sigma_y^\varepsilon)^{1/2}$	diag([0m/s, 0rad/s, 0rad/s, 0m, 0rad, 0rad])
std. dev. observation	$\sqrt{\text{diag}(\Sigma_y^\varepsilon)}$	diag([0.02m/s, 0.0349rad/s, 0.0349rad/s, 0.001m, 0.00175rad, 0.00175rad])
cost lengthscale	λ_c	1.0m
sensor lag	τ_{lag}	0s

APPENDIX E

PROOFS

E.1 MOMENTS OF EXPONENTIAL FUNCTIONS OF GAUSSIANS

This section derives the identities of Appendix A.2.5 in 1D. Let $X \sim \mathcal{N}\left(\mu, \sigma^2\right)$ and $z \sim \mathcal{N}(0,1)$,

$$
\begin{aligned}
\mathbb{E}_X\left[\exp(aX)\right] &= \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{a(\mu+\sigma z)} e^{-z^2/2} \mathrm{d}z \\
&= e^{a\mu + a^2\sigma^2/2} \cdot \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-(z-a\sigma)^2/2} \mathrm{d}z \\
&= e^{a\mu + a^2\sigma^2/2}, \\
\mathbb{E}_X\left[\exp\left(-\tfrac{1}{2}(X-b)^\mathsf{T} B (X-b)\right)\right] &= \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-(\mu+\sigma z-b)^\mathsf{T} B (\mu+\sigma z-b)/2} e^{-z^2/2} \mathrm{d}z \\
&= e^{-(\mu-b)^2 B/2} \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\left(z^2(\sigma^2 B+1) + 2z\sigma B(\mu-b)\right)/2} \mathrm{d}z \\
&= e^{-\left((\mu-b)^2 B - \frac{\sigma^2 B^2 (\mu-b)^2}{\sigma^2 B+1}\right)/2} \cdot \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-(\sigma^2 B+1)\left(z + \frac{\sigma B(\mu-b)}{\sigma^2 B+1}\right)^2/2} \mathrm{d}z \\
&= e^{-(\mu-b)^2 B\left(1 - \frac{\sigma^2 B}{\sigma^2 B+1}\right)/2} \cdot (\sigma^2 B+1)^{-1/2} \\
&= e^{-(\mu-b)^2 B\left(\frac{1}{\sigma^2 B+1}\right)/2} \cdot (\sigma^2 B+1)^{-1/2}, \\
\mathbb{V}_X\left[\exp\left(-\tfrac{1}{2}(X-b)^\mathsf{T} B (X-b)\right)\right] &= \mathbb{E}\left[\exp(-(X-b)^\mathsf{T} B (X-b))\right] - \mathbb{E}\left[\cdot\right]^2 \qquad \text{(i.e. } B \to 2B\text{)} \\
&= e^{-(\mu-b)^2 2B\left(\frac{1}{\sigma^2 2B+1}\right)/2} \cdot (\sigma^2 2B+1)^{-1/2} - \mathbb{E}\left[\cdot\right]^2, \\
\mathbb{C}_X\left[X, \exp\left(-\tfrac{1}{2}(X-b)^\mathsf{T} B (X-b)\right)\right] &= \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} (\mu+\sigma z) e^{-(\mu+\sigma z-b)^\mathsf{T} B (\mu+\sigma z-b)/2} e^{-z^2/2} \mathrm{d}z,
\end{aligned}
\tag{E.1}
$$

where $\mathbb{E}\left[\cdot\right]$ is shorthand for (E.1).

E.2 Moments of Trigonometric Functions of Gaussians

This section derives the identities of Appendix A.2.5 in 1D. Let $X \sim \mathcal{N}(\mu, \sigma^2)$ and $z \sim \mathcal{N}(0,1)$,

$$
\begin{aligned}
\mathbb{E}_X[\cos(aX) + i\sin(aX)] &= \mathbb{E}\left[e^{iaX}\right] \\
&= \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ia(\mu+\sigma z)} e^{-z^2/2} \mathrm{d}z \\
&= e^{ia\mu - a^2\sigma^2/2} \cdot \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-(z-ia\sigma)^2/2} \mathrm{d}z \\
&= e^{-a^2\sigma^2/2}(\cos(a\mu) + i\sin(a\mu)), \\
\mathbb{V}_X[\sin(aX)] &= \mathbb{E}\left[\sin^2(aX)\right] - \mathbb{E}[\sin(aX)]^2 \\
&= \tfrac{1}{2}\mathbb{E}[1 - \cos(2aX)] - \mathbb{E}[\sin(aX)]^2 \\
&= \tfrac{1}{2}(1 - e^{-2a^2\sigma^2}\cos(2a\mu)) - e^{-a^2\sigma^2}\sin^2(a\mu) \\
&= \tfrac{1}{2}(1 - e^{-2a^2\sigma^2}\cos(2a\mu) - e^{-a^2\sigma^2}(1 - \cos(2a\mu))) \\
&= \tfrac{1}{2}(1 + e^{-a^2\sigma^2}\cos(2a\mu))(1 - e^{-a^2\sigma^2}), \\
\mathbb{V}_X[\cos(aX)] &= \mathbb{E}\left[\cos^2(aX)\right] - \mathbb{E}[\cos(aX)]^2 \\
&= \tfrac{1}{2}\mathbb{E}[1 + \cos(2aX)] - \mathbb{E}[\cos(aX)]^2 \\
&= \tfrac{1}{2}(1 + e^{-2a^2\sigma^2}\cos(2a\mu)) - e^{-a^2\sigma^2}\cos^2(a\mu) \\
&= \tfrac{1}{2}(1 + e^{-2a^2\sigma^2}\cos(2a\mu) - e^{-a^2\sigma^2}(1 + \cos(2a\mu)) \\
&= \tfrac{1}{2}(1 - e^{-a^2\sigma^2}\cos(2a\mu))(1 - e^{-a^2\sigma^2}), \\
\mathbb{C}_X[\sin(aX), \cos(aX)] &= \mathbb{E}[\sin(aX)\cos(aX)] - \mathbb{E}[\sin(aX)]\,\mathbb{E}[\cos(aX)] \\
&= \tfrac{1}{2}\mathbb{E}[\sin(2aX)] - \mathbb{E}[\sin(aX)]\,\mathbb{E}[\cos(aX)] \\
&= \tfrac{1}{2}(e^{-2a^2\sigma^2}\sin(2a\mu)) - e^{-a^2\sigma^2}\sin(a\mu)\cos(a\mu) \\
&= -\tfrac{1}{2}e^{-a^2\sigma^2}\sin(2a\mu)(1 - e^{-a^2\sigma^2}), \\
\mathbb{C}_X[X, \cos(X) + i\sin(X)] &= \mathbb{E}\left[Xe^{iX}\right] - \mathbb{E}[X]\,\mathbb{E}\left[e^{iX}\right] \\
&= \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} (\mu + \sigma z) e^{i(\mu+\sigma z) - z^2/2} \mathrm{d}z - \mu\mathbb{E}\left[e^{iX}\right] \\
&= \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} (\mu + \sigma z) e^{-(z-i\sigma)^2/2} e^{i\mu - \sigma^2/2} \mathrm{d}z - \mu\mathbb{E}\left[e^{iX}\right] \\
&= e^{i\mu-\sigma^2/2} \tfrac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} (\mu + \sigma(\hat{z} + i\sigma)) e^{-\hat{z}^2/2} d\hat{z} - \mu\mathbb{E}\left[e^{iX}\right], \quad \hat{z} = z - i\sigma \\
&= e^{i\mu-\sigma^2/2}(\mu + i\sigma^2) - \mu\mathbb{E}\left[e^{iX}\right] \\
&= \mathbb{E}\left[e^{iX}\right](\mu + i\sigma^2 - \mu) \\
&= i\sigma^2\mathbb{E}\left[e^{iX}\right].
\end{aligned}
$$

using the trigonometric identities

$$\begin{aligned}\sin(2x) &= 2\sin(x)\cos(x),\\ \sin^2(x) &= \tfrac{1}{2}(1-\cos(2x)),\\ \cos^2(x) &= \tfrac{1}{2}(1+\cos(2x)).\end{aligned}$$

E.3 Matrix Derivatives

This section derives some identities of Appendix A.4

$$\begin{aligned}\frac{\partial}{\partial x_{ij}}\log\det(X) &= \frac{1}{\det(X)}\frac{\partial}{\partial x_{ij}}\det(X)\\ &= \frac{1}{\det(X)}\operatorname{adj}(X)_{ji}\\ &= (X^{-1})_{ji}.\end{aligned}$$

$$\begin{aligned}U^{-1}U &= I,\\ \partial(U^{-1}U) &= 0,\\ \partial(U^{-1})U+U^{-1}\partial(U) &= 0,\\ \partial(U^{-1}) &= U^{-1}\partial(U)U^{-1}.\end{aligned}$$

E.4 Gaussian Process Prediction

This section proves (B.9) and (B.10).

$$\begin{aligned}\text{integral} &\doteq \int (y-t)q(x,t,\Lambda,V)\mathcal{N}(t;\mu,\Sigma)\,\mathrm{d}t\\ &= (2\pi)^{\frac{D}{2}}|\Lambda|^{\frac{1}{2}}\int (y-t)\mathcal{N}(t;x,\Lambda+V)\mathcal{N}(t;\mu,\Sigma)\,\mathrm{d}t\\ &= (2\pi)^{\frac{D}{2}}|\Lambda|^{\frac{1}{2}}\mathcal{N}(x;\mu,\Lambda+V+\Sigma)\int (y-t)\mathcal{N}(t;z,Z)\,\mathrm{d}t\\ &= q(x,\mu,\Lambda,V+\Sigma)(y-z)\\ &= q(x,\mu,\Lambda,V+\Sigma)(y-\Sigma[\Lambda+V+\Sigma]^{-1}x-(\Lambda+V)[\Lambda+V+\Sigma]^{-1}\mu),\end{aligned}$$

where

$$Z \doteq [(\Lambda+V)^{-1}+\Sigma^{-1}]^{-1},$$

and

$$\begin{aligned} z &\doteq Z[(\Lambda+V)^{-1}x+\Sigma^{-1}\mu] \\ &= \Sigma[\Lambda+V+\Sigma]^{-1}x+(\Lambda+V)[\Lambda+V+\Sigma]^{-1}\mu. \end{aligned}$$

Now if $y=\mu$ then

$$\text{integral} = q(x,\mu,\Lambda,V+\Sigma)\Sigma(\Lambda+V+\Sigma)^{-1}(\mu-x),$$

else if $y=x$ then

$$\text{integral} = q(x,\mu,\Lambda,V+\Sigma)(\Lambda+V)(\Lambda+V+\Sigma)^{-1}(x-\mu).$$

E.5 Hierarchical Cost Moments

This section derives some identities of the hierarchical cost in § 4.5.3. A previous result from Deisenroth and Rasmussen (2011) of saturating costs given normally distributed inputs $X \sim \mathcal{N}(\mu^x,\Sigma^x)$, the cost-mean and cost-variance are respectively computed as

$$\begin{aligned} m^c \doteq \mathbb{E}_X[\text{cost}(X;x^*,\Lambda_c)] &= 1-\det(I+\Sigma^x\Lambda_c^{-1})^{-1/2} \\ &\quad \times \exp\left(-\tfrac{1}{2}(\mu^x-x^*)^\intercal\Lambda_c^{-1}(I+\Sigma^x\Lambda_c^{-1})^{-1}(\mu^x-x^*)\right), \end{aligned} \tag{E.2}$$

$$\begin{aligned} V^c \doteq \mathbb{V}_X[\text{cost}(X;x^*,\Lambda_c)] &= -(m^c-1)^2+\det(I+2\Sigma^x\Lambda_c^{-1})^{-1/2} \\ &\quad \times \exp\left(-(\mu^x-x^*)^\intercal\Lambda_c^{-1}(I+2\Sigma^x\Lambda_c^{-1})^{-1}(\mu^x-x^*)\right). \end{aligned} \tag{E.3}$$

Now given a hierarchically distributed variable $X \sim \mathcal{N}(M^x,V^x)$ and $M^x \sim \mathcal{N}(\mu^x,\Sigma^x)$ as input, and using shorthand $\hat{\Lambda}_c = (I+V^x\Lambda_c^{-1})\Lambda_c$, then the expected of the cost-mean

is:

$$\begin{aligned}
\mu^c &\doteq \mathbb{E}_{M^x}[M^c] \\
&= \mathbb{E}_{M^x}\left[1-\det(I+V^x\Lambda_c^{-1})^{-\frac{1}{2}}\exp\left(-\tfrac{1}{2}(M^x-x^*)^\intercal\hat{\Lambda}_c^{-1}(M^x-x^*)\right)\right] \\
&= 1-\det(I+V^x\Lambda_c^{-1})^{-\frac{1}{2}}\left(1-\mathbb{E}_{M^x}\left[\text{cost}(M^x;x^*,\hat{\Lambda}_c)\right]\right) \\
&= 1-\det\left((I+V^x\Lambda_c^{-1})(I+\Sigma^x\hat{\Lambda}_c^{-1})\right)^{-\frac{1}{2}}\exp\left(-\tfrac{1}{2}(\mu^x-x^*)^\intercal\hat{\Lambda}_c^{-1}(I+\Sigma^x\hat{\Lambda}_c^{-1})^{-1}(\mu^x-x^*)\right) \\
&= 1-\det\left(I+(\Sigma^x+V^x)\Lambda_c^{-1}\right)^{-\frac{1}{2}}\exp\left(-\tfrac{1}{2}(\mu^x-x^*)^\intercal\Lambda_c^{-1}(I+(\Sigma^x+V^x)\Lambda_c^{-1})^{-1}(\mu^x-x^*)\right) \\
&= \mathbb{E}\left[\text{cost}(\mathcal{N}(\mu^x,\Sigma^x+V^x)\,;\,x^*,\Lambda_c)\right], \qquad \text{(E.4)}
\end{aligned}$$

the variance of the cost-mean is:

$$\begin{aligned}
\Sigma^c &\doteq \mathbb{V}_{M^x}[M^c] \\
&= \mathbb{V}_{M^x}\left[\det(I+V^x\Lambda_c^{-1})^{-\frac{1}{2}}\exp\left(-\tfrac{1}{2}(M^x-x^*)^\intercal\hat{\Lambda}_c^{-1}(M^x-x^*)\right)\right] \\
&= \det(I+V^x\Lambda_c^{-1})^{-1}\mathbb{V}_{M^x}\left[\text{cost}(M^x;x^*,\hat{\Lambda}_c)\right], \qquad \text{(E.5)}
\end{aligned}$$

and the average of the cost-variance can be computed from the 'leftover variance':

$$\begin{aligned}
\bar{V}^c &\doteq \mathbb{E}_{M^x}[V^c] \\
&= \mathbb{V}\left[\text{cost}(\mathcal{N}(\mu^x,\Sigma^x+V^x);x^*,\Lambda_c)\right]-\mathbb{V}\left[\text{cost}(\mathcal{N}(\mu^x,\Sigma^x);x^*,\Lambda_c)\right], \qquad \text{(E.6)}
\end{aligned}$$

where all $\mathbb{E}[\text{cost}(\cdot)]$ are computed with (E.2) and $\mathbb{V}[\text{cost}(\cdot)]$ with (E.3).

E.6 Latent Variable Belief-MDPs More General than Belief-MDPs

Here we show how the latent-variable belief-MDP of Fig. 3.13 is in fact a generalisation of belief-MDPs seen Fig. 3.4 since it is not clear by a visual comparison between both PGMs. We show Fig. 3.13 is in fact a generalisation of Fig. 3.4 via demonstrating their equivalence under special constraints. Four constraints are required:

- $\mu_t^x = \mu_t^m$,
- $\Sigma_t^x = \Sigma_{t|t-1}^m + V_{t|t-1}$,
- $\Sigma_{t|t-1}^{mx} = \Sigma_{t|t-1}^m$,
- $f_b = f_x = f$.

Under the above four conditions, the belief $p(B) \sim \mathcal{N}(M,V)$ is a sufficient statistic for state X at all future timesteps, and the process seen *Fig.* 3.13 reduces to *Fig.* 3.4. We show this is first true for the filtering update step, and then for the prediction step.

Under the above constraints, the prior distribution of the system is

$$B_{t|t-1} \sim \mathcal{N}\left(M_{t|t-1}, V_{t|t-1}\right), \tag{E.7}$$

$$H_t = \begin{bmatrix} X_t \\ M_{t|t-1} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu^m_{t|t-1} \\ \mu^m_{t|t-1} \end{bmatrix}, \begin{bmatrix} \Sigma^m_{t|t-1} + V_{t|t-1} & \Sigma^m_{t|t-1} \\ \Sigma^m_{t|t-1} & \Sigma^m_{t|t-1} \end{bmatrix} \right). \tag{E.8}$$

Then, according to the latent-variable belief-MDP update equations ((3.82) – (3.86)) the same variable post update-step become

$$\mu^m_{t|t} = \mu^m_{t|t-1}, \tag{E.9}$$

$$\begin{aligned} \Sigma^m_{t|t} &= \Sigma^m_{t|t-1} + W_y(V_{t|t-1} + \Sigma^\varepsilon_y)W_y^\intercal \\ &= \Sigma^m_{t|t-1} + W_y V_{t|t-1}, \end{aligned} \tag{E.10}$$

$$\begin{aligned} \Sigma^{mx}_{t|t} &= \Sigma^m_{t|t-1} + W_y V_{t|t-1} \\ &= \Sigma^m_{t|t}, \end{aligned} \tag{E.11}$$

$$V_{t|t} = W_m V_{t|t-1}. \tag{E.12}$$

Now notice, the mean is unchanged (E.9) and the total variance is conserved: $\Sigma^m_{t|t} + V_{t|t} = \Sigma^m_{t|t-1} + V_{t|t-1} = \Sigma^x_t$. So up until now we have shown that the update step preserves that fact that both $B_{t|t-1}$ and $B_{t|t}$ are sufficient statistics of $X_{t|t}$.

Next we look at the prediction step, showing under the four constraints, that the hierarchical prediction equations (3.61) – (3.67) result in a new predictive belief $B_{t+1|t}$ that still acts as a sufficient statistic of X_{t+1} which would otherwise need to be computed using PILCO's equations (2.40) – (2.45). We copy the relevant equations below for reader convenience.

First is the predictive means. Since $\Sigma^x_t = \Sigma^m_{t|t-1} + V_{t|t-1}$ and given (E.9), then $q = \hat{q}$ ((E.15), (E.21)), so $\mu^x_{t+1} = \mu^m_{t+1|t}$ ((E.13), (E.18)). Second is the predictive variances. Note $C_{\tilde{x}x'} = C_{\tilde{m}m'}$ ((E.17), (E.24)) and $Q = \tilde{Q}$ ((E.16), (E.23)). Then the addition of both (E.19) and (E.20) (i.e. $\Sigma^m_{t+1|t} + V_{t+1|t}$) cancel their $\hat{Q}$ terms, and together with $\Sigma^m_{t|t-1} + V_{t|t-1} = \Sigma^x_t$, results in the definition of Σ^x_{t+1} found (E.14). So far we have shown the full distribution of $B_{t+1|t}$ is a sufficient statistic of the full distribution of X_{t+1} without the need to explicitly do a prediction step for X_{t+1}.

Predictive state equations $p(X_{t+1})$, copied from (2.40) – (2.45) on page 37

$$\begin{aligned}
\mu^{x,a}_{t+1} &= s_a^2 \beta_a^\intercal q^a + \phi_a^\intercal \mu_t^{\tilde{x}}, && \text{(E.13)}\\
\Sigma^{x,ab}_{t+1} &= s_a^2 s_b^2 \left[\beta_a^\intercal (Q^{ab} - q^a q^{b\intercal}) \beta_b \delta_{ab} \left(s_a^{-2} - \mathrm{tr}((K_a + \Sigma_a^\varepsilon)^{-1} Q^{aa})\right)\right] + \\
&\quad + C^{a\intercal}_{\tilde{x}x'} \Sigma_t^{\tilde{x}} \phi_b + \phi_a^\intercal \Sigma_t^{\tilde{x}} C^b_{\tilde{x}x'} + \phi_a^\intercal \Sigma_t^{\tilde{x}} \phi_b, && \text{(E.14)}\\
q_i^a &\doteq q(X_i, \mu_t^{\tilde{x}}, \Lambda_a, \Sigma_t^{\tilde{x}}), && \text{(E.15)}\\
Q_{ij}^{ab} &\doteq Q\left(X_i, X_j, \Lambda_a, \Lambda_b, 0, \mu_t^{\tilde{x}}, \Sigma_t^{\tilde{x}}\right), && \text{(E.16)}\\
C^a_{\tilde{x}x'} &= s_a^2 (\Lambda_a + \Sigma_t^{\tilde{x}})^{-1} (X - \mu_t^{\tilde{x}}) \beta_a^\intercal q^a. && \text{(E.17)}
\end{aligned}$$

Predictive belief equations $p(B_{t+1|t})$, copied from (3.61) – (3.67) on page 65

$$\begin{aligned}
\mu^{m,a}_{t+1|t} &= s_a^2 \beta_a^\intercal \hat{q}^a + \phi_a^\intercal \mu^{\tilde{m}}_{t|t}, && \text{(E.18)}\\
\Sigma^{m,ab}_{t+1|t} &= s_a^2 s_b^2 \beta_a^\intercal (\hat{Q}^{ab} - \hat{q}^a \hat{q}^{b\intercal}) \beta_b + C^{a\intercal}_{\tilde{m}m'} \Sigma^{\tilde{m}}_{t|t} \phi_b + \phi_a^\intercal \Sigma^{\tilde{m}}_{t|t} C^b_{\tilde{m}m'} + \phi_a^\intercal \Sigma^{\tilde{m}}_{t|t} \phi_b, && \text{(E.19)}\\
V^{ab}_{t+1|t} &= s_a^2 s_b^2 \left[\beta_a^\intercal (\tilde{Q}^{ab} - \hat{Q}^{ab}) \beta_b + \delta_{ab} \left(s_a^{-2} - \mathrm{tr}((K_a + \Sigma_a^\varepsilon)^{-1} \tilde{Q}_{aa})\right)\right] \\
&\quad + C^{a\intercal}_{\tilde{m}m'} \tilde{V}_{t|t} \phi_b + \phi_a^\intercal \tilde{V}_{t|t} C^b_{\tilde{m}m'} + \phi_a^\intercal \tilde{V}_{t|t} \phi_b, && \text{(E.20)}\\
\hat{q}_i^a &\doteq q\left(X_i, \mu^{\tilde{m}}_{t|t}, \Lambda_a, \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t}\right), && \text{(E.21)}\\
\hat{Q}_{ij}^{ab} &\doteq Q(X_i, X_j, \Lambda_a, \Lambda_b, \tilde{V}_{t|t}, \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t}), && \text{(E.22)}\\
\tilde{Q}_{ij}^{ab} &\doteq Q(X_i, X_j, \Lambda_a, \Lambda_b, 0, \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t}), && \text{(E.23)}\\
C^a_{\tilde{m}m'} &\doteq s_a^2 (\Lambda_a + \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t})^{-1} (X - \mu^{\tilde{m}}_{t|t}) \beta_a^\intercal \hat{q}_a. && \text{(E.24)}
\end{aligned}$$

Finally third, we compute the new covariance term, to show the if special structure found in (E.8) is true on one timestep, then is persists to the next timestep, and thereafter by induction. Using the 'exact method' of computing $\mathbb{C}\left[X_{t+1},\, M_{t+1|t}\right]$ from (3.105), we simplify based on our constraints above, including $f_x = f_b$:

$$\begin{aligned}
\mathbb{C}\left[X^k_{t+1},\, M^l_{t+1|t}\right] &= s_k^{x2} s_l^{b2} \left[\beta_k^{x\intercal} (\hat{\hat{Q}}_{kl} - q_k^x q_l^{b\intercal}) \beta_l^b\right] + \hat{C}_k^{x\intercal} \Sigma^z \hat{\phi}_l^b + \hat{\phi}_k^{x\intercal} \Sigma^z \hat{C}_l^b + \hat{\phi}_k^{x\intercal} \Sigma^z \hat{\phi}_l^b \\
&= s_k^{x2} s_l^{b2} \left[\beta_k^{x\intercal} (\hat{\hat{Q}}_{kl} - q_k^x q_l^{b\intercal}) \beta_l^b\right] + C_k^{x\intercal} \Sigma' \phi_l^b + \phi_k^{x\intercal} \Sigma' C_l^b + \phi_k^{x\intercal} \Sigma' \phi_l^b \\
&= s_k^2 s_l^2 \left[\beta_k^\intercal (\hat{Q}_{kl} - \hat{q}_k \hat{q}_l^\intercal) \beta_l\right] + C_k^\intercal \Sigma^{\tilde{m}}_{t|t} \phi_l + \phi_k^\intercal \Sigma^{\tilde{m}}_{t|t} C_l + \phi_k^\intercal \Sigma^{\tilde{m}}_{t|t} \phi_l \\
&= \Sigma^{m,kl}_{t+1|t}, \qquad \text{as per (E.19)}, && \text{(E.25)}
\end{aligned}$$

using and the definition of Q (B.3) together with $\Sigma_t^x = \Sigma_{t|t}^m + V_{t|t}$, as well as since

$$\begin{aligned}\Sigma' &= \begin{bmatrix}\Sigma_{t|t}^{xm} & \Sigma_t^{xu} \\ \Sigma_{t|t}^{um} & \Sigma_t^u\end{bmatrix} = \begin{bmatrix}\Sigma_{t|t}^{xm} & \Sigma_{t|t}^{xm} C_{m_{t|t}u} \\ C_{m_{t|t}u}^\intercal \Sigma_{t|t}^m & \Sigma_u\end{bmatrix} = \begin{bmatrix}\Sigma_{t|t}^m & \Sigma_{t|t}^m C_{m_{t|t}u} \\ C_{m_{t|t}u}^\intercal \Sigma_{t|t}^m & \Sigma_u\end{bmatrix} \\ &= \Sigma_{t|t}^{\tilde{m}},\end{aligned} \tag{E.26}$$

where Σ' was simplified to $\Sigma_{t|t}^{\tilde{m}}$ using (3.98), (3.88), and (E.11).

Thus we have shown, given the four constraints, some reflected in the structure of (E.8), the same structure persists thereafter:

$$H_{t+1} = \begin{bmatrix}X_{t+1} \\ M_{t+1|t}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mu_{t+1|t}^m \\ \mu_{t+1|t}^m\end{bmatrix}, \begin{bmatrix}\Sigma_{t+1|t}^m + V_{t+1|t} & \Sigma_{t+1|t}^m \\ \Sigma_{t+1|t}^m & \Sigma_{t+1|t}^m\end{bmatrix}\right). \tag{E.27}$$

In addition, the definition of the new terms $\Sigma_{t+1|t}^m$ and $V_{t+1|t}$, which were computed by the latent-variable belief-MDP equations, are the same as those computed by the belief-MDP equations. This is easily seen showing the equivalence of the update equations, providing identical inputs to the f_b inputs, follows the same rule for both PGMs. The update variables using latent-variable belief-MDP equations were given (E.9) – (E.12). These are consistent with what is computed by the belief-MDP equations in (3.53) – (3.56). This completes the proof that latent-variable belief-MDPs seen Fig. 3.13 generalise belief-MDPs seen Fig. 3.4 since Fig. 3.13 can be reduced to Fig. 3.4 given the four constraints introduced above.

LIST OF FIGURES

LIST OF TABLES

Nomenclature

Roman Symbols

A	linear(ised) dynamics function component $\partial x_{t+1}/\partial x_t$
b, B	probabilistic belief function of state (capitalised if hierarchically-distributed)
b	(alternate use) a friction / damping coefficicent
B	(alternate use) linear(ised) dynamics function component $\partial x_{t+1}/\partial u_t$
c	instantaneous cost value
$\bar{c}$	expected instantaneous cost given uncertain input X
cost	instantaneous cost function
C	input-output covariance term premultiplied by inverse input variance
C	(alternate use) linear(ised) observation function component $\partial y_t/\partial x_t$
$\mathcal{C}$	cumulative cost
$\mathbb{C}$	covariance operator, defined (A.2)
d	derivative
D	linear(ised) observation function component $\partial y_t/\partial u_t$
$\mathcal{D}$	data
E	number of episodes (trials)
$\hat{E}$	number of episodes (trials) remaining
$\mathbb{E}$	expectation operator, defined (A.1)

f	dynamics function
f_b	dynamics function used for (online) belief monitoring
f_x	dynamics function used for (offline) system simulation
g	observation function (sensor function)
g	(alternate use) acceleration due to gravity
h, H	'hybrid state' concatenating state x and belief-mean m (capitalised if uncertain)
I	identity matrix
J, J^*	loss function for one episode = expected cumulative cost (starred if optimal)
$\mathbf{J}$	summation of all episode's losses
k, K	a Gaussian process covariance vector (capitalised if matrix)
m, M	belief-mean parameters (capitalised if uncertain)
M	number of Gaussian process inducing points
$\tilde{m}, \tilde{M}$	belief-mean parameters m concat'd with control u (capitalised if uncertain)
$\mathcal{N}$	Normal (or Gaussian) distribution, defined (A.8)
N	number of dynamics model training datum
P	number of MC particles representing a distribution
q	a Gaussian process prediction function, defined (B.1)
Q	a Gaussian process prediction function, defined (B.2) and (B.3)
R	number of RBF policy centroids
$\mathbb{R}$	space of real numbers
s	Gaussian process signal standard deviation
t	integer timestep
T	time horizon (as an integer number of discrete timesteps)
u, U	control output (capitalised if uncertain)

U	dimension of control u
vec	function that does columnwise unwrapping of a matrix into a vector
V	belief-variance parameters
$\bar{V}$	expected belief-variance parameters
$\mathbb{V}$	variance operator, defined (A.3)
W_m	filtering update weight matrix associated with bleief mean m, defined (3.39)
W_y	filtering update weight matrix associated with observation y, defined (3.40)
W	concatencated weight matrices $W \doteq [W_y, W_m]$
x, X	system state (capitalised if uncertain)
x^*	a desired goal state
x_c	cart's lateral position
$\hat{x}$	estimated state (a point prediction, not a probabilistic prediction)
$\tilde{x}$, $\tilde{X}$	system state x concatenated with control u (capitalised if uncertain)
X	Gaussian process and dynamics model training input data over all epsiodes
X	dimension of state x
y, Y	observation (capitalised if uncertain)
y	(alternate use) Gaussian process and dynamics model training targets over all epsiodes
Y	dimension of observation y
z, Z	various local uses, including concatenation of $\{x, b, u\}$ (capitalised if uncertain)

Greek Symbols

β	a Gaussian process term: targets premultiplied by inverse covariance matrix
Δt	time discretisation
ε_t^c	random cost noise associated with cost function outputting cost c

ε_t^u random control noise associated with controller/policy π outputting contol u

ε_t^x random process noise associated with system dynamics f outputting state x

ε^{x_0} random initialisation of the system state x

ε_t^y random observation noise associated with sensor g outputting observation y

γ discount factor

λ lengthscale scalar

λ (alternate use) Gittins index of a Gaussian bandit

Λ lengthscale matrix

μ generic mean vector

ϕ Gaussian process linear function

ϕ (alternate use) standard Gaussian distribution function

Φ standard Gaussian cumulative distribution function

π, π^* controller/policy function deciding control u (starred if optimal)

Π controller/policy function linearisation matrix

ψ controller parameters

Ψ space of controller parameters

Σ generic (co)variance matrix

σ generic standard deviation scalar

θ pendulum angle

ξ_t a controlled path: $\{x_t, u_t, \ldots, x_T\}$

Superscripts

p particle index

$\intercal$ matrix transpose

$-\intercal$ inverse of matrix transpose

$*$ optimal quantity, optimal function, or goal state

Subscripts

e current episode e

t current time t

$t:T$ a sequence of quantities indexed in time from t to T inclusive, e.g. $x_{t:T} = \{x_t, x_{t+1}, ..., x_T\}$.

$t|t-1$ prediction of latent system state x at time t given observations from time 0 to time $t-1$ inclusive

Other Symbols

$\odot$ element-wise product

$\mathbb{1}$ indicator function

∂ partial derivative

$[\cdot,\cdot]$ row-wise concatenation matrix operator

$[\cdot;\cdot]$ column-wise concatenation matrix operator

Acronyms / Abbreviations

ADF Assumed Density Filtering

BNN Bayesian Neural Network

BNP Bayesian NonParametric model

BO Bayesian Optimisation

BRL Bayesian Reinforcement Learning

DP Dynamic Programming

EI Expected Improvement

EKF Extended Kalman Filter

FITC Fully Independent Training Conditional

GI Gittins Index

GP, $\mathcal{GP}$ Gaussian Process

HJB Hamilton-Jacobi-Bellman equation

IE Interval Estimation

iid Independent and Identically Distributed

KF Kalman Filter

KL Kullback-Leibler divergence

LQ Linear dynamics function and Quadratic cost function

LQG, iLQG (Iterative) Linear Quadratic Gaussian controller

LQR, iLQR (Iterative) Linear Quadratic Regulator

MAP Maximum A Posteriori

MC Monte Carlo

MDP Markov Decision Process

MPC Model Predictive Control

NN Neural Network

ODE Ordinary Differential Equation

PAC Probably Approximately Correct

PDE Partial Differential Equation

PGM Probabilistic Graphical Model

PI Probability of Improvement

PID Proportional-Integral-Derivative controller

PILCO Probabilistic Inference and Learning for COntrol

PMP Pontryagin's Maximum (or Minimum) Principle

POMDP Partially Observable Markov Decision Process

PSD Positive Semi-Definite matrix

RBF Radial Basis Function

RL Reinforcement Learning

UKF Unscented Kalman Filter

VFE Variational Free Energy

w.r.t. With Respect To

REFERENCES

AAAI Association for the Advancement of Artificial Intelligence
ACM Association for Computing Machinery
ICML International Conference on Machine Learning
ICRA International Conference on Robotics and Automation
IEEE Institute of Electrical and Electronics Engineers
IROS Intelligent RObots and Systems
JMLR Journal of Machine Learning Research
NIPS Neural Information Processing Systems
UAI Uncertainty in Artificial Intelligence

Anderson, B. 1985. Adaptive systems, lack of persistency of excitation and bursting phenomena. *Automatica*, 21(3):247–258.

Adaptive control's potential for instability.

Asmuth, J., Li, L., Littman, M. L., Nouri, A., and Wingate, D. 2009. A Bayesian sampling approach to exploration in reinforcement learning. In *UAI*, pages 19–26.

Astrom, K. 1965. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10(1):174–205.

First POMDP paper.

Aström, K. J. and Murray, R. 2010. *Feedback systems: an introduction for scientists and engineers*. Princeton University Press.

Åström, K. J. and Wittenmark, B. 2013. *Adaptive control*. Courier Corporation.

Atkeson, C. and Santamaria, J. 1997. A comparison of direct and model-based reinforcement learning. In *ICRA*. Citeseer.

Auer, P. 2003. Using confidence bounds for exploitation-exploration trade-offs. *JMLR*, 3:397–422.

Bar-Shalom, Y. and Tse, E. 1974. Dual effect, certainty equivalence, and separation in stochastic control. *IEEE Transactions on Automatic Control*, 19(5):494–500.

Barber, D., Cemgil, T., and Chiappa, S. 2011. *Bayesian time series models*. Cambridge University Press.

Bauer, M., van der Wilk, M., and Rasmussen, C. 2016. Understanding probabilistic sparse Gaussian process approximations. In *NIPS*.

Comparitive analysis of FITC and VFE.

Bellman, R. 1965. *Dynamic programming and modern control theory*. Academic Press New York.

HJB theory.

Bellman, R. and Kalaba, R. E. 1965. *Dynamic programming and modern control theory*, volume 81. Citeseer.

Benson, D. A., Huntington, G. T., Thorvaldsen, T. P., and Rao, A. V. 2006. Direct trajectory optimization and costate estimation via an orthogonal collocation method. *Journal of Guidance, Control, and Dynamics*, 29(6):1435–1440.

Berg, J. V. D., Patil, S., and Alterovitz, R. 2012. Efficient approximate value iteration for continuous Gaussian POMDPs. In *AAAI*.

Berry, D. and Fristedt, B. 1985. *Bandit problems: sequential allocation of experiments*. Springer.

Bishop, C. M. 2006. Pattern recognition. *Machine Learning*, 128:1–58.

Boone, G. 1997. Efficient reinforcement learning: Model-based acrobot control. In *ICRA*, volume 1, pages 229–234.

Brafman, R. and Tennenholtz, M. 2003. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *JMLR*, 3:213–231.

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

Bui, T. D., Yan, J., and Turner, R. E. 2016. A unifying framework for sparse Gaussian process approximation using power expectation propagation. *arXiv preprint arXiv:1605.07066*.

Candela, J., Girard, A., Larsen, J., and Rasmussen, C. 2003. Propagation of uncertainty in bayesian kernel models - application to multiple-step ahead forecasting. In *International Conference on Acoustics, Speech, and Signal Processing.*, volume 2, pages II–701–4 vol.2.

GP prediction with uncertain inputs.

Cassandra, A. and Kaelbling, L. P. 1995. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, page 362.

Csató, L. and Opper, M. 2002. Sparse on-line Gaussian processes. *Neural computation*, 14(3):641–668.

Dallaire, P., Besse, C., Ross, S., and Chaib-draa, B. 2009. Bayesian reinforcement learning in continuous POMDPs with Gaussian processes. In *IROS*, pages 2604–2609.

Dearden, R., Friedman, N., and Russell, S. 1998. Bayesian Q-learning. *National Conference on Artificial Intelligence*, pages 761–768.

Models rewards as iid normal distributions (with normal-gamma priors). Using Q-learning. Action selection uses optimisitic myopic exploration bonus, called value of perfect information, function of immediate reward uncertainty.

Deisenroth, M. 2009. *Efficient Reinforcement Learning Using Gaussian Processes*. KIT Scientific Publishing.

Marc Deisenroth's PhD.

Deisenroth, M., Fox, D., and Rasmussen, C. 2015. Gaussian processes for data-efficient learning in robotics and control. *Pattern Analysis and Machine Intelligence*, 37(2):408–423.

Deisenroth, M. and Peters, J. 2012. Solving nonlinear continuous state-action-observation POMDPs for mechanical systems with Gaussian noise. In *European Workshop on Reinforcement Learning*.

Deisenroth, M., Peters, J., and Rasmussen, C. 2008. Approximate dynamic programming with Gaussian processes. In *American Control Conference*, pages 4480–4485.

Transition function and value function are modelled by GPs. Used in a underactuated pendulum swing-up experiment.

Deisenroth, M. and Rasmussen, C. 2011. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, pages 465–472, New York, NY, USA.

Delage, E. and Mannor, S. 2007. Percentile optimization in uncertain Markov decision processes with application to efficient exploration. In *International Conference on Machine Learning*, pages 225–232. ACM.

Der Kiureghian, A. and Ditlevsen, O. 2009. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112.

Doya, K. 2000. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245.

More HJB theory.

Duff, M. 2002. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst.

Michael Duff's PhD framing reinforcement learning as a POMDP.

Durbin, J. and Koopman, S. J. 2012. *Time series analysis by state space methods*. Number 38. Oxford University Press.

Discusses MLE and Bayesian parameter estimation of grey-box systems.

Engel, Y., Mannor, S., and Meir, R. 2003. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *International Conference on Machine Learning*, pages 154–161.

Engel, Y., Mannor, S., and Meir, R. 2005. Reinforcement learning with Gaussian processes. In *International Conference on Machine Learning*, pages 201–208. ACM.

Enright, P. J. and Conway, B. A. 1992. Discrete approximations to optimal trajectories using direct transcription and nonlinear programming. *Journal of Guidance, Control, and Dynamics*, 15(4):994–1002.

Ernst, D., Geurts, P., and Wehenkel, L. 2005. Tree-based batch mode reinforcement learning. *JMLR*, 6(Apr):503–556.

Model-free example

Even-Dar, E., Mannor, S., and Mansour, Y. 2002. PAC bounds for multi-armed bandit and Markov decision processes. In *International Conference on Computational Learning Theory*, pages 255–270. Springer.

Fernández Cara, E. and Zuazua Iriondo, E. 2003. Control theory: History, mathematical achievements and perspectives. *Boletín de la Sociedad Española de Matemática Aplicada, 26, 79-140.*

Ferson, S., Joslyn, C. A., Helton, J., Oberkampf, W., and Sentz, K. 2004. Summary from the epistemic uncertainty workshop: consensus amid diversity. *Reliability Engineering & System Safety*, 85(1):355–369.

Frigola, R. 2015. *Bayesian Time Series Learning with Gaussian Processes*. PhD thesis, University of Cambridge.

Gal, Y. 2016a. Homoscedastic and heteroscedastic models. *https://github.com/yaringal/ HeteroscedasticDropoutUncertainty*.

Gal, Y. 2016b. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge.

Gal, Y. and Ghahramani, Z. 2015. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *arXiv:1506.02142*.

Garcia, C. E., Prett, D. M., and Morari, M. 1989. Model predictive control: theory and practice – a survey. *Automatica*, 25(3):335–348.

Gelb, A. 1974. *Applied optimal estimation*. MIT press.

Ghavamzadeh, M., Mannor, S., Pineau, J., and Tamar, A. 2016. Bayesian reinforcement learning: A survey. *ArXiv e-prints*.

Gittins, J., Glazebrook, K., and Weber, R. 1989. *Multi-armed bandit allocation indices*. John Wiley & Sons.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. 2016. Continuous deep Q-learning with model-based acceleration. *arXiv preprint arXiv:1603.00748*.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. 2015. Learning continuous control policies by stochastic value gradients. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *NIPS*, pages 2944–2952. Curran Associates, Inc.

Model-free example

Hemakumara, P. and Sukkarieh, S. 2013. Learning UAV stability and control derivatives using Gaussian processes. *IEEE Transactions on Robotics*, 29(4):813–824.

GP dynamics modelling for a UAV.

Hespanha, J., Liberzon, D., and Morse, S. 2003. Overcoming the limitations of adaptive control by means of logic-based switching. *Systems & Control Letters*, 49(1):49–65. Adaptive Control.

Adaptive control's potential for instability.

Ioannou, P. and Sun, J. 2012. *Robust adaptive control*. Courier Corporation.

Isard, M. and Blake, A. 1998. Condensation—conditional density propagation for visual tracking. *IJCV*, 29(1):5–28.

Jaynes, E. 2003. *Probability theory: The logic of science*. Cambridge University Press.

Julier, S., Uhlmann, J., and Durrant-Whyte, H. 1995. A new approach for filtering nonlinear systems. In *American Control Conference*, volume 3, pages 1628–1632.

Jung, T. and Stone, P. 2010. Gaussian processes for sample efficient reinforcement learning with RMAX-like exploration. In *Machine Learning and Knowledge Discovery in Databases*, pages 601–616. Springer.

Kaelbling, L. P. 1993. *Learning in embedded systems*. The MIT Press.

Kaelbling, L. P., Littman, M., and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134.

Kalman, R. 1959. On the general theory of control systems. *IRE Transactions on Automatic Control*, 4(3):110–110.

Defined observability.

Kalman, R. 1960. A new approach to linear filtering and prediction problems. *Journal of Basic Eng.*, 82(1):35–45.

Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.

Kearns, M., Mansour, Y., and Ng, A. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208.

Sparse sampling. Grows a balanced lookahead tree, expanding a set number of actions then outcomes, to the horizon. Suitable for large MDPs, as computational complexity is independent of state space size.

Kearns, M. and Singh, S. 2002. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232.

Khalil, H. K. 1996. *Noninear Systems*. Prentice-Hall, New Jersey.

Kingma, D. and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Ko, J. and Fox, D. 2009. GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90.

Kocijan, J., Murray-Smith, R., Rasmussen, C., and Girard, A. 2004. Gaussian process model based predictive control. In *American Control Conference*, volume 3, pages 2214–2219.

Kolter, Z. and Ng, A. 2009. Near-Bayesian exploration in polynomial time. In *International Conference on Machine Learning*, pages 513–520. ACM.

Lagoudakis, M. and Parr, R. 2003. Least-squares policy iteration. *JMLR*, 4(Dec):1107–1149.

Model-free example

Lavalle, S. 1998. Rapidly-exploring random trees: A new tool for path planning. Technical report.

LaValle, S. 2006. *Planning algorithms*. Cambridge university press.

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

MacKay, D. 1992. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology.

Maybeck, P. 1982. *Stochastic models, estimation, and control*, volume 3. Academic press.

McAllister, R., Peynot, T., Fitch, R., and Sukkarieh, S. 2012. Motion planning and stochastic control with experimental validation on a planetary rover. In *IROS*, pages 4716–4723.

McFarlane, D. and Glover, K. 1992. A loop-shaping design procedure using h∞ synthesis. *Transactions on Automatic Control*, 37(6):759–769.

McHutchon, A. 2014. *Nonlinear modelling and control using Gaussian processes*. PhD thesis, University of Cambridge.

Andrew McHutchon's PhD thesis with the 'Direct method' of system identification given noisy observations.

McHutchon, A. and Rasmussen, C. 2011. Gaussian process training with input noise. In *NIPS*, pages 1341–1349.

Meuleau, N. and Bourgine, P. 1999. Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 35(2):117–154.

Global backups of exploration bonuses, for non-myopic BRL, tested on indirect RL algorithms (e.g. Gittins index local bonus IQL+, IE local bonus IEDP+).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mundhenk, M., Goldsmith, J., Lusena, C., and Allender, E. 1997. Encyclopaedia of complexity results for finite-horizon Markov decision process problems. *Center for Discrete Mathematics & Theoretical Computer Science*.

Murray-Smith, R. and Sbarbaro, D. 2002. Nonlinear adaptive control using nonparametric gaussian process prior models. *IFAC Proceedings Volumes*, 35(1):325–330.

Ng, A., Harada, D., and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

Nilsson, J. 1998. Real-time control systems with delays. *Department of Automatic Control, Lund Institute of Technology*, 1049.

LQG-optimal control with random time delays.

Nise, N. 2007. *Control Systems Engineering*. John Wiley & Sons.

A well-known introductory control textbook.

Ogata, K. and Yang, Y. 1970. *Modern control engineering*. Prentice-Hall Englewood Cliffs.

Another well-known introductory control textbook.

Osband, I., Blundell, C., Pritzel, A., and Roy, B. V. 2016. Deep exploration via bootstrapped DQN. *arXiv preprint arXiv:1602.04621*.

Pan, Y. and Theodorou, E. 2014. Probabilistic differential dynamic programming. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *NIPS*, pages 1907–1915. Curran Associates, Inc.

Pan, Y., Theodorou, E., and Kontitsis, M. 2015. Sample efficient path integral control under uncertainty. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *NIPS*, pages 2314–2322. Curran Associates, Inc.

Papadimitriou, C. and Tsitsiklis, J. 1987. The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450.

Pontryagin, L. S. 1987. *Mathematical theory of optimal processes*. CRC Press.

Poupart, P., Vlassis, N., Hoey, J., and Regan, K. 2006. An analytic solution to discrete Bayesian reinforcement learning. *ICML*, pages 697–704.

Powell, W. and Ryzhov, I. 2012. *Optimal learning*, volume 841. John Wiley & Sons.

Rasmussen, C. and Williams, C. 2006. *Gaussian processes for machine learning*. Citeseer.

The Gaussian process bible.

Ross, S., Chaib-draa, B., and Pineau, J. 2008. Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In *ICRA*, pages 2845–2851.

Recursive backups on belief states (belief over uncertain-MDP and uncertain-continuous-state), uniform sampling actions, random sampling observations, updates belief with a particle filter to depth d.

Ryzhov, I., Powell, W., and Frazier, P. 2012. The knowledge gradient algorithm for a general class of online learning problems. *Operations Research*, 60(1):180–195.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. 2016. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Slotine, J.-J. E., Li, W., et al. 1991. *Applied nonlinear control*, volume 199. prentice-Hall Englewood Cliffs, NJ.

Snelson, E. and Ghahramani, Z. 2006. Sparse Gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257.

Snoek, J., Larochelle, H., and Adams, R. 2012. Practical Bayesian optimization of machine learning algorithms. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *NIPS*, pages 2951–2959. Curran Associates, Inc.

Sondik, E. 1971. The optimal control of partially observable Markov processes. Technical report.

POMDP paper.

Sontag, E. 2013. *Mathematical control theory: deterministic finite dimensional systems*, volume 6. Springer.

A mathematically oriented control textbook, with some introduction to nonlinear control.

Stadie, B., Levine, S., and Abbeel, P. 2015. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

Stengel, R. F. 1986. *Stochastic optimal control*. John Wiley and Sons New York.

Strehl, A., Li, L., Wiewiora, E., Langford, J., and Littman, M. 2006. PAC model-free reinforcement learning. In *International Conference on Machine Learning*, pages 881–888. ACM.

Strens, M. 2000. A Bayesian framework for reinforcement learning. *Machine Learning*, pages 943–950.

Models transitions with Dirichlets, expected-rewards with Gaussians. Continually monitors belief, but occasionally samples a point-estimate MDP from it, solves it, and follows it greedily until next time to re-sample.

Sutton, R. and Barto, A. 1998. *Reinforcement learning: An introduction*. MIT press.

The reinforcement learning bible.

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. 1999. Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K., editors, *NIPS*, pages 1057–1063. MIT Press.

Szepesvári, C. 2010. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103.

Shorter, more recent and more mathmatically inclined introductory book for reinforcement learning.

Thompson, W. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

The original Thompson sampling paper.

Thrun, S. 1992. Efficient exploration in reinforcement learning. Technical report.

Titsias, M. 2009. Variational learning of inducing variables in sparse Gaussian processes. In *AISTATS*, volume 12, pages 567–574.

Variation free energy method for sparse Gaussian processes.

Umlauft, J. 2014. Using probabilistic models for non-linear system identification and control. Master's thesis, University of Cambridge.

Jonas Umlauft's masters thesis analyising stability of PILCO systems.

van den Berg, J., Patil, S., and Alterovitz, R. 2012. Motion planning under uncertainty using iterative local optimization in belief space. *International Journal of Robotics Research*, 31(11):1263–1278.

Vinogradska, J., Bischoff, B., Nguyen-Tuong, D., Schmidt, H., Romer, A., and Peters, J. 2016. Stability of controllers for Gaussian process forward models. In *ICML*, pages 545–554.

Wahlström, N., Schön, T., and Deisenroth, M. 2015. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*.

Walsh, T. J., Goschin, S., and Littman, M. L. 2010. Integrating sample-based planning and model-based reinforcement learning. In *AAAI*.

Wang, T., Lizotte, D., Bowling, M., and Schuurmans, D. 2005. Bayesian sparse sampling for on-line reward optimization. In *ICML*, pages 956–963. ACM.

Similar to Sparse Sampling, creates a lookahead tree, expanding C sampled outcomes per action, to depth d. Differences include 1) planning in belief space, 2) unbalanced tree by selecting worthy action-nodes to expand judged by Thompson sampling.

Wang, Y., Won, K. S., Hsu, D., and Lee, W. S. 2012. Monte Carlo Bayesian reinforcement learning. *arXiv preprint arXiv:1206.6449*.

BRL as a POMDP, the MDP belief b in the MDP is discretised by K samples.

Watkins, C. J. C. H. 1989. *Learning from delayed rewards*. PhD thesis, University of Cambridge England.

Webb, D. J., Crandall, K. L., and van den Berg, J. 2014. Online parameter estimation via real-time replanning of continuous Gaussian POMDPs. In *ICRA*, pages 5998–6005.

Wilcoxon, F., Katti, S., and Wilcox, R. A. 1970. Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test. *Selected tables in mathematical statistics*, 1:171–259.

Williams, R. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

REINFORCE algorithm.

Wittenmark, B. 1995. Adaptive dual control methods: An overview. In *IFAC Symposium on Adaptive Systems in Control and Signal Proc*, pages 67–72.

Xie, C., Patil, S., Moldovan, T., Levine, S., and Abbeel, P. 2015. Model-based reinforcement learning with parametrized physical models and optimism-driven exploration. *arXiv preprint arXiv:1509.06824*.