Technical Report UW-CSE-10-06-01

A Practical and Conceptual Framework for Learning in Control

Marc Peter Deisenroth[1,2] and Carl Edward Rasmussen[2,3]

[1]Department of Computer Science & Engineering, University of Washington, Seattle
[2]Department of Engineering, University of Cambridge, UK
[3]Max Planck Institute for Biological Cybernetics, Tübingen, Germany

June 1, 2010

Abstract

We propose a fully Bayesian approach for efficient reinforcement learning (RL) in Markov decision processes with continuous-valued state and action spaces when no expert knowledge is available. Our framework is based on well-established ideas from statistics and machine learning and learns fast since it carefully models, quantifies, and incorporates available knowledge when making decisions. The key ingredient of our framework is a probabilistic model, which is implemented using a Gaussian process (GP), a distribution over functions. In the context of dynamic systems, the GP models the transition function. By considering all plausible transition functions simultaneously, we reduce model bias, a problem that frequently occurs when deterministic models are used. Due to its generality and efficiency, our RL framework can be considered a conceptual and practical approach to learning models and controllers when expert knowledge is difficult to obtain or simply not available, which makes system identification hard.

Contents

Chapter 1

Introduction

Automatic control of dynamic systems has been a major discipline in engineering for decades. By using a controller, external signals can be applied to the system to modify the state of the system. The state fully describes the system at a particular point in time. Typically, the controller is designed by a skilled engineer, the *expert*, to drive the system in an optimal way. For the controller design, the expert identifies the dynamics of the system. Roughly speaking, the expert derives a mathematical formulation of the dynamics of the underlying system and identifies its parameters using data from the system. As an example let us consider the mass-spring mechanical system described in Figure 1.1. The mathematical formulation of the system's dynamics is given by Newton's law of motion. The parameters to be identified are for example friction coefficients, the mass of the block, or the spring constant.

One issue with the classical approach to automatic control is that it often relies on idealized assumptions and expert knowledge to derive the mathematical formulation for each system. The expert provides an intricate understanding of the properties of system's dynamics and the control task. Depending on the system, expert knowledge might not be available or expensive to obtain. When neither valid idealized assumptions[1] about a dynamic system can be made due to too many hidden parameters nor sufficient expert knowledge is available, (computational) *learning* techniques can be valuable in automatic control. Therefore, learning algorithms got used more often in automatic control during the last decades. In particular, in the context of system identification, learning has been employed to reduce the dependency on idealized assumptions. A learning algorithm can be considered a method that automatically extracts structure from data. The extracted information can be used for predictions and for decision making by speculating about the long-term consequences of particular actions. In a human/animal learning context the data used for learning are often referred to as *experience*.

Computational approaches for artificial learning from experience are studied in neuroscience, reinforcement learning (RL), approximate dynamic programming, and adaptive control, for instance. Although these fields have been studied for decades, the rate at which artificial systems learn still lags behind biological learners with respect to the amount of experience required to learn a task if no expert knowledge is available. Experience can be gathered by direct interaction with the environment. Interaction, however,

[1] Often, the parameters of a dynamic system are assumed to follow Newton's laws of motion exactly, which we call "idealized" assumptions.

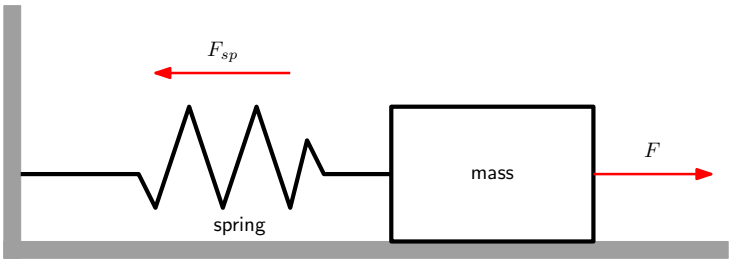


Figure 1.1: Simplified mass-spring system described for example in the book by Khalil (2002). The mass is subjected to an external force F. The restoring force of the spring is denoted by F_{sp}.

can be time consuming or, in a robotic system, accelerate attrition. Hence, a central issue in RL is to speed up artificial learning algorithms by making them more efficient in terms of required interactions with the system.

There are broadly two ways to increase the (interaction) efficiency of reinforcement learning. One approach is to exploit expert knowledge to constrain the task in various ways and to simplify learning. This approach is highly problem dependent since it relies on an intricate understanding of the characteristics of the task and the solution. Expert knowledge can be difficult to obtain, expensive, or is simply not available. A second approach to make reinforcement learning more efficient is to extract more useful information from available experience. This approach does not rely on expert knowledge, but requires to model available data carefully. In a practical application, one would typically combine these two approaches. In this report, however, we are solely concerned with the second approach:

How can we learn as fast as possible given only a very general prior understanding of a task?

Thus, we do not look for an engineering solution to a particular problem. Instead, we elicit a general and principled framework for efficient learning in the context of control applications.

Our approach mimics two fundamental properties of human experience-based learning. The first important characteristic of humans is that we can *generalize* our experience to unknown situations. Second, humans explicitly model and incorporate *uncertainty* into our decisions as shown by Körding and Wolpert (2004a) and Körding and Wolpert (2006).

Unlike for discrete domains, where Poupart et al. (2006) and Poupart and Vlassis (2008) propose a general, theoretical framework for Bayesian RL, generalization and incorporation of uncertainty into the decision-making process are not coherently combined in continuous RL, although heuristics exist (Abbeel et al., 2006). In the context of motor control, generalization typically requires a model or a simulator, that is, an internal representation of the (system) dynamics. In the case of only few interactions with the system, we face the problem of dealing with fairly limited experience to solve a task. We use this limited experience to learn a model of the underlying dynamics. We explicitly require a *probabilistic* model to represent and to quantify uncertainty for a coherent generalization of available experience to unknown situations. Moreover, we incorporate the model uncertainty into the decision-making process.

We present a general and fully Bayesian framework for efficient RL. Our approach does not rely on expert knowledge and can naturally be applied to episodic tasks with continuous-valued states and actions. Although only discussed in the context of mechanical control problems, the approach is more widely applicable, including biological process control.

The structure is as follows. In Chapter 2, we introduce Gaussian process regression with a focus on predicting with GPs. Chapter 3 details the proposed learning framework. A discussion about the properties of our approach is given in Chapter 4.

Chapter 2

Regression with Gaussian Processes

Regression is the problem of estimating a function h given a set of input vectors $\mathbf{x}_i \in \mathbb{R}^D$ and observations $y_i = h(\mathbf{x}_i) + \varepsilon_i \in \mathbb{R}$ of the corresponding function values, where ε_i is a noise term. Regression problems frequently arise in the context of reinforcement learning, control theory, and control applications. For example, the transitions in a dynamic system are typically described by a stochastic or deterministic function h. Due to the presence of noise, the quantity of interest, that is, the estimate of the function h, is uncertain. The Bayesian framework allows us to express this uncertainty in terms of probability distributions requiring the concept of distributions over functions—a Gaussian process (GP) provides such a distribution.

In a classical control context, we typically define the transition function to be estimated by means of a finite number of parameters $\boldsymbol{\phi}$, which are often motivated by Newton's laws of motion. These parameters can be masses and friction coefficients, for instance. In this context, regression aims to find a parameter set $\boldsymbol{\phi}^*$ such that $h(\boldsymbol{\phi}^*, \mathbf{x}_i)$ fit the observations y_i, $i = 1, \ldots, n$, best. Within the Bayesian framework, a probability distribution over the parameters $\boldsymbol{\phi}$ expresses our uncertainty and beliefs about the function h.

Often we are interested in making predictions using the model for the function h. To make predictions at an arbitrary input $\mathbf{x}_*$, we take the uncertainty about the parameters into account by averaging over them with respect to their probability distribution. We then obtain a predictive distribution $p(y_*|\mathbf{x}_*)$, which sheds light not only on the expected value of y_*, but also on the uncertainty of this estimated value.

In these so called *parametric models*, the parameter set $\boldsymbol{\phi}$ imposes a particular structure upon the function h. The number of parameters is fixed in advance and independent of the number of observations, the sample size. If the parametric model is too restrictive, we might think that the current set of parameters motivated by physical laws is not the full set of parameters acting on the dynamic system: Often one assumes idealized circumstances, such as massless sticks and frictionless systems. One option to make the model more flexible is to add parameters to $\boldsymbol{\phi}$, which we think they may be of importance. However, this approach quickly gets complicated, and some effects such as slack can be difficult to describe or to identify. At this point, we can go one step back, dispense with the physical interpretability of the system parameters, and employ so called *non-parametric models*.

The basic idea of non-parametric regression is to determine the shape of the underlying function h from the data and higher-level assumptions, such as the smoothness of h. The term "non-parametric" does not imply that the model has no parameters, but that the number of the parameters is flexible and grows with the sample size. Usually, this means using statistical models that are infinite-dimensional (Wasserman, 2006). In the context of non-parametric regression, the "parameters" of interest are the values of the underlying function h itself.

In this report, we will focus on *Gaussian process (GP) regression*, also known as *kriging*. GPs are used for state-of-the-art regression and combine the flexibility of non-parametric modeling with tractable Bayesian inference: Instead of inferring a single function (a point estimate) from data, GPs infer a distribution over functions.

2.1 Definition and Model

A *stochastic process* is a function b of two arguments $\{b(t,\omega) : \ t \in T, \ \omega \in \Omega\}$, where T is an index set, time for example, and Ω is a sample space. For fixed $t \in T$, $\{b(t,\cdot)\}$ is thus a collection of random variables.

A *Gaussian process* is a distribution over functions and a generalization of the Gaussian distribution to an infinite-dimensional function space: Let $h_1, \ldots, h_{|T|}$ be a set of random variables, where T is an index set. For $|T| < \infty$, we can collect these random variables in a random vector $\mathbf{h} = [h_1, \ldots, h_{|T|}]^\top$. Generally, a vector can be regarded as a function $h : i \mapsto h(i) = h_i$ with finite domain, $i = 1, \ldots, |T|$, which maps indices to vector entries. For $|T| = \infty$ the domain of the function is infinite and the mapping is given by $h : x \mapsto h(x)$. Roughly speaking, the image of the function is an infinitely long vector. Let us now consider the case $(x_t)_{t\in T}$ and $h : x_t \mapsto h(x_t)$, where $h(x_1), \ldots, h(x_{|T|})$ have a joint (Gaussian) distribution. For $|T| < \infty$ the values $h(x_1), \ldots, h(x_{|T|})$ are distributed according to a multivariate Gaussian. For $|T| = \infty$, the corresponding infinite-dimensional distribution of the random variables $h(x_t)$, $t = 1, \ldots, \infty$ is a stochastic process, more precisely, a Gaussian process. Therefore, a Gaussian distribution and a Gaussian process are not the same. A GP is a collection of random variables, any finite number of which have a consistent joint Gaussian distributions (Åström, 2006; Rasmussen and Williams, 2006). However, all computations required for regression and inference with GPs can be broken down to manipulating multivariate Gaussian distribution as we see in the following.

In the GP regression model, we assume that the data $\mathcal{D} \coloneqq \{\mathbf{X} \coloneqq [\mathbf{x}_1, \ldots, \mathbf{x}_n], \mathbf{y} \coloneqq [y_1, \ldots, y_n]^\top\}$ have been generated according to $y_i = h(\mathbf{x}_i) + \varepsilon_i$, where $h : \mathbb{R}^D \to \mathbb{R}$ and $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is independent (measurement) noise. GPs consider h a random function and infer a posterior distribution over h from data. The posterior is used to make predictions about function values $h(\mathbf{x}_*)$ for arbitrary inputs $\mathbf{x}_* \in \mathbb{R}^D$.

Similar to a Gaussian distribution, which is fully specified by a mean vector and a covariance matrix, a GP is fully specified by a mean *function* $m_h(\,\cdot\,)$ and a covariance *function*

$$k_h(\mathbf{x}, \mathbf{x}') \coloneqq \mathbb{E}_h[(h(\mathbf{x}) - m_h(\mathbf{x}))(h(\mathbf{x}') - m_h(\mathbf{x}'))] = \operatorname{cov}_h[h(\mathbf{x}), h(\mathbf{x}')], \quad \mathbf{x}, \ \mathbf{x}' \in \mathbb{R}^D,$$

which specifies the covariance between any two function values. Here, $\mathbb{E}_h$ denotes the expectation with respect to the function h. The covariance function $k_h(\,\cdot\,,\,\cdot\,)$ is also called a *kernel*. Similar to the mean value of a Gaussian distribution, the mean function m_h describes how the "average" function is expected to look.

The GP definition yields that any finite set of function values $h(\mathbf{X}) \coloneqq [h(\mathbf{x}_1), \ldots, h(\mathbf{x}_n)]$ is jointly Gaussian distributed. Using the notion of the mean function and the covariance function, the Gaussian distribution of any finite set of function values $h(\mathbf{X})$ can be explicitly written down as

$$p(h(\mathbf{X})) = \mathcal{N}(m_h(\mathbf{X}), k_h(\mathbf{X}, \mathbf{X})), \tag{2.1}$$

where $k_h(\mathbf{X}, \mathbf{X})$ is the full covariance matrix of all function values $h(\mathbf{X})$ under consideration. The graphical model of a GP is given in Figure 2.1. We denote a function that is GP distributed by $h \sim \mathcal{GP}$ or $h \sim \mathcal{GP}(m_h, k_h)$.

2.2 Bayesian Inference

To find a posterior distribution of the (random) function h, we employ Bayesian inference techniques within the GP framework. Gelman et al. (2004) considers Bayesian inference as the process of fitting a probability model to a set of data and summarizing the result by a probability distribution on the unknown quantity. Bayesian inference can be considered a three-step procedure: First, a prior on the unknown quantity has to be specified. In our case, the unknown quantity is the function h itself. Second, data are observed. Third, a posterior distribution over h is computed that refines the prior by incorporating evidence from the observations. Let us briefly go through these steps in the context of GPs.

2.2.1 Prior

When modeling with Gaussian processes, we place a Gaussian process prior $p(h)$ directly on the space of functions. In the GP model, we have to specify the prior mean function and the prior covariance function.

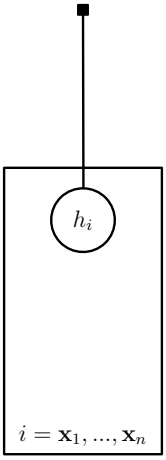


Figure 2.1: Factor graph of a GP model. The node h_i is a short-hand notation for $h(\mathbf{x}_i)$. The plate notation is a compact representation of a n-fold copy of the node h_i, $i = \mathbf{x}_1, \ldots, \mathbf{x}_n$. The black square is a factor. In the GP model any finite collection of function values $h(\mathbf{x}_1), \ldots, h(\mathbf{x}_n)$ has a joint Gaussian distribution.

Unless stated otherwise, we consider a prior mean function $m_h \equiv 0$ and use the squared exponential (SE) covariance function

$$k_h(\mathbf{x}_p, \mathbf{x}_q) = \alpha^2 \exp\left(-\tfrac{1}{2}(\mathbf{x}_p - \mathbf{x}_q)^\top \mathbf{\Lambda}^{-1}(\mathbf{x}_p - \mathbf{x}_q)\right), \quad \mathbf{x}_p, \mathbf{x}_q \in \mathbb{R}^D, \tag{2.2}$$

plus a noise covariance function $\delta_{pq}\sigma_\varepsilon^2$, where $\mathbf{\Lambda} = \operatorname{diag}([\ell_1^2, \ldots, \ell_D^2])$ is a diagonal matrix of squared characteristic length-scales ℓ_i, $i = 1, \ldots, D$, and α^2 is the signal variance of the latent function h. The δ denotes the Kronecker symbol that is unity when $p = q$ and zero otherwise, which essentially encodes that the measurement noise is independent.[1] With the SE covariance function in equation (2.2) we assume that the latent function h is smooth and stationary. Examples of covariance functions that encode other model assumptions are given by Rasmussen and Williams (2006, Chapter 4).

2.2.2 Posterior

After having observed function values $\mathbf{y}$ for a set of input vectors $\mathbf{X}$, we compute the posterior distribution over h according to

$$p(h|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}|h, \mathbf{X})p(h)}{p(\mathbf{y}|\mathbf{X})}. \tag{2.3}$$

We assume that the observations y_i are conditionally independent given $\mathbf{X}$. Therefore, the likelihood of h factors into

$$p(\mathbf{y}|h, \mathbf{X}) = \prod_{i=1}^{n} p(y_i|h(\mathbf{x}_i)) = \prod_{i=1}^{n} \mathcal{N}(y_i \mid h(\mathbf{x}_i), \sigma_\varepsilon^2) = \mathcal{N}(\mathbf{y} \mid h(\mathbf{X}), \sigma_\varepsilon^2 \mathbf{I}). \tag{2.4}$$

2.2.3 Hierarchical Inference

Thus far, it is not quite clear how to interpret the GP prior $p(h)$ in equation (2.3) since the covariance function and possibly the mean function depend on a set of *hyper-parameters*. In our case, only the covariance function contains hyper-parameters, namely the variance σ_ε^2 of the measurement noise, the variance α^2 of the latent function, and the length-scales of the SE covariance function given in equation (2.2). To take them into account, we require a hierarchical inference scheme.

Figure 2.2 is a graphical model that describes the hierarchical inference structure we consider: At the bottom is an observed level given by the data $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$. Above the data is the function h, the random

[1] I thank Ed Snelson for pointing out that the Kronecker symbol is defined on the indices of the samples and not on input locations. Therefore, $\mathbf{x}_p$ and $\mathbf{x}_q$ are uncorrelated according to the noise covariance function if $\mathbf{x}_p = \mathbf{x}_q$, but $p \neq q$.

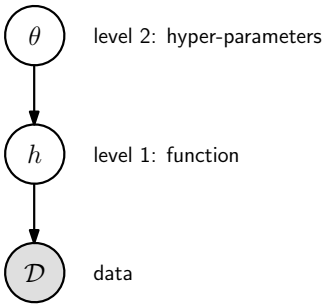


Figure 2.2: Hierarchical model for Bayesian inference with GPs.

"variable" we are primarily interested in. The top level is defined by the hyper-parameters $\boldsymbol{\theta}$ specifying the (prior) distribution of the function values $h(\mathbf{x})$. A third level of models $\mathcal{M}_i$, for example different covariance functions, could be added on top. This case is not discussed in this report since we always choose a single covariance function. Rasmussen and Williams (2006) provide the details on a three-level inference in the context of model selection.

Let us have a closer look at the two-level inference. In a fully Bayesian setup, one places a hyper-prior on the hyper-parameters and integrates them out, such that

$$p(h) = \int p(h|\boldsymbol{\theta})p(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}\,,$$
$$p(\mathbf{y}|\mathbf{X}) = \iint p(\mathbf{y}|\mathbf{X},h,\boldsymbol{\theta})p(h|\boldsymbol{\theta})p(\boldsymbol{\theta})\,\mathrm{d}h\,\mathrm{d}\boldsymbol{\theta}\,.$$

Often, the integration required for $p(\mathbf{y}|\mathbf{X})$ is analytically intractable since $p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})$ is a nasty function of $\boldsymbol{\theta}$. Approximate averaging over the hyper-parameters can be done using computationally demanding Monte Carlo method. In the following, we do not follow the fully Bayesian path but instead find a good point estimate $\hat{\boldsymbol{\theta}}$ of hyper-parameters on which we condition our inference.

Level-1 Inference

When we condition on the hyper-parameters, the posterior on the function is

$$p(h|\mathbf{X},\mathbf{y},\boldsymbol{\theta}) = \frac{p(\mathbf{y}|\mathbf{X},h)\,p(h|\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})}\,, \tag{2.5}$$

where $p(\mathbf{y}|\mathbf{X},h)$ is the likelihood of the function h, see equation (2.4), and $p(h|\boldsymbol{\theta})$ is the GP prior on h. The normalizing constant

$$p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{X},h)\,p(h|\boldsymbol{\theta})\,\mathrm{d}h \tag{2.6}$$

in equation (2.5) is the *marginal likelihood* also called the *evidence*. The marginal likelihood can be considered the likelihood of the hyper-parameters given the data after having marginalized out the function h.

Level-2 Inference

The posterior on the hyper-parameters is

$$p(\boldsymbol{\theta}|\mathbf{X},\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})\,p(\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X})}\,, \tag{2.7}$$

where $p(\boldsymbol{\theta})$ is the hyper-prior. The marginal likelihood at the second level is

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})\,p(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}\,, \tag{2.8}$$

where we average over the hyper-parameters. This integral is analytically intractable in most interesting cases. However, we can find a point estimate $\hat{\boldsymbol{\theta}}$ of the hyper-parameters. In the following, we discuss how to find this point estimate.

2.2.4 Estimating the Hyper-Parameters via Marginal-Likelihood Maximization

When we choose the hyper-prior $p(\boldsymbol{\theta})$, a priori we must not exclude any possible settings of the hyper-parameters. By choosing a "flat" prior, we assume that any values for the hyper-parameters are a priori possible. The flat prior on the hyper-parameters comes along with computational advantages since it makes the posterior distribution over $\boldsymbol{\theta}$ (see equation (2.7)) proportional to the marginal likelihood in equation (2.6). To find a vector of "good" hyper-parameters (level-2 inference), we therefore maximize the marginal likelihood in equation (2.6) with respect to the hyper-parameters, which is recommended by MacKay (1999). In particular, the log-marginal likelihood (log-evidence) is

$$\begin{aligned}\log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) &= \log \int p(\mathbf{y}|h,\mathbf{X},\boldsymbol{\theta})\, p(h|\boldsymbol{\theta})\, \mathrm{d}h \\ &= \underbrace{-\tfrac{1}{2}\mathbf{y}^\top (\mathbf{K}_\theta + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{y}}_{\text{data-fit term}} \underbrace{-\tfrac{1}{2}\log|(\mathbf{K}_\theta + \sigma_\varepsilon^2\mathbf{I})|}_{\text{complexity term}} - \tfrac{D}{2}\log(2\pi)\,,\end{aligned} \tag{2.9}$$

where D is the dimension of the input space and $\mathbf{K}$ is the kernel matrix with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. We made the dependency of $\mathbf{K}$ on the hyper-parameters $\boldsymbol{\theta}$ explicit by writing $\mathbf{K}_\theta$. Equation (2.9) now also reveals why the marginal likelihood is a nasty function of the hyper-parameters $\boldsymbol{\theta}$. The hyper-parameter vector

$$\hat{\boldsymbol{\theta}} \in \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})$$

is called a *type II maximum likelihood estimate* (ML-II) of the hyper-parameters, which can be used at the bottom level of the hierarchical inference scheme to determine the posterior distribution over h (Rasmussen and Williams, 2006).[2]

Evidence maximization yields a posterior GP model that trades off data-fit and model complexity. Hence, it implements Occam's razor, which tells us to use the simplest model that explains the data. MacKay (2003) and Rasmussen and Ghahramani (2001) show that coherent Bayesian inference automatically embodies Occam's razor.

Maximizing the evidence using equation (2.9) is a nonlinear, non-convex optimization problem. This can be hard depending on the data set. However, after optimizing the hyper-parameters, the GP model can always explain the data although a global optimum has not necessarily been found. Alternatives to ML-II, such as cross validation or hyper-parameter marginalization, can be employed. Cross validation is computationally expensive, and marginalizing out the hyper-parameters is typically analytically intractable and requires Monte Carlo methods.

Given the estimate $\hat{\boldsymbol{\theta}}$, the Gaussian likelihood $p(\mathbf{y}|\mathbf{X}, h, \hat{\boldsymbol{\theta}})$ and the GP prior $p(h|\hat{\boldsymbol{\theta}})$ in equation (2.5) lead to the GP posterior in equation (2.5) with a mean function and a covariance function

$$\begin{aligned}\mathbb{E}_h[h(\tilde{\mathbf{x}})] &= m_h^{\text{post}}(\tilde{\mathbf{x}}) = k_h(\tilde{\mathbf{x}}, \mathbf{X})(k_h(\mathbf{X},\mathbf{X}) + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{y}\,, \\ \operatorname{cov}_h[h(\tilde{\mathbf{x}}), h(\mathbf{x}')] &= k_h^{\text{post}}(\tilde{\mathbf{x}}, \mathbf{x}') = k_h(\tilde{\mathbf{x}}, \mathbf{x}') - k_h(\tilde{\mathbf{x}}, \mathbf{X})(k_h(\mathbf{X},\mathbf{X}) + \sigma_\varepsilon^2\mathbf{I})^{-1}k_h(\mathbf{X}, \mathbf{x}')\,,\end{aligned}$$

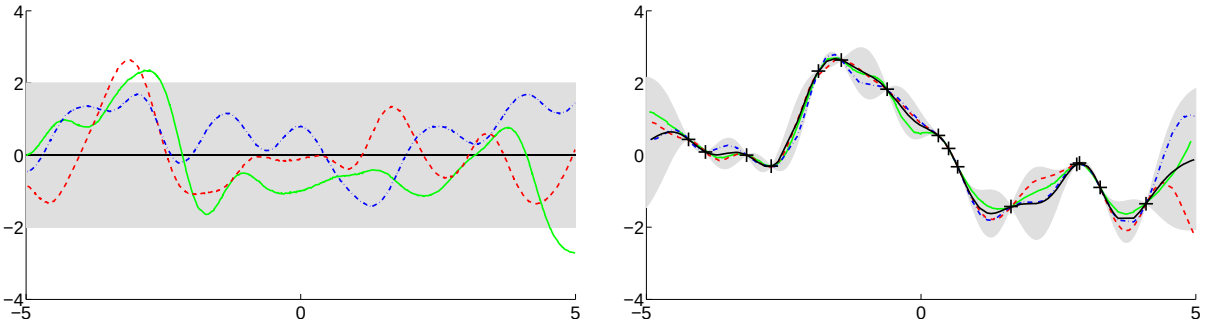
respectively, where $\tilde{\mathbf{x}}, \mathbf{x}' \in \mathbb{R}^D$ are arbitrary vectors, which we call *test inputs*. For notational convenience, we write $k_h(\mathbf{X}, \mathbf{x}')$ for $[k_h(\mathbf{x}_1, \mathbf{x}'), \ldots, k_h(\mathbf{x}_n, \mathbf{x}')] \in \mathbb{R}^{n\times 1}$. Note that $k_h(\mathbf{x}', \mathbf{X}) = k_h(\mathbf{X}, \mathbf{x}')^\top$. Figure 2.3 pictorially summarizes the Bayesian inference for GP regression and shows samples from the GP prior and the GP posterior.

A graphical model for a full GP is given in Figure 2.4. We distinguish between three sets of input vectors: training, testing, and "other". Training inputs are the vectors based on which the hyper-parameters have been optimized, test inputs are query points for predictions. All "other" variables are marginalized out during training and testing and added to the figure for completeness.

2.3 Predictions

The main focus of this report lies on how we can use of GP models for reinforcement learning and smoothing. Both tasks require iterative predictions with GPs when the input is given by a probability

[2] We computed the ML-II estimate $\hat{\boldsymbol{\theta}}$ using the `gpml`-software, which is publicly available at `http://www.gaussianprocess.org`.



(a) Samples from the GP prior. Without any observations, the prior uncertainty about the underlying function is constant everywhere.

(b) Samples from the GP posterior after having observed 15 function values (black crosses). The posterior uncertainty varies and depends on the location of the data.

Figure 2.3: Samples from the GP prior and the GP posterior for fixed hyper-parameters. The solid black lines represent the mean functions, the shaded areas show twice the standard deviation. The colored lines represent three sample functions from the prior GP and the posterior GP, panel (a) and panel (b), respectively.

distribution. In the following, we provide the provide central theoretical foundations of this report by discussing predictions with GPs in detail. We cover both predictions with deterministic and random inputs for univariate and multivariate targets.

In the following, we always assume a GP posterior, that is, we gathered training data and learned the hyper-parameters using marginal-likelihood maximization. The posterior GP can be used to compute the posterior predictive distribution of $h(\mathbf{x}_*)$ for any test input $\mathbf{x}_*$. From now on, we call the "posterior predictive distribution" simply a "predictive distribution" and omit the explicit dependence on the ML-II estimate $\hat{\boldsymbol{\theta}}$ of the hyper-parameters.

2.3.1 Predictions with Deterministic Inputs

From the definition of the GP, the function values for test inputs and training inputs are jointly Gaussian, that is,

$$p(\mathbf{h}, \mathbf{h}_*) = \mathcal{N}\left(\begin{bmatrix} m_h(\mathbf{X}) \\ m_h(\mathbf{X}_*) \end{bmatrix}, \begin{bmatrix} \mathbf{K} & k_h(\mathbf{X}, \mathbf{X}_*) \\ k_h(\mathbf{X}_*, \mathbf{X}) & \mathbf{K}_* \end{bmatrix}\right), \tag{2.10}$$

where we define $\mathbf{h} \coloneqq [h(\mathbf{x}_1), \ldots, h(\mathbf{x}_n)]^\top$ and $\mathbf{h}_* \coloneqq [h(\mathbf{x}_{*1}), \ldots, h(\mathbf{x}_{*m})]^\top$. All "other" function values have been integrated out.

Univariate Predictions

Let us start with the case that the observations y are scalar and the test input $\mathbf{x}_*$ is deterministic. From equation (2.10), it follows that the predictive marginal distribution $p(h(\mathbf{x}_*)|\mathcal{D}, \mathbf{x}_*)$ of the function value $h(\mathbf{x}_*)$ is Gaussian with mean and variance

$$\mu_* \coloneqq m_h(\mathbf{x}_*) \coloneqq \mathbb{E}_h[h(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}] = k_h(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{y} = k_h(\mathbf{x}_*, \mathbf{X})\boldsymbol{\beta}\,, \tag{2.11}$$

$$\sigma_*^2 \coloneqq \sigma_h^2(\mathbf{x}_*) \coloneqq \mathrm{var}_h[h(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}] = k_h(\mathbf{x}_*, \mathbf{x}_*) - k_h(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1} k_h(\mathbf{X}, \mathbf{x}_*)\,, \tag{2.12}$$

respectively, where $\boldsymbol{\beta} \coloneqq (\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{y}$. Note that $k_h(\mathbf{x}_*, \mathbf{x}_*)$ in equation (2.12) is essentially the prior model uncertainty plus measurement noise. From this prior uncertainty, we subtract an expression that encodes how much information we can transfer from the training set $\mathbf{X}$ to the test input $\mathbf{x}_*$. Since $k_h(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1} k_h(\mathbf{X}, \mathbf{x}_*)$ is positive definite, the posterior variance $\sigma_h^2(\mathbf{x}_*)$ cannot be larger than the prior uncertainty, which makes intuitive sense.

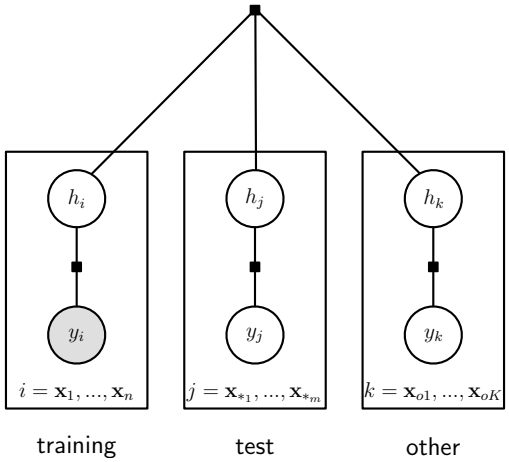


Figure 2.4: Factor graph for GP regression. The shaded nodes are observed variables. The factor inside the plates correspond to the likelihood. In the left plate, the variables $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, denote the training set. The variables $\mathbf{x}_{*_j}$ in the center plate are a finite number of test inputs. The corresponding test targets are unobserved. The right-hand plate subsumes any "other" finite set of inputs, function values, and observations. In GP regression, the "other" nodes are integrated out.

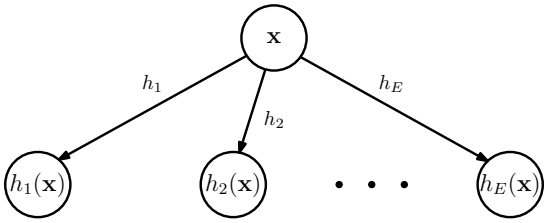


Figure 2.5: Graphical model if the latent function h maps into $\mathbb{R}^E$. The function values across dimensions are conditionally independent given the input.

Multivariate Predictions

If $\mathbf{y} \in \mathbb{R}^E$ is a multivariate target, we train E independent GP models using the same training inputs $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$, $\mathbf{x}_i \in \mathbb{R}^D$, but different training targets $\mathbf{y}_a = [y_1^a, \ldots, y_n^a]^\top$, $a = 1, \ldots, E$. Under this model, we assume that the function values $h_1(\mathbf{x}), \ldots, h_E(\mathbf{x})$ are conditionally independent given an input $\mathbf{x}$. Within the same dimension, however, the function values are still fully jointly Gaussian. The graphical model in Figure 2.5 shows the independence structure in the model across dimensions. Intuitively, the target values of different dimensions can only "communicate" via $\mathbf{x}$. Therefore, the target values covary only if $\mathbf{x}$ is uncertain.

For a known $\mathbf{x}_*$, the distribution of a predicted function value for a single target dimension is given by the equations (2.11) and (2.12), respectively. Under the model described by Figure 2.5, the predictive distribution of $h(\mathbf{x}_*)$ is Gaussian with mean and covariance

$$\boldsymbol{\mu}_* = \begin{bmatrix} m_{h_1}(\mathbf{x}_*) & \ldots & m_{h_E}(\mathbf{x}_*) \end{bmatrix}^\top , \tag{2.13}$$

$$\boldsymbol{\Sigma}_* = \operatorname{diag}\left(\begin{bmatrix} \sigma_{h_1}^2 & \ldots & \sigma_{h_E}^2 \end{bmatrix} \right) , \tag{2.14}$$

respectively.

2.3.2 Predictions with Uncertain Inputs

In the following, we discuss how to predict with Gaussian processes when the test input $\mathbf{x}_*$ has a probability distribution. Many derivations in the following are based on the thesis by Kuss (2006) and the work by Quiñonero-Candela et al. (2003a,b).

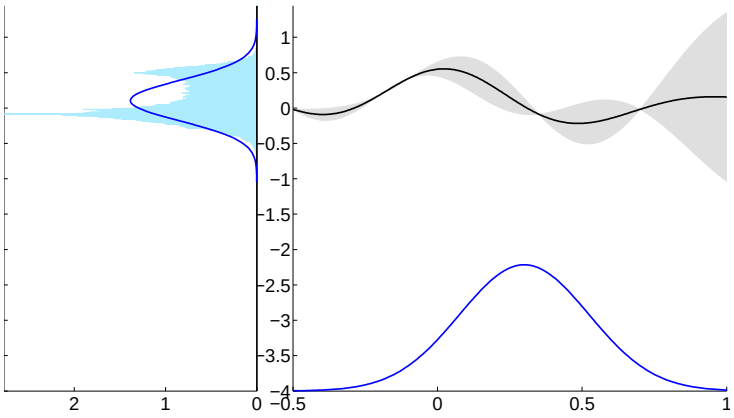


Figure 2.6: GP prediction with an uncertain test input. To determine the expected function value, we average over both the input distribution (blue, sitting on the bottom of the right panel) and the function distribution (GP model). The exact predictive distribution (histogram in the left panel) is approximated by a Gaussian (blue) that possesses the mean and the covariance of the exact predictive distribution (moment matching). Therefore, the blue Gaussian distribution in the left panel minimizes the Kullback-Leibler $\mathrm{KL}(p||q)$ divergence between the histogram representing the true distribution p and the approximate distribution q.

Consider the problem of predicting a function value $h(\mathbf{x}_*)$, $h : \mathbb{R}^D \to \mathbb{R}$, for an *uncertain* test input $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $h \sim \mathcal{GP}$ with an SE kernel k_h. This situation is illustrated in Figure 2.6. The input distribution $p(\mathbf{x}_*)$ is the blue Gaussian sitting on the bottom of the right panel. This panel also shows the mean function (black) and twice the standard deviation (shaded). Generally, if a Gaussian input $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is mapped through a nonlinear function, the exact predictive distribution

$$p(h(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \int p(h(\mathbf{x}_*)|\mathbf{x}_*) p(\mathbf{x}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \, \mathrm{d}\mathbf{x}_* \tag{2.15}$$

is not Gaussian and unimodal as shown in the left panel of Figure 2.6, where the histogram represents the exact distribution over function values. By explicitly conditioning on $\mathbf{x}_*$ in $p(h(\mathbf{x}_*)|\mathbf{x}_*)$, we emphasize that $\mathbf{x}_*$ is a deterministic argument of h in this conditional distribution. We approximate the exact predictive distribution $p(h(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ by a Gaussian (blue in left panel of Figure 2.6) that possesses the same mean and variance (moment matching). To determine the moments of the predictive function value, we average over both the input distribution and the distribution of the function given by the GP.

Univariate Predictions

Suppose $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a Gaussian distributed test point. The mean and the variance of the GP predictive distribution for $p(h(\mathbf{x}_*)|\mathbf{x}_*)$ in the integrand in equation (2.15) are given in equation (2.11) and equation (2.12), respectively. For the SE kernel, we can compute the mean μ_* and the variance σ_*^2 of the predictive distribution in equation (2.15) in closed form.[3] The mean μ_* can be computed using the law of iterated expectations (Fubini's theorem) and is given by

$$\begin{aligned} \mu_* &= \iint h(\mathbf{x}_*) p(h, \mathbf{x}_*) \, \mathrm{d}(h, \mathbf{x}_*) = \mathbb{E}_{\mathbf{x}_*, h}[h(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{\mathbf{x}_*}[\mathbb{E}_h[h(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \\ &\overset{(2.11)}{=} \mathbb{E}_{\mathbf{x}_*}[m_h(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \int m_h(\mathbf{x}_*) \mathcal{N}(\mathbf{x}_* \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \, \mathrm{d}\mathbf{x}_* \overset{(2.11)}{=} \boldsymbol{\beta}^\top \mathbf{q}, \end{aligned} \tag{2.16}$$

where $\mathbf{q} = [q_1, \ldots, q_n]^\top \in \mathbb{R}^n$ with

$$\begin{aligned} q_i &\coloneqq \int k_h(\mathbf{x}_i, \mathbf{x}_*) \mathcal{N}(\mathbf{x}_* \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \, \mathrm{d}\mathbf{x}_* \\ &= \alpha^2 |\boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|^{-\frac{1}{2}} \exp\big(-\tfrac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu})^\top (\boldsymbol{\Sigma} + \boldsymbol{\Lambda})^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) \big) . \end{aligned} \tag{2.17}$$

[3] This statement holds for all kernels, for which the integral of the kernel times a Gaussian can be computed analytically. In particular, this is true for kernels that involve polynomials, squared exponentials, and trigonometric functions.

Each q_i is an expectation of $k_h(\mathbf{x}_i, \mathbf{x}_*)$ with respect to the probability distribution of $\mathbf{x}_*$. This means, q_i is the expected covariance between the function values $h(\mathbf{x}_i)$ and $h(\mathbf{x}_*)$. The values q_i correspond to the standard SE kernel $k_h(\mathbf{x}_i, \boldsymbol{\mu})$, which has been "inflated" by $\boldsymbol{\Sigma}$. For a deterministic input $\mathbf{x}_*$ with $\boldsymbol{\Sigma} \equiv \mathbf{0}$, we obtain $\boldsymbol{\mu} = \mathbf{x}_*$ and recover $q_i = k_h(\mathbf{x}_i, \mathbf{x}_*)$. Then, the predictive mean (2.16) equals the predictive mean for certain inputs given in equation (2.11). Note that the predictive mean in equation (2.16) depends explicitly on the mean and covariance of the distribution of the input $\mathbf{x}_*$. Using Fubini's theorem, the variance σ_*^2 of the predictive distribution $p(h(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is

$$
\begin{aligned}
\sigma_*^2 &= \operatorname{var}_{\mathbf{x}_*,h}[h(\mathbf{x}_*)] = \mathbb{E}_{\mathbf{x}_*}[\operatorname{var}_h[h(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] + \operatorname{var}_{\mathbf{x}_*}[\mathbb{E}_h[h(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \\
&= \mathbb{E}_{\mathbf{x}_*}[\sigma_h^2(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] + \left(\mathbb{E}_{\mathbf{x}_*}[m_h(\mathbf{x}_*)^2|\boldsymbol{\mu}, \boldsymbol{\Sigma}] - \mathbb{E}_{\mathbf{x}_*}[m_h(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}]^2\right) \\
&= \int k_h(\mathbf{x}_*, \mathbf{x}_*) - k_h(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1} k_h(\mathbf{X}, \mathbf{x}_*) p(\mathbf{x}_*) \, \mathrm{d}\mathbf{x}_* \\
&\quad + \int \underbrace{k_h(\mathbf{x}_*, \mathbf{X})}_{1\times n} \boldsymbol{\beta}\boldsymbol{\beta}^\top \underbrace{k_h(\mathbf{X}, \mathbf{x}_*)}_{n\times 1} p(\mathbf{x}_*) \, \mathrm{d}\mathbf{x}_* - (\boldsymbol{\beta}^\top \mathbf{q})^2 \\
&= \alpha^2 - \operatorname{tr}\left((\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1} \int k_h(\mathbf{X}, \mathbf{x}_*) k_h(\mathbf{x}_*, \mathbf{X}) p(\mathbf{x}_*) \, \mathrm{d}\mathbf{x}_*\right) \\
&\quad + \boldsymbol{\beta}^\top \underbrace{\int k_h(\mathbf{X}, \mathbf{x}_*) k_h(\mathbf{x}_*, \mathbf{X}) p(\mathbf{x}_*) \, \mathrm{d}\mathbf{x}_*}_{=:\tilde{\mathbf{Q}}} \boldsymbol{\beta} - (\boldsymbol{\beta}^\top \mathbf{q})^2 \\
&= \underbrace{\alpha^2 - \operatorname{tr}\left((\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1} \tilde{\mathbf{Q}}\right)}_{=\mathbb{E}_{\mathbf{x}_*}[\operatorname{var}_h[h(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}]} + \underbrace{\boldsymbol{\beta}^\top \tilde{\mathbf{Q}} \boldsymbol{\beta} - \mu_*^2}_{=\operatorname{var}_{\mathbf{x}_*}[\mathbb{E}_h[h(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}]} ,
\end{aligned}
\tag{2.18}
$$

where we re-arranged the inner products to pull the expressions that are independent of $\mathbf{x}_*$ out of the integrals. The entries of $\tilde{\mathbf{Q}} \in \mathbb{R}^{n \times n}$ are given by

$$
\tilde{Q}_{ij} = \frac{k_h(\mathbf{x}_i, \boldsymbol{\mu}) k_h(\mathbf{x}_j, \boldsymbol{\mu})}{|2\boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|^{\frac{1}{2}}} \exp\left((\tilde{\mathbf{z}}_{ij} - \boldsymbol{\mu})^\top (\boldsymbol{\Sigma} + \tfrac{1}{2}\boldsymbol{\Lambda})^{-1} \boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} (\tilde{\mathbf{z}}_{ij} - \boldsymbol{\mu})\right) \tag{2.19}
$$

with $\tilde{\mathbf{z}}_{ij} \coloneqq \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j)$. Like the predicted mean in equation (2.16), the predictive variance depends explicitly on the mean $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ of the input distribution $p(\mathbf{x}_*)$.

Multivariate Predictions

In the multivariate case, the predictive mean vector $\boldsymbol{\mu}_*$ of $p(h(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the collection of all E independently predicted means computed according to equation (2.16). We obtain the predicted mean

$$
\boldsymbol{\mu}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\beta}_1^\top \mathbf{q}_1 & \dots & \boldsymbol{\beta}_E^\top \mathbf{q}_E \end{bmatrix}^\top . \tag{2.20}
$$

Unlike predicting with deterministic inputs, the target dimensions now covary, and the corresponding predictive covariance matrix

$$
\boldsymbol{\Sigma}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma} = \begin{bmatrix} \operatorname{var}[h_1^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] & \dots & \operatorname{cov}[h_1^*, h_E^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \\ \vdots & \ddots & \vdots \\ \operatorname{cov}[h_E^*, h_1^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] & \dots & \operatorname{var}[h_E^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \end{bmatrix} \tag{2.21}
$$

is no longer diagonal. Here, $h_a(\mathbf{x}_*)$ is abbreviated by h_a^*, $a \in \{1, \dots, E\}$. The variances on the diagonal are the predictive variances of the individual target dimensions given in equation (2.18). The cross-covariances are given by

$$
\operatorname{cov}[h_a^*, h_b^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{h,\mathbf{x}_*}[h_a^* h_b^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] - \mathbb{E}_{h,\mathbf{x}_*}[h_a^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \mathbb{E}_{h,\mathbf{x}_*}[h_b^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] .
$$

With $p(\mathbf{x}_*) = \mathcal{N}(\mathbf{x}_* \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ we obtain

$$
\mathbb{E}_{h,\mathbf{x}_*}[h_a^* h_b^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \overset{(2.16)}{=} \mathbb{E}_{\mathbf{x}_*}\left[\mathbb{E}_{h_a}[h_a^*|\mathbf{x}_*] \, \mathbb{E}_{h_b}[h_b^*|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right] \overset{(2.11)}{=} \int m_h^a(\mathbf{x}_*) m_h^b(\mathbf{x}_*) p(\mathbf{x}_*) \, \mathrm{d}\mathbf{x}_* \tag{2.22}
$$

due to the conditional independence of h_a and h_b given $\mathbf{x}_*$. According to equation (2.11), the mean function m_h^a is

$$m_h^a(\mathbf{x}_*) = k_h^a(\mathbf{x}_*, \mathbf{X}) \underbrace{(\mathbf{K}_a + \sigma_{\varepsilon_a}^2 \mathbf{I})^{-1} \mathbf{y}_a}_{=:\boldsymbol{\beta}_a}, \tag{2.23}$$

which leads to

$$\begin{aligned}
\mathbb{E}_{h,\mathbf{x}_*}[h_a^* \, h_b^* | \boldsymbol{\mu}, \boldsymbol{\Sigma}] &\overset{(2.22)}{=} \int m_h^a(\mathbf{x}_*) m_h^b(\mathbf{x}_*) p(\mathbf{x}_*) \, \mathrm{d}\mathbf{x}_* \overset{(2.23)}{=} \int \underbrace{k_h^a(\mathbf{x}_*, \mathbf{X})\boldsymbol{\beta}_a}_{\in \mathbb{R}} \underbrace{k_h^b(\mathbf{x}_*, \mathbf{X})\boldsymbol{\beta}_b}_{\in \mathbb{R}} p(\mathbf{x}_*) \, \mathrm{d}\mathbf{x}_* \\
&= \boldsymbol{\beta}_a^\top \underbrace{\int k_h^a(\mathbf{x}_*, \mathbf{X})^\top \, k_h^b(\mathbf{x}_*, \mathbf{X}) p(\mathbf{x}_*) \, \mathrm{d}\mathbf{x}_*}_{=:\mathbf{Q}} \boldsymbol{\beta}_b \,,
\end{aligned}$$

where we re-arranged the inner products to pull terms out of the integral that are independent of the test input $\mathbf{x}_*$. The entries of $\mathbf{Q}$ are given by

$$\begin{aligned}
Q_{ij} &= \alpha_a^2 \alpha_b^2 |(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1})\boldsymbol{\Sigma} + \mathbf{I}|^{-\frac{1}{2}} \\
&\quad \times \exp\big(-\tfrac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^\top (\boldsymbol{\Lambda}_a + \boldsymbol{\Lambda}_b)^{-1}(\mathbf{x}_i - \mathbf{x}_j)\big) \\
&\quad \times \exp\big(-\tfrac{1}{2}(\hat{\mathbf{z}}_{ij} - \boldsymbol{\mu})^\top ((\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1})^{-1} + \boldsymbol{\Sigma})^{-1}(\hat{\mathbf{z}}_{ij} - \boldsymbol{\mu})\big) \,, \\
\hat{\mathbf{z}}_{ij} &\coloneqq \boldsymbol{\Lambda}_b(\boldsymbol{\Lambda}_a + \boldsymbol{\Lambda}_b)^{-1}\mathbf{x}_i + \boldsymbol{\Lambda}_a(\boldsymbol{\Lambda}_a + \boldsymbol{\Lambda}_b)^{-1}\mathbf{x}_j \,.
\end{aligned} \tag{2.24}$$

We define $\mathbf{R} \coloneqq \boldsymbol{\Sigma}(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1}) + \mathbf{I}$ and $\mathbf{z}_{ij} \coloneqq \boldsymbol{\Lambda}_a^{-1}(\mathbf{x}_i - \boldsymbol{\mu}) + \boldsymbol{\Lambda}_b^{-1}(\mathbf{x}_j - \boldsymbol{\mu})$. Using the matrix inversion lemma from Appendix A.4, we obtain an alternative expression

$$\begin{aligned}
Q_{ij} &= \frac{k_a(\mathbf{x}_i, \boldsymbol{\mu}) k_b(\mathbf{x}_j, \boldsymbol{\mu})}{\sqrt{|\mathbf{R}|}} \exp\left(\tfrac{1}{2}\mathbf{z}_{ij}^\top \mathbf{R}^{-1}\boldsymbol{\Sigma}\mathbf{z}_{ij}\right) = \frac{\exp(n_{ij}^2)}{\sqrt{|\mathbf{R}|}} \,, \\
n_{ij}^2 &= 2(\log(\alpha_a) + \log(\alpha_b)) - \tfrac{1}{2}\big((\mathbf{x}_i - \boldsymbol{\mu})^\top \boldsymbol{\Lambda}_a^{-1}(\mathbf{x}_i - \boldsymbol{\mu}) + (\mathbf{x}_j - \boldsymbol{\mu})^\top \boldsymbol{\Lambda}_b^{-1}(\mathbf{x}_j - \boldsymbol{\mu}) - \mathbf{z}_{ij}^\top \mathbf{R}^{-1}\boldsymbol{\Sigma}\mathbf{z}_{ij}\big) \,,
\end{aligned} \tag{2.25}$$

which can be used for a numerically fairly stable implementation: First, no matrix inverse of potentially low-rank matrices, such as $\boldsymbol{\Sigma}$, is required.[4] Second, due to limited machine precision, the multiplication of exponentials in equation (2.24) is reformulated as an exponential of a sum. We emphasize that $\mathbf{R}^{-1}\boldsymbol{\Sigma} = (\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1} + \boldsymbol{\Sigma}^{-1})^{-1}$ is symmetric. The matrix $\mathbf{Q}$ depends on the covariance of the input distribution as well as on the SE kernels k_h^a and k_h^b. Note that $\mathbf{Q}$ in equation (2.25) equals $\tilde{\mathbf{Q}}$ in equation (2.19) for identical target dimensions $a = b$.

To summarize, the entries of the covariance matrix of the predictive distribution are

$$\operatorname{cov}[h_a^*, h_b^* | \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \begin{cases} \boldsymbol{\beta}_a^\top \mathbf{Q} \boldsymbol{\beta}_b - \mathbb{E}_{h,\mathbf{x}_*}[h_a^* | \boldsymbol{\mu}, \boldsymbol{\Sigma}] \, \mathbb{E}_{h,\mathbf{x}_*}[h_b^* | \boldsymbol{\mu}, \boldsymbol{\Sigma}] \,, & a \neq b \\ \boldsymbol{\beta}_a^\top \mathbf{Q} \boldsymbol{\beta}_a - \mathbb{E}_{h,\mathbf{x}_*}[h_a^* | \boldsymbol{\mu}, \boldsymbol{\Sigma}]^2 + \alpha_a^2 - \operatorname{tr}\big((\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{Q}\big) \,, & a = b \,. \end{cases} \tag{2.26}$$

If $a = b$, we have to include the term $\mathbb{E}_{\mathbf{x}_*}[\operatorname{var}_h[h(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \alpha_a^2 - \operatorname{tr}\big((\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{Q}\big)$, which is zero for $a \neq b$ due to the assumption that the target dimensions a and b are conditionally independent given the input.

These results yield the exact mean $\boldsymbol{\mu}_*$ and the exact covariance $\boldsymbol{\Sigma}_*$ of the generally non-Gaussian predictive distribution $p(h(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $h \sim \mathcal{GP}$ and $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Table 2.1 summarizes how to predict with Gaussian processes.

2.3.3 Input-Output Covariance

It is sometimes necessary to compute the covariance between a test input $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and the corresponding predicted function value $h(\mathbf{x}_*) \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$. As an example suppose that the joint distribution

$$p(\mathbf{x}_*, h(\mathbf{x}_*)) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma} & \boldsymbol{\Sigma}_{x_*,h} \\ \boldsymbol{\Sigma}_{x_*,h}^\top & \boldsymbol{\Sigma}_* \end{bmatrix}\right) \tag{2.27}$$

[4] In particular, the formulation in equation (2.25) allows for (a fairly inefficient) computation of the predictive covariance matrix if the input is deterministic, that is, $\boldsymbol{\Sigma} \equiv \mathbf{0}$.

Table 2.1: Predictions with Gaussian processes—overview.

mean prediction	certain input	uncertain input
univariate	eq. (2.11)	eq. (2.16)
multivariate	eq. (2.13)	eq. (2.20)
covariance prediction		
univariate	eq. (2.12)	eq. (2.18)
multivariate	eq. (2.14)	eq. (2.21), eq. (2.26)

is desired. The marginal distributions of $\mathbf{x}_*$ and $h(\mathbf{x}_*)$ are either given or computed according to Section 2.3.2. The missing piece is the cross-covariance matrix

$$\boldsymbol{\Sigma}_{x_*,h} = \mathbb{E}_{\mathbf{x}_*,h}[\mathbf{x}_* h(\mathbf{x}_*)^\top] - \mathbb{E}_{\mathbf{x}_*}[\mathbf{x}_*]\,\mathbb{E}_{h,\mathbf{x}_*}[h(\mathbf{x}_*)]^\top = \mathbb{E}_{\mathbf{x}_*,h}[\mathbf{x}_* h(\mathbf{x}_*)^\top] - \boldsymbol{\mu}\boldsymbol{\mu}_*^\top\,.$$

For each target dimension $a = 1, \ldots, E$, we compute $\mathbb{E}_{\mathbf{x}_*,h_a}[\mathbf{x}_*\, h_a(\mathbf{x}_*)]$ as

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}_*,h_a}[\mathbf{x}_*\, h_a(\mathbf{x}_*)|\boldsymbol{\mu},\boldsymbol{\Sigma}] = \mathbb{E}_{\mathbf{x}_*}[\mathbf{x}_*\,\mathbb{E}_{h_a}[h_a(\mathbf{x}_*)|\mathbf{x}_*]] &= \int \mathbf{x}_*\, m_h^a(\mathbf{x}_*) p(\mathbf{x}_*)\,\mathrm{d}\mathbf{x}_* \\
&\overset{(2.11)}{=} \int \mathbf{x}_* \left(\sum_{i=1}^n \beta_i^a\, k_h^a(\mathbf{x}_*,\mathbf{x}_i)\right) p(\mathbf{x}_*)\,\mathrm{d}\mathbf{x}_* \\
&= \sum_{i=1}^n \beta_i^a \int \mathbf{x}_*\, k_h^a(\mathbf{x}_*,\mathbf{x}_i) p(\mathbf{x}_*)\,\mathrm{d}\mathbf{x}_* \\
&= \sum_{i=1}^n \beta_i^a \int \mathbf{x}_*\, c_1\, \mathcal{N}(\mathbf{x}_*|\mathbf{x}_i,\boldsymbol{\Lambda}_a)\mathcal{N}(\mathbf{x}_*|\boldsymbol{\mu},\boldsymbol{\Sigma})\,\mathrm{d}\mathbf{x}_*\,, \qquad (2.28)
\end{aligned}$$

where

$$c_1^{-1} = \alpha^{-2}(2\pi)^{-\frac{D}{2}}|\boldsymbol{\Lambda}_a|^{-\frac{1}{2}}$$

is the "normalizing constant" of the SE kernel k_h^a. Note that $\mathbf{x}_i$, $i = 1, \ldots, n$, are the training inputs of the GP. The product of the two Gaussians in equation (2.28) results in a new (unnormalized) Gaussian $c_2^{-1}\mathcal{N}(\mathbf{x}_*\,|\,\boldsymbol{\psi},\boldsymbol{\Psi})$ with

$$\begin{aligned}
c_2^{-1} &= (2\pi)^{-\frac{D}{2}}|\boldsymbol{\Lambda}_a + \boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\big(-\tfrac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu})^\top(\boldsymbol{\Lambda}_a + \boldsymbol{\Sigma})^{-1}(\mathbf{x}_i - \boldsymbol{\mu})\big)\,, \\
\boldsymbol{\Psi} &= (\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Sigma}^{-1})^{-1}\,, \\
\boldsymbol{\psi} &= \boldsymbol{\Psi}(\boldsymbol{\Lambda}_a^{-1}\mathbf{x}_i + \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})\,.
\end{aligned}$$

The mean $\boldsymbol{\psi}$ of this new Gaussian is a function of $\mathbf{x}_i$ and $\boldsymbol{\mu}$ and is denoted by $\boldsymbol{\psi}(\mathbf{x}_i,\boldsymbol{\mu})$. Pulling all constants out of the integral in equation (2.28), the integral determines the expected value of the product of the two Gaussians, $\boldsymbol{\psi}(\mathbf{x}_i,\boldsymbol{\mu})$. Hence, we finally obtain

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}_*,h_a}[\mathbf{x}_* h_a(\mathbf{x}_*)|\boldsymbol{\mu},\boldsymbol{\Sigma}] &= c_1 c_2^{-1}\sum_{i=1}^n \beta_i^a \boldsymbol{\psi}(\mathbf{x}_i,\boldsymbol{\mu})\,,\ a = 1,\ldots,E\,, \\
\operatorname{cov}[\mathbf{x}_*, h_a(\mathbf{x}_*)|\boldsymbol{\mu},\boldsymbol{\Sigma}] &= c_1 c_2^{-1}\sum_{i=1}^n \beta_i^a \boldsymbol{\psi}(\mathbf{x}_i,\boldsymbol{\mu}) - \boldsymbol{\mu}\boldsymbol{\mu}_*^\top\,,\ a = 1,\ldots,E\,,
\end{aligned}$$

and the joint distribution $p(\mathbf{x}_*, h(\mathbf{x}_*))$ in equation (2.27) is fully determined.

2.3.4 Computational Complexity

For an n-sized training set, training a GP using gradient-based evidence maximization requires $\mathcal{O}(n^3)$ computations per gradient step due to the inversion of the kernel matrix $\mathbf{K}$ in equation (2.9). For E different target dimensions, this sums up to $\mathcal{O}(En^3)$ operations.

Predictions with uncertain inputs according to Section 2.3.2 require $\mathcal{O}(E^2n^2D)$ operations per time step to determine the $\mathbf{Q}$ matrix in equation (2.25). Here, D is the dimensionality of the training inputs, and E is the dimensionality of the training targets.

2.4 Sparse Approximations using Inducing Inputs

A common problem in training and predicting with Gaussian processes is that the computational burden becomes prohibitively expensive when the size of the data set becomes large. Sparse GP approximations aim to reduce the computational burden coming along with training and predictions. The computations in a full GP are dominated by either the inversion of the $n \times n$ kernel matrix $\mathbf{K}$ or the multiplication of $\mathbf{K}$ with vectors, see equations (2.9), (2.11), (2.12), (2.18), and (2.19), for some examples. Typically, sparse approximations aim to find a low-rank approximation of $\mathbf{K}$. Quiñonero-Candela and Rasmussen (2005) describe several sparse approximations within a unifying framework. In the following, we briefly touch upon one class of sparse approximations.

For fixed hyper-parameters, the GP predictive distribution in equations (2.11) and (2.12) can be considered essentially parameterized by the training inputs $\mathbf{X}$ and the training targets $\mathbf{y}$. Snelson and Ghahramani (2006), Snelson (2007), and Titsias (2009) introduce sparse GP approximations that use *inducing inputs*. A representative *pseudo-training set* of fictitious training data $\{\bar{\mathbf{X}}, \bar{\mathbf{h}}\}$ of size M replaces the real data set $\{\mathbf{X}, \mathbf{y}\}$ of size n. Intuitively, Snelson and Ghahramani (2006), Snelson (2007), and Titsias (2009) find pseudo-input locations $\bar{\mathbf{X}}$ in order to predict the targets $\mathbf{y}$ of the real data set optimally. The GP predictive distribution from the pseudo-data set is used as a parameterized marginal likelihood

$$p(\mathbf{y}) = \int p(\mathbf{y}|\bar{\mathbf{h}}) p(\bar{\mathbf{h}}) \, \mathrm{d}\bar{\mathbf{h}} = \int \mathcal{N}\left(\mathbf{y} \,|\, \mathbf{K}_{nM}\mathbf{K}_{MM}^{-1}\bar{\mathbf{h}}, \boldsymbol{\Gamma}\right) \mathcal{N}\left(\bar{\mathbf{h}} \,|\, \mathbf{0}, \mathbf{K}_{MM}\right) \mathrm{d}\bar{\mathbf{h}} = \mathcal{N}\left(\mathbf{y} \,|\, \mathbf{0}, \mathbf{Q}_{nn} + \boldsymbol{\Gamma}\right) ,$$

where the pseudo-targets $\bar{\mathbf{h}}$ have been integrated out and

$$\mathbf{Q}_{nn} \coloneqq \mathbf{K}_{nM}\mathbf{K}_{MM}^{-1}\mathbf{K}_{Mn} , \quad \mathbf{K}_{MM} \coloneqq k_h(\bar{\mathbf{X}}, \bar{\mathbf{X}}) .$$

$\mathbf{Q}_{nn}$ is a low-rank approximation of the full-rank kernel matrix $\mathbf{K}_{nn} = k_h(\mathbf{X}, \mathbf{X})$.[5] The matrix $\boldsymbol{\Gamma} \in \{\boldsymbol{\Gamma}_{\mathrm{FITC}}, \boldsymbol{\Gamma}_{\mathrm{VB}}\}$ depends on the sparse approximation. Snelson and Ghahramani (2006) use

$$\boldsymbol{\Gamma}_{\mathrm{FITC}} \coloneqq \mathrm{diag}(\mathbf{K}_{nn} - \mathbf{Q}_{nn}) + \sigma_\varepsilon^2 \mathbf{I} ,$$

whereas Titsias (2009) employs the matrix

$$\boldsymbol{\Gamma}_{\mathrm{VB}} \coloneqq \sigma_\varepsilon^2 \mathbf{I} ,$$

which is in common with previous sparse approximations by Silverman (1985), Wahba et al. (1999), Smola and Bartlett (2001), Csató and Opper (2002), and Seeger et al. (2003), which are not discussed further in this report. Using the $\boldsymbol{\Gamma}$-notation, the log-marginal likelihood of the sparse GP methods by Titsias (2009) and Snelson and Ghahramani (2006) and Snelson (2007) is given by

$$\log q(\mathbf{y}|\bar{\mathbf{X}}) = -\tfrac{1}{2}\log|\mathbf{Q}_{nn} + \boldsymbol{\Gamma}| - \tfrac{1}{2}\mathbf{y}^\top(\mathbf{Q}_{nn} + \boldsymbol{\Gamma})^{-1}\mathbf{y} - \tfrac{n}{2}\log(2\pi) - \underbrace{\tfrac{1}{2\,\sigma_\varepsilon^2}\mathrm{tr}(\mathbf{K}_{nn} - \mathbf{Q}_{nn})}_{\text{only used by VB}} , \tag{2.29}$$

where the last term can be considered a regularizer and is solely used in the variational approximation by Titsias (2009). The methods by Snelson and Ghahramani (2006) and Titsias (2009) therefore use different marginal likelihoods to be optimized.[6] In particular, Titsias (2009) found a principled way of sidestepping possible overfitting issues by maximizing a variational lower bound of the true log marginal likelihood, that is, he attempts to minimize the KL divergence $\mathrm{KL}(q||p)$ between the approximate marginal likelihood q in equation (2.29) and the true marginal likelihood p in equation (2.9). With the matrix inversion lemma in Appendix A.4, the matrix operations in equation (2.29) do no longer require to invert

[5] For clarity, we added the subscripts that describe the dimensions of the corresponding matrices.

[6] The parameters to be optimized are the hyper-parameters of the covariance function plus the input locations $\bar{\mathbf{X}}$.

full $n \times n$ matrices. We rewrite

$$\begin{aligned}(\mathbf{Q}_{nn} + \mathbf{\Gamma})^{-1} &= (\mathbf{K}_{nM}\mathbf{K}_{MM}^{-1}\mathbf{K}_{Mn} + \mathbf{\Gamma})^{-1} \\ &= \mathbf{\Gamma}^{-1} - \mathbf{\Gamma}^{-1}\mathbf{K}_{nM}(\mathbf{K}_{MM} + \mathbf{K}_{Mn}\mathbf{\Gamma}^{-1}\mathbf{K}_{nM})^{-1}\mathbf{K}_{Mn}\mathbf{\Gamma}^{-1}\,, \\ \log|\mathbf{Q}_{nn} + \mathbf{\Gamma}| &= \log\left(|\mathbf{K}_{nM}\mathbf{K}_{MM}^{-1}\mathbf{K}_{Mn} + \mathbf{\Gamma}|\right) = \log\left(|\mathbf{\Gamma}||\mathbf{K}_{MM}^{-1}||\mathbf{K}_{MM} + \mathbf{K}_{Mn}\mathbf{\Gamma}^{-1}\mathbf{K}_{nM}|\right) \\ &= \log|\mathbf{\Gamma}| - \log|\mathbf{K}_{MM}| + \log|\mathbf{K}_{MM} + \mathbf{K}_{Mn}\mathbf{\Gamma}^{-1}\mathbf{K}_{nM}|\,,\end{aligned}$$

where the inversion of $\mathbf{\Gamma}$ is computationally cheap since $\mathbf{\Gamma}$ is a diagonal matrix.

The predictive distribution at a test input $\mathbf{x}_*$ is given by the mean and the variance

$$\mathbb{E}_h[h(\mathbf{x}_*)] = k_h(\mathbf{x}_*, \bar{\mathbf{X}})\mathbf{B}^{-1}\mathbf{K}_{Mn}\mathbf{\Gamma}^{-1}\mathbf{y}\,, \tag{2.30}$$

$$\mathrm{var}_h[h(\mathbf{x}_*)] = k_h(\mathbf{x}_*, \mathbf{x}_*) - k_h(\mathbf{x}_*, \bar{\mathbf{X}})(\mathbf{K}_{MM}^{-1} - \mathbf{B}^{-1})k_h(\bar{\mathbf{X}}, \mathbf{x}_*)\,, \tag{2.31}$$

respectively, with $\mathbf{B} \coloneqq \mathbf{K}_{MM} + \mathbf{K}_{Mn}\mathbf{\Gamma}^{-1}\mathbf{K}_{nM}$.

One key difference between the algorithms is that the algorithm by Snelson (2007) can be interpreted as a GP with heteroscedastic noise whereas the algorithm by Titsias (2009) maintains a GP with homoscedastic noise resembling the original GP. Note that both sparse methods are not degenerate, that means, they have sensible variances far away from the data.[7]

2.4.1 Computational Complexity

Let M be the size of the pseudo training set, and n be the size of the real data set with $M \ll n$. The sparse approximations allow for training a GP in $\mathcal{O}(nDM^2)$ (see equation (2.29)) and predicting in $\mathcal{O}(DM)$ for the mean in equation (2.30) and $\mathcal{O}(DM^2)$ for the variance in equation (2.31), respectively, where D is the dimension of the input vectors.[8] Multivariate predictions (with uncertain inputs) then require $\mathcal{O}(ME)$ computations for the mean vector and $\mathcal{O}(E^2M^2D)$ computations for the covariance matrix.

2.5 Further Reading

In geostatistics and spatial statistics, Gaussian processes are known as *kriging*. Classical references for kriging are the books by Matheron (1973), Cressie (1993), and Stein (1999). O'Hagan (1978) first describes GPs as a non-parametric prior over functions. Neal (1996, Chapter 2.1) shows that a neural network converges to a GP if the number of hidden units tends to infinity and the weights and the biases have zero-mean Gaussian priors. Williams (1995), Williams and Rasmussen (1996), and Rasmussen (1996) introduced GPs into the machine learning community. For details on Gaussian processes in the context of machine learning, we refer to the books by Rasmussen and Williams (2006), Bishop (2006), and MacKay (2003).

GP prediction with uncertain inputs has previously been discussed in the papers by Girard et al. (2002, 2003), Quiñonero-Candela et al. (2003a,b), Kuss (2006), and Ko et al. (2007). All methods approximate the true predictive distribution by a Gaussian distribution. Ko et al. (2007) propose a first-order Taylor series expansion of the mean function or a deterministic sampling method to obtain approximate moments of the true predictive distribution. Girard et al. (2002, 2003) use a second-order Taylor series expansion of the mean function and the covariance function to compute the predictive moments approximately. Quiñonero-Candela et al. (2003a,b) derive the analytic expressions for the exact predictive moments and show its superiority over the second-order Taylor series expansion by Girard et al. (2002, 2003).

[7] By contrast, the predictive variances in a radial basis function network collapse to zero far away from the means of the basis functions.

[8] The vector $\mathbf{B}^{-1}\mathbf{K}_{Mn}\mathbf{\Gamma}^{-1}\mathbf{y}$ and the matrix $\mathbf{K}_{MM}^{-1} - \mathbf{B}^{-1}$ can be computed in advance since they are independent of $\mathbf{x}_*$.

Chapter 3

Probabilistic Models for Efficient Learning in Control

3.1 General Setup

We consider discrete-time control problems with continuous-valued states $\mathbf{x} \in \mathbb{R}^D$ and external control signals (actions) $\mathbf{u} \in \mathbb{R}^F$. The dynamics of the system are described by a Markov decision process (MDP), a computational framework for decision-making under uncertainty. An MDP is a tuple of four objects: the state space, the control space (also called the action space), the one-step transition function f, and the immediate cost function $c(\mathbf{x})$ that penalizes the distance to a given target $\mathbf{x}_{\text{target}}$. If not stated otherwise, we assume that all states can be measured exactly and are fully observable. However, the deterministic transition dynamics

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \tag{3.1}$$

are not known in advance, but they are assumed to be smooth. We additionally assume that the immediate cost function $c(\,\cdot\,)$ is not unknown, but instead it is a design criterion.[1] The graphical model of the considered setup is shown in Figure 3.1.

The goal in RL is to find a *policy* π^* that minimizes the expected long-term cost

$$V^\pi(\mathbf{x}_0) = \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=0}^{T} c(\mathbf{x}_t)\right] = \sum_{t=0}^{T} \mathbb{E}_{\mathbf{x}_t}\left[c(\mathbf{x}_t)\right] \tag{3.2}$$

[1] In the context of control applications, this assumption is common (Bertsekas, 2005) although it does not comply with the most general RL setup.

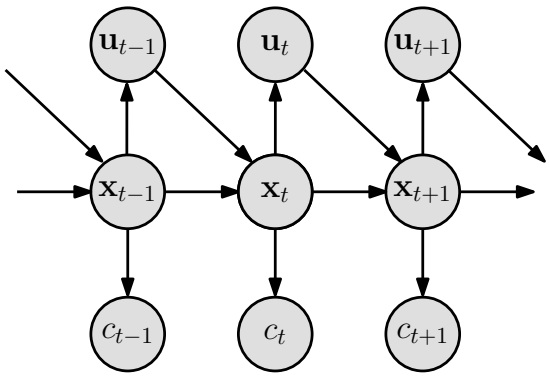


Figure 3.1: Graphical model for the problem setup. The state $\mathbf{x}$ of the system follows Markovian dynamics and can be modified by applying external forces $\mathbf{u}$. The cost $c_t := c(\mathbf{x}_t)$ can either be computed or observed.

of following a policy π for a finite horizon of T time steps. Here, $\boldsymbol{\tau} \coloneqq (\mathbf{x}_0, \ldots, \mathbf{x}_T)$ denotes the trajectory of states visited. The function V^π is called the *value function*, and $V^\pi(\mathbf{x}_0)$ is called the *value of the state* $\mathbf{x}_0$ under policy π.

A policy π is defined as a function that maps states to actions. We consider stationary deterministic policies that are parameterized by a vector $\boldsymbol{\psi}$. Therefore, $\mathbf{u}_{t-1} = \pi(\mathbf{x}_{t-1}, \boldsymbol{\psi})$ and $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \pi(\mathbf{x}_{t-1}, \boldsymbol{\psi}))$ meaning that a state $\mathbf{x}_t$ at time t depends implicitly on the policy parameters $\boldsymbol{\psi}$. Using this notation, we can now formulate our objective more precisely:

> In the context of motor control problems, we aim to find a good policy π^* that leads to a low value $V^{\pi^*}(\mathbf{x}_0)$ given an initial state distribution $p(\mathbf{x}_0)$.[2] We assume that no task-specific expert knowledge is available. Furthermore, we desire to find the policy using only a small number of interactions with the system. The setup can be considered an RL problem with very limited interaction resources.

Interacting with the system by applying actions/control signals and observing the system's response at each time step yields experience. Experience from interactions can be used for two purposes: It can be used either to update the current model of the system (indirect RL, model-based RL) or it can be used to improve the value function and/or the policy directly (direct RL, model-free RL), or combinations of the two.

In model-based RL the learned model can be used to *simulate* the system internally without interacting with the system. The policy is then optimized based on these simulations. A major weakness of model-based RL is that the quality of the found policy when being applied to the real system strongly depends on the accuracy of the model employed (model bias). If the model does not capture the important characteristics of the system, the optimized policy can be far from optimal when applying it to the system. Schaal (1997), Atkeson and Schaal (1997), Atkeson and Santamaría (1997), and many others report problems with this type of "incorrect" models. The bias toward a particular model and the resulting problems can be sidestepped by using model-free RL. Model-free algorithms do not learn a model of the system. Instead, they use experience from interaction to determine an optimal policy directly (Sutton and Barto, 1998; Bertsekas and Tsitsiklis, 1996). Unlike model-based RL, model-free RL is statistically inefficient, but computationally congenial since it learns by bootstrapping.

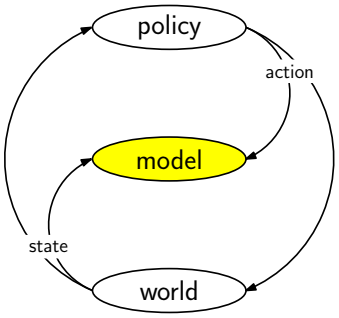
Humans and animals employ model-based goal-directed learning when only a moderate amount of experience is available. After having obtained a lot of experience (intensive training), humans and animals switch to model-free learning as concluded by Daw et al. (2005). Our objective is to learn control tasks with a small number of required interactions with the system. Thus, we follow a model-based approach. By using experience from interaction with the system, we learn a statistical model of the system to generate simulated experience.

In a model-based RL setup, we generally distinguish between two phases: interaction with the real world[3] and internal simulation. Figure 3.2 illustrates these phases. Typically, the interaction and the simulation phase alternate. In the interaction phase, a policy is applied to the real world. Data in form of (state, action, successor state)-tuples $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$ are collected to train a model for $f : (\mathbf{x}_t, \mathbf{u}_t) \mapsto \mathbf{x}_{t+1}$ of the world. In the simulation phase, this model is used to emulate the world and to generate simulated experience. The policy is optimized using the simulated experience of the model.

Figure 3.2 also emphasizes the dependency of the policy on the model: The policy is refined in the light of the model of the world, not the world itself (see Figure 3.2(b)). If the model is not sufficiently similar to the real world, the model-optimal policy is not good when applied to the real world. In the context of motor control, the world corresponds to a dynamic system. Abbeel et al. (2006) used approximate models for RL. Their algorithm was initialized with a policy, which was locally optimal for the initial (approximate) model of the dynamics. Moreover, a time-dependent bias term was used to account for discrepancies between real experience and the model's predictions. To generate approximate models, expert knowledge in terms of a parametric dynamics model was used, where the model parameters were randomly initialized around ground truth or good-guess values. A different approach to deal with model inaccuracies is to add a noise term to the system equation, a common practice in systems engineering when the system cannot be identified sufficiently well. All these approaches attempt to express uncertainty about the model of the system.

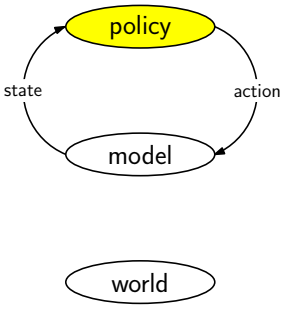
[2] The *distribution* of the initial state is not necessary, but finding a good policy for a single state $\mathbf{x}_0$ is often not an interesting problem in continuous-valued state spaces. Instead, a good policy for states *around* $\mathbf{x}_0$ is more useful.

[3] We briefly switch to the RL expression "world" instead of the "dynamic system" used in control.



interaction

(a) Interaction phase. An action is applied to the real world. The world changes its state and returns the state to the policy. The policy selects a corresponding action and applies it to the real world again. The model takes the applied actions and the states of the real world and refines itself.



simulation

(b) Simulation phase. An action is applied to the model of the world. The model simulates the real world and returns a state of which it thinks the world might be in. The policy determines an action according to the state returned by the model and applies it again. Using this simulated experience, the policy is refined.

Figure 3.2: Two alternating phases in model-based reinforcement learning. We distinguish between the real world, an internal model of the real world, and a policy. Yellow color indicates that the corresponding component is being refined. In the interaction phase, the model of the world is refined. In the simulation phase, this model is used to simulate experience, which in turn is used to improve the policy. The improved policy can be used in the next interaction phase.

top layer: policy optimization/learning	π^*
intermediate layer: (approximate) inference	V^π
bottom layer: learning the transition dynamics	f

Figure 3.3: The learning problem can be divided into three hierarchical problems. At the bottom layer, the transition dynamics f are learned. Based on the transition dynamics, the value function V^π can be evaluated using approximate inference techniques. At the top layer, an optimal control problem has to be solved to determine a model-optimal policy π^*.

In our learning approach, we require not only a system model for efficient RL, but additionally require that this system model is probabilistic and faithfully describes its own accuracy to treat model uncertainties in a principled way. Humans do something similar: Körding and Wolpert (2004a) and Körding and Wolpert (2006) show that if we have only little experience, we employ an internal forward model for predictions and average over the uncertainty when predicting or making decisions.

In our model-based setup, the learning problem can be decomposed into a hierarchy of three sub-problems as described in Figure 3.3. At the bottom level, a probabilistic model of the transition function f is learned (Section 3.3). Given the model of the transition dynamics and a policy π, the expected long-term cost in equation (3.2) is evaluated. This *policy evaluation* requires the computation of the predictive state distributions $p(\mathbf{x}_t)$ for $t = 1, \ldots, T$ (intermediate layer in Figure 3.3 and Section 3.4). At the top layer (Section 3.5), the policy parameters $\boldsymbol{\psi}$ are optimized based on the result of the policy evaluation, which is called an *indirect policy search*. The search is typically non-convex and requires iterative optimization techniques. Therefore, the policy evaluation and policy improvement steps alternate until the policy search converges to a local optimum. For given transition dynamics, the two top layers in Figure 3.3 correspond to an optimal control problem.

3.2 High-Level Perspective

Before going into details, Algorithm 1 describes the proposed learning algorithm at a high level. Initially, we set the policy to random (line 1), that is, we apply actions sampled from a uniform distribution. The

Algorithm 1 Efficient RL for control

```
1: set policy to random                                   ▷ policy initialization
2: loop
3:     execute policy                                     ▷ interaction
4:     record collected experience
5:     learn probabilistic dynamics model                 ▷ bottom layer
6:     loop                                               ▷ policy search
7:         simulate system with policy π                  ▷ intermediate layer
8:         compute expected long-term cost V^π, eq. (3.2) ▷ policy evaluation
9:         improve policy                                 ▷ top layer
10:    end loop
11: end loop
```

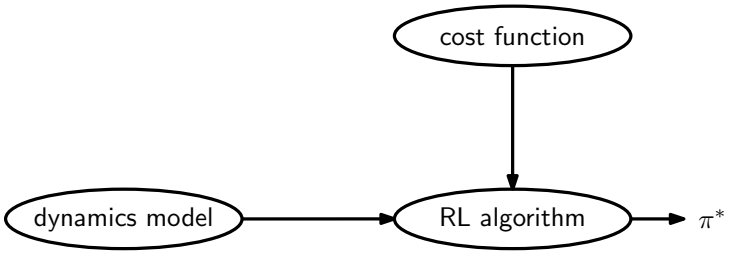


Figure 3.4: Three necessary components in an RL framework. Learning a good policy π^* requires determining the dynamics model, specifying a cost function, and then applying an RL algorithm. The interplay of these three components can be crucial for the success.

framework involves learning in two stages: First, when interacting with the system (line 3), that is we follow the current policy, experience is collected (line 4), and the internal probabilistic dynamics model is updated based on both historical and novel observations (line 5). Second, the policy is refined in the light of the updated probabilistic dynamics model (loop over lines 7–9) by using (approximate) inference techniques for the policy evaluation and gradient-based optimization for the policy improvement. The model-optimized policy is applied to the system (line 3) to gather novel experience (line 4). These two stages of learning correspond to the interaction phase and the simulation phase, respectively, which are described in Figure 3.2. The subsequent model update (line 5) accounts for possible discrepancies between the predicted and the actually encountered state trajectory.

With increasing experience, the probabilistic model describes the dynamics with high certainty in regions of the state space that have been explored well. If the algorithm finds a solution to the learning problem, the well-explored regions of the state space contain trajectories with low expected long-term cost. Note that the dynamics model is updated after each trial and not online. Therefore, Algorithm 1 describes batch learning.

Three components are crucial to determine a good policy π^* using model-based RL: a dynamics model, a cost function, and the RL algorithm itself. Figure 3.4 illustrates the relationship amongst these three components, which have to complement each other for successful RL. A bad interplay of these three components can lead to a failure of the learning algorithm. In our framework, the dynamics model is required to be probabilistic and is implemented by a Gaussian process (Section 3.3). The RL algorithm is an indirect gradient-based policy search algorithm using approximate inference for policy evaluation (Section 3.4) and gradient-based optimization techniques for policy improvement (Section 3.5). The cost function we use in our RL framework is a saturating function (Section 3.6).

3.3 Bottom Layer: Learning the Transition Dynamics

A (generative) model for the transition dynamics f in equation (3.1) is a compact statistical representation of collected experience originating from interacting with the system. To compute the expected long-term cost in equation (3.2), we employ the model to predict the evolution of the system T time steps ahead.

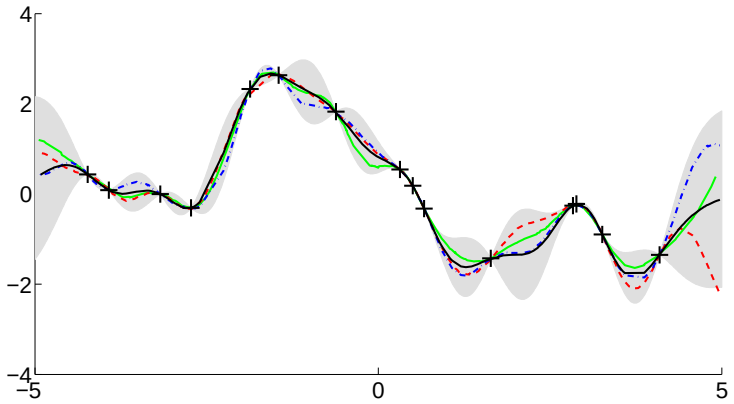


Figure 3.5: Gaussian process posterior as a distribution over transition functions. The x-axis are state-action pairs, the y-axis represents the successor states. The mean transition function is black, the shaded area represents the model uncertainty. The crosses are observed successor states for given state-action pairs. The colored functions are transition function samples drawn from the GP posterior distribution.

With a smoothness prior on the transition function in equation (3.1), the model can generalize from previously observed data to states that have not been visited. Crucially, in order for the predictions to reflect reality as faithfully as possible, the dynamics model must coherently represent the accuracy of the model itself. For example, if a simulated state is encountered in a part of the state space about which not much knowledge has been acquired, the model must express this uncertainty, and not simply assume that its best guess is close to the truth. A probabilistic model captures and quantifies both knowledge and uncertainty. By averaging according to the model distribution, we explicitly incorporate the uncertainty when predicting.

We propose learning the short-term transition dynamics f in equation (3.1) by using Gaussian process models (see Chapter 2 and the references therein). The GP can be considered a model that describes all plausible transition functions by a distribution over them. Let us consider Figure 3.5, which is a repetition of Figure 2.3(b) in Chapter 2, to clarify this point: The observed data (black crosses) now represent the set of observed successor states $f(\mathbf{x}_i, \mathbf{u}_i)$ for a finite number n of state-action pairs $(\mathbf{x}_i, \mathbf{u}_i)$, which are the projection of the black crosses onto the x-axis. The GP model trained on this data set is represented by the mean function in black and the shaded area showing the model's uncertainty. For novel state-action pairs $(\mathbf{x}_*, \mathbf{u}_*)$ that are close to previously encountered state-action pairs, the predicted successor state $f(\mathbf{x}_*, \mathbf{u}_*)$ is fairly certain (see for instance a state-action pair close to the origin of the x-axis). If we move away from the data, the model uncertainty increases, which is illustrated by the bumps of the shading between the crosses (see for example a state-action pair around $+2$ on the x-axis). The increase in uncertainty is reasonable since the model cannot be certain about the function values for a test input $(\mathbf{x}_*, \mathbf{u}_*)$ that is not close to the training set $(\mathbf{x}_i, \mathbf{u}_i)$. Since the GP is non-degenerate[4], far away from the training set, the model uncertainty falls back to the prior uncertainty, which can be seen at the left end or right end of the figure. The GP model captures all transition functions that plausibly could have generated the data. Examples are given by the three colored functions in Figure 3.5. With increasing experience, the probabilistic model gets more confident about its own accuracy and eventually converges to the true function (if the true function is in the class of smooth functions).

For a D-dimensional state space, we use D separate GPs, one for each state dimension. The GP dynamics models take as input a representation of state-action pairs $(\mathbf{x}_i, \mathbf{u}_i)$, $i = 1, \ldots, n$. The corresponding training targets for the dth target dimension are

$$\Delta x_{id} \coloneqq f_d(\mathbf{x}_i, \mathbf{u}_i) - x_{id}\,, \;\; d = 1, \ldots, D\,, \tag{3.3}$$

where f_d maps the input to the dth dimension of the successor state. The GP targets Δx_{id} are the differences between the dth dimension of a state $\mathbf{x}_i$ and the dth dimension of the successor state $f(\mathbf{x}_i, \mathbf{u}_i)$

[4]With "degeneracy" we mean that the uncertainty declines to zero when going away from the training set. The non-degeneracy is due to the fact that in this report the GP is an infinite model, where the SE kernel has an infinite number of non-zero eigenvalues. Any finite model with dimension N gives rise to a degenerate kernel with $\leq N$ non-zero eigenvalues.

of an input $(\mathbf{x}_i, \mathbf{u}_i)$. As opposed to learning the function values directly, learning the differences can be advantageous since they vary less than the original function. Learning differences Δx_{id} approximately corresponds to learning the gradient of the function. The mean and the variance of the Gaussian successor state distribution $p(f_d(\mathbf{x}_*, \mathbf{u}_*))$ for a deterministically given state-action pair $(\mathbf{x}_*, \mathbf{u}_*)$ are given by

$$\mathbb{E}_f[f_d(\mathbf{x}_*, \mathbf{u}_*)] = x_{*d} + \mathbb{E}_f[\Delta x_{*d}]\,, \tag{3.4}$$

$$\mathrm{var}_f[f_d(\mathbf{x}_*, \mathbf{u}_*)] = \mathrm{var}_f[\Delta x_{*d}]\,, \tag{3.5}$$

respectively, $d = 1, \ldots, D$. Note that this predictive distribution reflects the uncertainty about the underlying function. The full predictive state distribution is then given by

$$p(f(\mathbf{x}_*, \mathbf{u}_*)) = \mathcal{N}\left(\begin{bmatrix} \mathbb{E}_f[f_1(\mathbf{x}_*, \mathbf{u}_*)] \\ \vdots \\ \mathbb{E}_f[f_D(\mathbf{x}_*, \mathbf{u}_*)] \end{bmatrix}, \begin{bmatrix} \mathrm{var}_f[f_1(\mathbf{x}_*, \mathbf{u}_*)] & 0 & \ldots & 0 \\ 0 & \ddots & & 0 \\ 0 & \ldots & 0 & \mathrm{var}_f[f_D(\mathbf{x}_*, \mathbf{u}_*)] \end{bmatrix}\right), \tag{3.6}$$

where the covariance matrix is diagonal. Note that the individual means and variances require averaging over the model uncertainty, which is indicated by the subscript f. The hyper-parameters of the D dynamics models are trained by evidence maximization. Details are given in Section 2.2.4 or in the book by Rasmussen and Williams (2006).

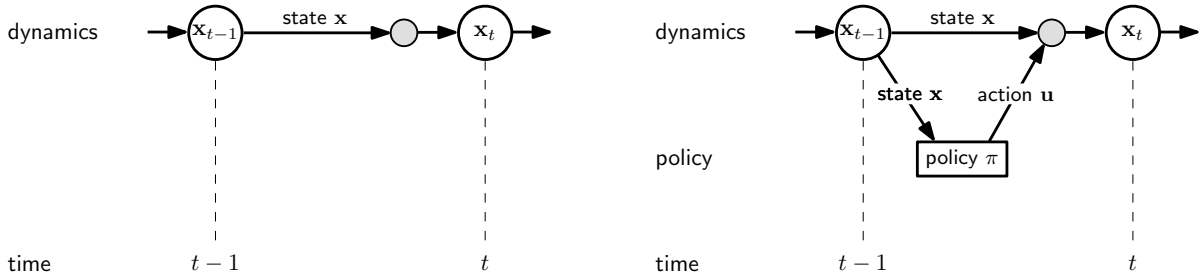
The advantages of using probabilistic GPs to model the transition dynamics are threefold: First, a parametric structure of the underlying function does not need to be known in advance. Instead, a probabilistic model for the latent transition dynamics is learned directly using the current experience captured by the training set. Second, the GP model represents uncertainty coherently. Consider Figure 3.5: Instead of simply interpolating the observations (crosses), a GP explicitly models its uncertainty about the underlying function between observations. Third, the GP "knows" when it does not know much: Far away from the training data the variance grows and levels out at the signal variance (non-degeneracy of the GP). Therefore, a probabilistic GP model can still be preferable to a deterministic model even if the underlying function itself is deterministic. When the transition function in equation (3.1) is described by a GP, the GP dynamics model is a distribution over all transition dynamics that plausibly could have generated the training set.[5] In the context of biological learning, the transition function f itself can be considered a forward model humans use for predictions (Miall and Wolpert, 1996).

3.4 Intermediate Layer: Approximate Inference for Long-Term Predictions

Even for a deterministically given pair $(\mathbf{x}_*, \mathbf{u}_*)$, the GP returns a Gaussian predictive distribution $p(f(\mathbf{x}_*, \mathbf{u}_*))$ (see equation (3.6)). Thus, when we simulate the probabilistic GP model forward in time, the predicted states are uncertain. To evaluate V^π in equation (3.2), we therefore require the predictive state distributions $p(\mathbf{x}_1), \ldots, p(\mathbf{x}_T)$, where the GP model on the one-step transition dynamics f yields the transition probability $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$. We cascade short-term predictions to obtain these long-term predictions at the intermediate layer in Figure 3.3. During the forward simulation, it is essential for coherent predictions to keep track of uncertainty evolution over time. To compute a successor state distribution when the current state-action pair is given by a probability distribution, we adopt the results from Section 2.3.2 and approximate the true predictive distribution by a Gaussian with the exact mean and the exact covariance matrix (exact moment matching).

Figure 3.6(a) illustrates how to cascade short-term predictions without control signals: Without any control signal, the distribution $p(\mathbf{x}_t)$ can be computed using the results from Section 2.3.2. The shaded node denotes a moment-matching approximation, such that the state distribution $p(\mathbf{x}_t)$ is approximately Gaussian. Figure 3.6(b) extends the model from Figure 3.6(a) by adding the policy as a function of the state. First, a distribution $p(\pi(\mathbf{x}_{t-1}))$ over actions is computed when mapping $p(\mathbf{x}_{t-1})$ through the

[5] The GP can naturally treat system noise and/or measurement noise. We do not go into further details as they are not required at this point.



(a) Cascading predictions without a policy.

(b) Cascading predictions with a policy.

Figure 3.6: Cascading predictions without and with a policy.

policy. Second, we compute a joint Gaussian distribution $p(\mathbf{x}_{t-1}, \pi(\mathbf{x}_{t-1}))$ (shaded node). Third, the distribution $p(\mathbf{x}_t)$ is computed using the results from Section 2.3.2.[6]

Throughout all computations, we explicitly take the model uncertainty into account by averaging over all plausible dynamics models captured by the GP. To predict a successor state, we average over both the uncertainty in the current state *and* the uncertainty about the dynamics model itself. Thus, we reduce model bias, which is one of the strongest arguments against model-based learning algorithms. See the work by Atkeson and Santamaría (1997), Atkeson and Schaal (1997), and Sutton and Barto (1998) for examples and further details.

Probabilistic models for the dynamics and approximate inference techniques for predictions allow us to keep track of the uncertainties in the internal simulations. Typically, in the early stages of learning, the predictive uncertainty in the states grows rapidly with increasing prediction horizon. With increasing experience and a good policy[7], however, we expect the predictive uncertainty to collapse because the system is being controlled. In the context of human learning, Bays and Wolpert (2007) provide evidence that the brain attempts to decide on controls that reduce uncertainty in an optimal control setup (which RL often mimics).

3.4.1 Policy Requisites

For the internal simulation, the policy employed has to fulfill two properties: First, for a state distribution $p(\mathbf{x})$ we need to be able compute a corresponding distribution over actions $p(\mathbf{u}) = p(\pi(\mathbf{x}))$. Second, in a realistic application, the policy must be able to deal with constrained control signals.

Predictive Distribution over Actions

For a single deterministic state, the policy deterministically returns a single action. However, during the forward simulation (Figure 3.6), the states are given by a probability distribution $p(\mathbf{x}_t)$, $t = 0, \ldots, T$. The probability distribution of the state $\mathbf{x}_t$ induces a predictive distribution over actions, even if the policy is deterministic. To give an intuitive example, let us for a moment represent a state distribution by a set of "micro-states". For each "micro-state", the policy deterministically returns a single "micro-action". The collection of (non-identical) micro-actions represent a distribution over actions. Figure 3.7 illustrates this simplified relationship.

More formally, we require a function representation $\tilde{\pi}$ that allows for the computation of a distribution over actions $p(\tilde{\pi}(\mathbf{x}_t))$, where $\mathbf{x}_t$ is a Gaussian distributed state vector. The function $\tilde{\pi}$ is called a *preliminary policy*. We compute exactly the mean and the variance of $p(\tilde{\pi}(\mathbf{x}))$ and approximate $p(\tilde{\pi}(\mathbf{x}))$ by a Gaussian with these moments (exact moment matching).

Constrained Control Signals

In most practical applications, it is often only possible to apply control signals with a bounded amplitude, that is, force or torque constraints have to be accounted for. Let us consider the preliminary policy $\tilde{\pi}$ with

[6] The joint distribution over states and actions is required since the GP training inputs are state-action pairs, which lead to a successor state.

[7] With a "good" policy we mean a policy that works well when being applied to the real system.

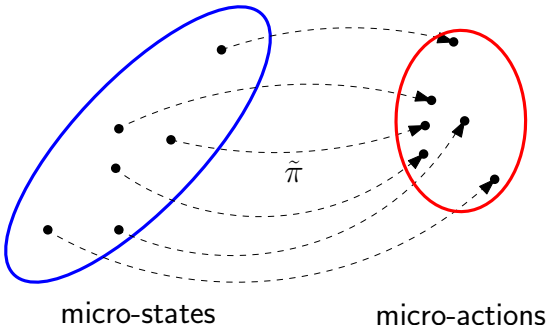


Figure 3.7: "Micro-states" being mapped through the preliminary policy $\tilde{\pi}$ to a set of "micro-actions". The deterministic preliminary policy $\tilde{\pi}$ maps any micro-state in the state distribution (blue ellipse) to possibly different micro-actions resulting in a distribution over actions (red ellipse).

an unconstrained amplitude. Suppose the maximum amplitude of the applied control signal is given by $2\,\mathbf{u}_{\max}$, that is, $\mathbf{u} \in [-\mathbf{u}_{\max}, \mathbf{u}_{\max}]$. To model the constrained control signal coherently during simulation, we squash the preliminary policy $\tilde{\pi}$ through a bounded function that limits the amplitude of the final policy π. More specifically, we map the preliminary policy through the sine function and multiply it by $\mathbf{u}_{\max}$. This means, the final policy π is given by

$$\pi(\mathbf{x}) = \mathbf{u}_{\max} \sin(\tilde{\pi}(\mathbf{x})) \in [-\mathbf{u}_{\max}, \mathbf{u}_{\max}]\,. \tag{3.7}$$

Instead of the sine, a saturating sigmoid function such as the logistic and the cumulative Gaussian could have been employed as the squashing function. The sine function has the nice property that it actually attains its extreme values ± 1 for finite values of $\tilde{\pi}(\mathbf{x})$, namely $\tilde{\pi}(\mathbf{x}) = \frac{\pi}{2} + k\,\pi,\ k \in \mathbb{Z}$. Therefore, it is sufficient for the preliminary policy $\tilde{\pi}$ to describe a function with function values in the range of $\pm\pi$. By contrast, if we mapped $\tilde{\pi}(\mathbf{x})$ through a sigmoid function that attains ± 1 in the limit for $\tilde{\pi}(\mathbf{x}) \to \pm\infty$, the function values of $\tilde{\pi}$ needed to be extreme in order to apply control signals of $\pm\mathbf{u}_{\max}$, which can lead to numerical instabilities. Another advantageous property of the sine is that it allows for an analytic computation of the mean and the covariance of $p(\tilde{\pi}(\mathbf{x}))$ if $\tilde{\pi}(\mathbf{x})$ is Gaussian distributed. Details are given in Appendix A.1.[8]

To summarize, we squash the Gaussian distribution $p(\tilde{\pi}(\mathbf{x}))$ through the sine according to equation (3.7), which allows for an analytical computation of the mean and the variance of the distribution over actions

$$p(\pi(\mathbf{x})) = p(\mathbf{u}) = p(\mathbf{u}_{\max} \sin(\tilde{\pi}(\mathbf{x}))) \tag{3.8}$$

using the results from Appendix A.1.

3.4.2 Representations of a Preliminary Policy

In the following, we discuss two possible representations of the preliminary policy $\tilde{\pi}$ that allow for a closed-form computation of the distribution $p(\tilde{\pi}(\mathbf{x}))$ when the state $\mathbf{x}$ is Gaussian distributed. In this dissertation, we consider a linear representation and a nonlinear representation of $\tilde{\pi}$, where the latter one is given by a radial basis function (RBF) network.

Linear Model

The linear preliminary policy is given by

$$\tilde{\pi}(\mathbf{x}_*) = \boldsymbol{\Psi}\mathbf{x}_* + \boldsymbol{\nu}\,, \tag{3.9}$$

where $\boldsymbol{\Psi}$ is a parameter matrix of weights and $\boldsymbol{\nu}$ is an offset vector. In each control dimension d, the policy (3.9) is a linear combination of the states (the weights are given by the dth row in $\boldsymbol{\Psi}$) plus an offset ν_d.

[8] Note that the cumulative Gaussian also allows for the computation of the mean and the covariance of the predictive distribution for a Gaussian distributed input $\tilde{\pi}(\mathbf{x})$. For details, we refer to the book by Rasmussen and Williams (2006, Chapter 3.9).

Predictive Distribution. The predictive distribution $p(\tilde{\pi}(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ for a state distribution $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is an exact Gaussian with mean and covariance

$$\mathbb{E}_{\mathbf{x}_*}[\tilde{\pi}(\mathbf{x}_*)] = \boldsymbol{\Psi}\boldsymbol{\mu} + \boldsymbol{\nu}\,,$$
$$\mathrm{cov}_{\mathbf{x}_*}[\tilde{\pi}(\mathbf{x}_*)] = \boldsymbol{\Psi}\boldsymbol{\Sigma}\boldsymbol{\Psi}^\top\,,$$

respectively. A drawback of a linear policy in equation (3.9) is that it is not very flexible. However, a linear controller can often be used to stabilize a system around an equilibrium point.

Nonlinear Model: RBF Network

In the nonlinear case, we represent the preliminary policy $\tilde{\pi}$ by a radial basis function network with Gaussian basis functions. The preliminary RBF policy is given by

$$\tilde{\pi}(\mathbf{x}_*) = \sum_{s=1}^{N} \beta_s k_\pi(\mathbf{x}_s, \mathbf{x}_*) = \boldsymbol{\beta}_\pi^\top k_\pi(\mathbf{X}_\pi, \mathbf{x}_*)\,, \tag{3.10}$$

where $\mathbf{x}_*$ is a test input, k_π is the squared exponential kernel (unnormalized Gaussian basis function) in equation (2.2) plus a noise kernel $\delta_{\mathbf{x},\mathbf{x}'}\sigma_\pi^2$, and $\boldsymbol{\beta}_\pi \coloneqq (\mathbf{K}_\pi + \sigma_\pi^2\mathbf{I})^{-1}\mathbf{y}_\pi$ is a weight vector.[9] The entries of $\mathbf{K}_\pi$ are given by $(K_\pi)_{ij} = k_\pi(\mathbf{x}_i, \mathbf{x}_j)$, the vector $\mathbf{y}_\pi \coloneqq \tilde{\pi}(\mathbf{X}_\pi) + \boldsymbol{\varepsilon}_\pi$, $\boldsymbol{\varepsilon}_\pi \sim \mathcal{N}(\mathbf{0}, \sigma_\pi^2\mathbf{I})$ collects the training targets, where $\boldsymbol{\varepsilon}_\pi$ is measurement noise. The set $\mathbf{X}_\pi = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$, $\mathbf{x}_s \in \mathbb{R}^D$, $s = 1, \ldots, N$, are the training inputs (locations of the means/centers of the Gaussian basis functions), also called the *support points*. The RBF network in equation (3.10) allows for flexible modeling, which is useful if the structure of the underlying function (in our case a good policy) is unknown.

The parameterization of the RBF network in equation (3.10) is rather unusual, but as expressive as the "standard" parameterization where the $\boldsymbol{\beta}$ is simply a set of parameters and not defined as $(\mathbf{K}_\pi+\sigma_\pi^2\mathbf{I})^{-1}\mathbf{y}_\pi$. See Chapter 4 for a more detailed discussion.

Predictive Distribution. The RBF network in equation (3.10) allows for a closed-form computation of a predictive distribution $p(\tilde{\pi}(\mathbf{x}_*))$:

- The predictive mean of $p(\tilde{\pi}(\mathbf{x}_*))$ for a known state $\mathbf{x}_*$ is equivalent to RBF policy in equation (3.10), which itself is identical to the predictive mean of a GP in equation (2.11). In contrast to the GP model, both the predictive variance and the uncertainty about the underlying function in an RBF network are zero. Thus, the predictive distribution $p(\tilde{\pi}(\mathbf{x}_*))$ for a given state $\mathbf{x}_*$ has zero variance.
- For a Gaussian distributed state $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ the predictive mean and the predictive covariance can be computed similarly to Section 2.3.2 when we consider the RBF network a "deterministic GP" with the restriction that $\mathrm{var}_{\tilde{\pi}}[\tilde{\pi}(\mathbf{x}_*)] = 0$ for all $\mathbf{x}_*$. In particular, the predictive mean is given by

$$\mathbb{E}_{\mathbf{x}_*,\tilde{\pi}}[\tilde{\pi}(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{\mathbf{x}_*}[\underbrace{\mathbb{E}_{\tilde{\pi}}[\tilde{\pi}(\mathbf{x}_*)|\mathbf{x}_*]}_{=m_{\tilde{\pi}}(\mathbf{x}_*)=\tilde{\pi}(\mathbf{x}_*)}|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \boldsymbol{\beta}_\pi^\top\mathbf{q}\,, \tag{3.11}$$

  where $\mathbf{q}$ is defined in equation (2.17). The predictive variance of $p(\tilde{\pi}(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is

$$\begin{aligned}\mathrm{var}[\tilde{\pi}(\mathbf{x}_*)] &= \mathbb{E}_{\mathbf{x}_*}[\underbrace{\mathrm{var}_{\tilde{\pi}}[\tilde{\pi}(\mathbf{x}_*)|\mathbf{x}_*]}_{=0}|\boldsymbol{\mu}, \boldsymbol{\Sigma}] + \mathrm{var}_{\mathbf{x}_*}[\underbrace{\mathbb{E}_{\tilde{\pi}}[\tilde{\pi}(\mathbf{x}_*)|\mathbf{x}_*]}_{=\tilde{\pi}(\mathbf{x}_*)}|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \\ &= \mathbb{E}_{\mathbf{x}_*}[\tilde{\pi}(\mathbf{x}_*)^2|\boldsymbol{\mu}, \boldsymbol{\Sigma}] - \mathbb{E}_{\mathbf{x}_*}[\tilde{\pi}(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}]^2 = \boldsymbol{\beta}_\pi^\top\mathbf{Q}\boldsymbol{\beta}_\pi - (\boldsymbol{\beta}_\pi^\top\mathbf{q})^2\,,\end{aligned} \tag{3.12}$$

  where the matrix $\mathbf{Q}$ is defined in equation (2.25). Note that the predictive variance in equation (3.12) equals the cross-covariance entries in the covariance matrix for a multivariate GP prediction, equation (2.26).

We approximate the predictive distribution $p(\tilde{\pi}(\mathbf{x}_*))$ by a Gaussian with the exact mean and the exact variance (moment matching). Similar to Section 2.3.2, these results can easily be extended to multivariate policies.

[9] The RBF network in equation (3.10) has the same representation as the mean function of a GP. The RBF network can be considered a deterministic GP with a fixed number of N basis function. Here, "deterministic" means that there is no uncertainty about the underlying function, that is, $\mathrm{var}_{\tilde{\pi}}[\tilde{\pi}(\mathbf{x})] = 0$. Note that the RBF network is a degenerate model; the predicted variances far away from the centers of the basis functions decline to zero.

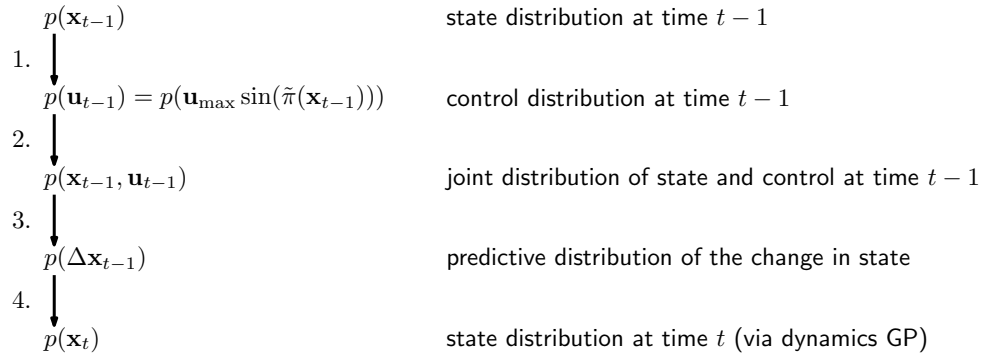


Figure 3.8: Computational steps required to determine $p(\mathbf{x}_t)$ from $p(\mathbf{x}_{t-1})$ and a policy $\pi(\mathbf{x}_{t-1}) = \mathbf{u}_{\max}\sin(\tilde{\pi}(\mathbf{x}_{t-1}))$.

3.4.3 Distribution of the Successor State

Figure 3.8 recaps and summarizes the computational steps required to compute the distribution of the successor state $p(\mathbf{x}_t)$ from $p(\mathbf{x}_{t-1})$:

1. The computation of a distribution over actions $p(\mathbf{u}_{t-1})$ from the state distribution $p(\mathbf{x}_{t-1})$ requires two steps:

   (a) For a Gaussian distribution $p(\mathbf{x}_{t-1})$ of the state at time $t-1$ a Gaussian approximation of the distribution $p(\tilde{\pi}(\mathbf{x}_{t-1}))$ of the preliminary policy is computed (exact moment matching).

   (b) The preliminary policy is squashed through the sine and an approximate Gaussian distribution of $p(\mathbf{u}_{\max}\sin(\tilde{\pi}(\mathbf{x}_{t-1})))$ is computed in equation (3.8) using the results from Appendix A.1 (exact moment matching).

2. We compute the joint distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = p(\mathbf{x}_{t-1}, \pi(\mathbf{x}_{t-1}))$ in two steps:

   (a) The distribution $p(\mathbf{x}_{t-1}, \tilde{\pi}(\mathbf{x}_{t-1}))$ of the state and the unsquashed control signal is computed. If $\tilde{\pi}$ is the linear model in equation (3.9), this computation is straightforward. If $\tilde{\pi}$ is the RBF network in equation (3.10), a Gaussian approximation to the joint distribution can be computed using the results from Section 2.3.3 (exact moment matching).

   (b) Using the results in Appendix A.1, we compute an approximate fully joint Gaussian distribution $p(\mathbf{x}_{t-1}, \tilde{\pi}(\mathbf{x}_{t-1}), \mathbf{u}_{\max}\sin(\tilde{\pi}(\mathbf{x}_{t-1})))$ and marginalize $\tilde{\pi}(\mathbf{x}_{t-1})$ out to obtain the desired joint distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$. We obtain obtain cross-covariance information between $\mathbf{x}_{t-1}$ and $\mathbf{u}_{\max}\sin(\tilde{\pi}(\mathbf{x}_{t-1})) = \mathbf{u}_{t-1}$ via

   $$\operatorname{cov}[\mathbf{x}_{t-1}, \mathbf{u}_{t-1}] \approx \operatorname{cov}[\mathbf{x}_{t-1}, \tilde{\pi}(\mathbf{x}_{t-1})]\operatorname{cov}[\tilde{\pi}(\mathbf{x}_{t-1}), \tilde{\pi}(\mathbf{x}_{t-1})]^{-1}\operatorname{cov}[\tilde{\pi}(\mathbf{x}_{t-1}), \mathbf{u}_{t-1}]\,,$$

   which generally leads to an approximate Gaussian joint distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = p(\mathbf{x}_{t-1}, \pi(\mathbf{x}_{t-1}))$ that does not match the moments of the corresponding true distribution.

3. With the Gaussian input distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$, the distribution $p(\Delta\mathbf{x}_{t-1})$ of the change in state can be computed by using the results from Section 2.3.2. Note that the inputs to the dynamics GP are state-action pairs and the targets are the differences $\Delta\mathbf{x}_{t-1} = f(\mathbf{x}_t, \mathbf{u}_t) - \mathbf{x}_t$, see equation (3.3).

4. A Gaussian approximation of the successor state distribution $p(\mathbf{x}_t)$ is given by the mean and the covariance

   $$\begin{aligned}\mathbb{E}_{\mathbf{x}_{t-1},f}[f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})] &= \boldsymbol{\mu}_{t-1} + \mathbb{E}_{\mathbf{x}_{t-1},f}[\Delta\mathbf{x}_{t-1}]\,,\\ \operatorname{cov}_{\mathbf{x}_{t-1},f}[f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})] &= \boldsymbol{\Sigma}_{t-1} + \operatorname{cov}_{\mathbf{x}_{t-1},f}[\mathbf{x}_{t-1}, \Delta\mathbf{x}_{t-1}] + \operatorname{cov}_{\mathbf{x}_{t-1},f}[\Delta\mathbf{x}_{t-1}, \mathbf{x}_{t-1}]\\ &\quad + \operatorname{cov}_{\mathbf{x}_{t-1},f}[\Delta\mathbf{x}_{t-1}]\,,\end{aligned}$$

   respectively.

3.4.4 Policy Evaluation

The predictive state distributions $p(\mathbf{x}_t)$, $t = 1, \ldots, T$, are computed iteratively and are necessary in the approximate inference step (intermediate layer in Figure 3.3) to evaluate the value function V^π. Since the value function is

$$V^\pi(\mathbf{x}_0) = \sum_{t=0}^{T} \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]$$

and the distributions $p(\mathbf{x}_t)$ are computed according to the scheme in Figure 3.8, the remaining problem reduces to compute

$$\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = \int c(\mathbf{x}_t) \underbrace{p(\mathbf{x}_t)}_{\text{Gaussian}} \, \mathrm{d}\mathbf{x}_t \,,$$

which corresponds to convolving the cost function c with the approximate Gaussian state distribution $p(\mathbf{x}_t)$. Depending on the representation of the immediate cost function c, this integral can be solved analytically. If the cost function c is unknown (not discussed in this report) and the values $c(\mathbf{x})$ are only observed, a GP can be employed to represent the cost function $c(\mathbf{x})$. This immediate-cost GP would also allow for the analytic computation of $\mathbb{E}_{\mathbf{x},c}[c(\mathbf{x})]$.

3.5 Top Layer: Optimization of the Policy Parameters

The optimization problem at the top layer in Figure 3.3 corresponds to finding policy parameters $\boldsymbol{\psi}^*$ that minimize the expected long-term cost in equation (3.2). Equation (3.2) can be extended to N_p paths $\boldsymbol{\tau}_i$ starting from different initial state distributions $p(\mathbf{x}_0^{(i)})$, for instance by considering the sample average

$$\frac{1}{N_p} \sum_{i=1}^{N_p} V^{\pi_\psi}(\mathbf{x}_0^{(i)}) \,,$$

where $V^{\pi_\psi}(\mathbf{x}_0^{(i)})$ is determined according to equation (3.2). Alternative approaches using an explicit value function model are also plausible. In the following, however, we restrict ourselves to a single initial state, but the extension to multiple initial states is straightforward.

We employ a *gradient-based policy search* method. This means, we aim to find a parameterized policy π^* from a class of policies Π with

$$\pi^* \in \arg\min_{\pi \in \Pi} V^{\pi_\psi}(\mathbf{x}_0) = \pi_{\psi^*} \in \arg\min_{\boldsymbol{\psi}} V^{\pi_\psi} \,.$$

In our case, the policy class Π defines a constrained policy space and is given either by the class of linear functions or by the class of functions that are represented by an RBF with N Gaussian basis functions (squashed through the sine). Restricting the policy search to the class Π generally leads to suboptimal policies. However, depending on the expressiveness of Π, the policy found causes a similar expected long-term cost V^π as a globally optimal policy. In the following, we do not distinguish between a globally optimal policy π^* and $\pi^* \in \Pi$.

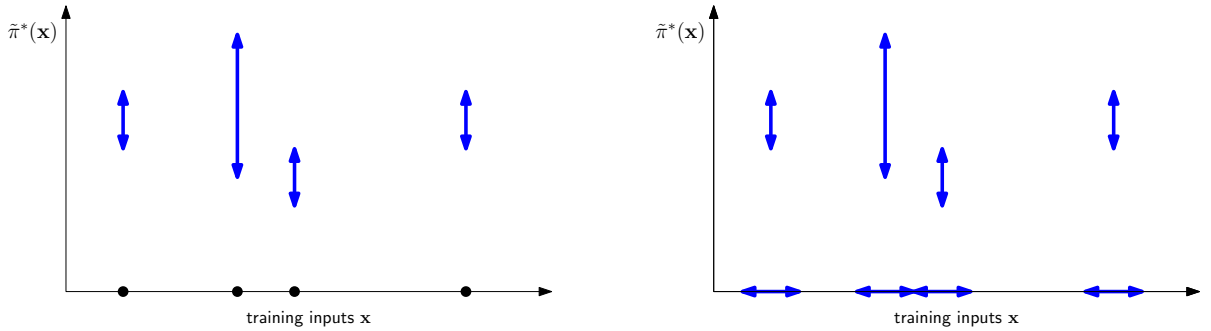
To learn the policy, we employ the deterministic conjugate gradients minimizer described by Rasmussen (1996), which requires the gradient of the value function V^{π_ψ} with respect to the policy parameters $\boldsymbol{\psi}$.[10] Since approximate inference for policy evaluation can be done in closed form (Section 3.4), these derivatives can be computed analytically by repeated application of the chainrule.

3.5.1 Policy Parameters

In the following, we describe the policy parameters for both the linear and the RBF policy[11] and provide some details about the computation of the required partial derivatives for both the linear policy in equation (3.9) and the RBF policy in equation (3.10).

[10] The minimizer is contained in the *gpml* software package, which is publicly available at `http://www.gaussianprocess.org/gpml/`.

[11] For notational convenience, with a linear/RBF policy we mean the linear/RBF preliminary policy $\tilde{\pi}$ squashed through the sine and subsequently multiplied by $\mathbf{u}_{\max}$.



(a) The training targets are treated as parameters.

(b) The training targets and the corresponding support points define the parameter set to be optimized.

Figure 3.9: Parameters of a function approximator for the preliminary policy $\tilde{\pi}^*$. The x-axes show the support points in the state space, the y-axes show the corresponding function values of an optimal preliminary policy $\tilde{\pi}^*$. For given support points, the corresponding function values of an optimal policy are uncertain as illustrated in panel (a). Panel (b) shows the situation where both the support points and the corresponding function values are treated as parameters and jointly optimized.

Linear Policy

The linear policy model

$$\pi(\mathbf{x}_*) = \mathbf{u}_{\max} \sin(\tilde{\pi}(\mathbf{x}_*)), \quad \tilde{\pi}(\mathbf{x}_*) = \boldsymbol{\Psi}\mathbf{x}_* + \boldsymbol{\nu}, \quad \mathbf{x}_* \in \mathbb{R}^D,$$

see equation (3.9), possesses $D+1$ parameters per control dimension: For policy dimension d there are D weights in the dth row of the matrix $\boldsymbol{\Psi}$. One additional parameter originates from the offset parameter ν_d.

RBF Policy

In short, the parameters of the nonlinear RBF policy are the locations $\mathbf{X}_\pi$ of the centers, the training targets $\mathbf{y}_\pi$, the hyper-parameters of the Gaussian basis functions, and the variance of the measurement noise. The RBF policy is represented as

$$\pi(\mathbf{x}_*) = \mathbf{u}_{\max} \sin(\tilde{\pi}(\mathbf{x}_*)), \quad \tilde{\pi}(\mathbf{x}_*) = \boldsymbol{\beta}_\pi^\top k_\pi(\mathbf{X}_\pi, \mathbf{x}_*), \quad \mathbf{x}_* \in \mathbb{R}^D,$$

see equation (3.10).

Let us motivate these parameters: Let $\mathbf{X}_\pi$ be the set of N support points $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$, that is, the locations of the means of the Gaussian basis functions. If the corresponding function values $\mathbf{y}_\pi = \tilde{\pi}(\mathbf{X}_\pi) + \boldsymbol{\varepsilon}_\pi$, $\boldsymbol{\varepsilon}_\pi \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\pi)$, were known, a function approximator such as interpolating polynomials or, as in our case, an RBF network could be fitted. However, the function values that lead to an optimal policy are unknown. Fortunately, we can characterize an optimal policy: An optimal policy π^* minimizes the expected long-term cost V^π in equation (3.2). For given support points $\mathbf{X}_\pi$, we can simply treat the corresponding function values $\tilde{\pi}(\mathbf{x}_s)$, $s = 1, \ldots, N$, as parameters to be optimized. This situation is illustrated in Figure 3.9(a). Note that the support points $\mathbf{X}_\pi = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$ of the policy are unknown as well. There are broadly two options to deal with this situation: One option is to manually set the support points to locations that "look good". This approach requires prior knowledge about the latent optimal policy π^*. Alternatively, an automatic procedure of selecting the support points can be employed. In this case, it is possible to place the support points according to a performance criterion, such as mutual information or space coverage, which often corresponds to maximum information gain as detailed by Chaloner and Verdinelli (1995), Verdinelli and Kadane (1992), and MacKay (1992). In the context of an optimal control problem, space-filling designs and well-separated support points do not necessarily lead to a good policy in a region of interest, that is, along a good trajectory. Instead, we use the expected long-term cost in equation (3.2) directly as the performance criterion according to which the locations of the support points are optimized. Figure 3.9(b) illustrates the *joint optimization* of support points and the corresponding training targets.

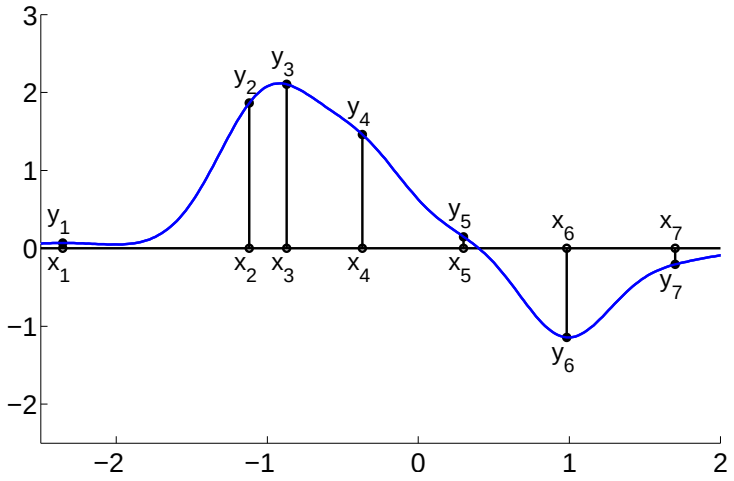


Figure 3.10: Preliminary policy $\tilde{\pi}$ (blue) implemented by an RBF network using a pseudo-training set. The values x_i and y_i are the pseudo-inputs and pseudo-targets, respectively. The blue function does not pass through the pseudo-training targets since they are noisy.

The support points $\mathbf{X}_\pi$ and the corresponding training targets $\mathbf{y}_\pi = \tilde{\pi}(\mathbf{X}_\pi) + \boldsymbol{\varepsilon}_\pi$ are also referred to as a *pseudo-training set* or a *fictitious training set* for the preliminary policy.[12] By modifying the pseudo-training set, we can control the implemented policy. Figure 3.10 shows an example of a function $\tilde{\pi}$ implemented by an RBF network using a pseudo-training set $\{x_1, \ldots, x_7, y_1, \ldots, y_7\}$.

Besides the pseudo-training set, the hyper-parameters are an additional set of policy parameters to be optimized. The hyper-parameters are the length-scales (widths) of the axis-aligned Gaussian basis functions, the (measurement) noise variance σ_π^2, and the variance of the implemented function itself[13].

In the most general case, where the entire pseudo-training set and the hyper-parameters are considered parameters to be optimized, the RBF policy in equation (3.10) for a scalar control law contains ND parameters for the pseudo-inputs, N parameters for the pseudo-targets, and $D + 2$ hyper-parameters. Here, N is the number of basis functions of the RBF network, and D is the dimensionality of the pseudo-inputs. Generally, if the policy implements an F-dimensional control signal, we need to optimize $N(D + F) + (D + 2)F$ parameters. As an example, for $N = 100$, $D = 10$, and $F = 2$, this leads to a 1,224-dimensional optimization problem. A gradient-based optimization method using *estimates* of the gradient of V^π such as finite differences or more efficient sampling-based methods (see the work by Peters and Schaal (2008b) for an overview) require many function evaluations, which can be computationally expensive. However, since in our case the policy evaluation can be performed analytically, we profit from closed-form expressions for the gradients.

3.5.2 Gradient of the Value Function

The gradient of the expected long-term cost V^π along a path $\boldsymbol{\tau}$ with respect to the policy parameters is given by

$$\frac{\mathrm{d}V^{\pi_\psi}(\mathbf{x}_0)}{\mathrm{d}\boldsymbol{\psi}} = \sum_{t=0}^{T} \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\psi}} \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)|\pi_\psi]\,, \tag{3.13}$$

where the subscript $\boldsymbol{\psi}$ emphasizes that π depends on the parameter set $\boldsymbol{\psi}$. Moreover, we conditioned explicitly on π_ψ in the expected value to emphasize the dependence of $\mathbb{E}_{\mathbf{x}}[c(\mathbf{x})]$ on the policy parameters. The total derivative with respect to the policy parameters is denoted by $\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\psi}}$. The expected immediate cost $\mathbb{E}_{\mathbf{x}}[c(\mathbf{x}_t)]$ solely depends on the state distribution $p(\mathbf{x}_t)$. Note, however, that the moments of $p(\mathbf{x}_t)$, which are essentially given by the equations (2.16), (2.25), and (2.26), are functionally dependent on both the policy parameter vector $\boldsymbol{\psi}$ and the moments $\boldsymbol{\mu}_{t-1}$ and $\boldsymbol{\Sigma}_{t-1}$ of the state distribution $p(\mathbf{x}_{t-1})$ at time

[12] The concept of the pseudo-training set is closely related to the ideas of inducing inputs used in the sparse GP approximation by Snelson and Ghahramani (2006) and the Gaussian process latent variable model by Lawrence (2005).

[13] The variance of the function is related to the amplitude of $\tilde{\pi}$.

$t-1$. The total derivative in equation (3.13) is therefore given by

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\psi}}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = \left(\frac{\partial}{\partial\boldsymbol{\mu}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]\right)\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\psi}} + \left(\frac{\partial}{\partial\boldsymbol{\Sigma}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]\right)\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\boldsymbol{\psi}}$$

since $p(\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. The partial derivative with respect to $\boldsymbol{\mu}_t$ is denoted by $\frac{\partial}{\partial\boldsymbol{\mu}_t}$. We recursively compute the required derivatives in the following three steps top-down layers:

1. First, we analytically determine the derivatives

$$\frac{\partial}{\partial\boldsymbol{\mu}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]\,, \quad \frac{\partial}{\partial\boldsymbol{\Sigma}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]\,, \tag{3.14}$$

   where $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ are the mean and the covariance of the state distribution $p(\mathbf{x}_t)$, respectively. The expressions in equation (3.14) depend on the representation of the cost function c. Section 3.6.1 presents the corresponding derivatives for one particular cost function representation.

2. The derivatives in the second step are then

$$\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\psi}} = \frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\mu}_{t-1}}\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\Sigma}_{t-1}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\psi}}\,, \tag{3.15}$$

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\boldsymbol{\psi}} = \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\mu}_{t-1}}\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\Sigma}_{t-1}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\psi}}\,, \tag{3.16}$$

   for which we can obtain analytic expressions. Note that the partial derivatives $\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\psi}}$ and $\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\psi}}$ have been computed in the previous recursion.[14]

3. In the third step, we compute the derivatives

$$\frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\psi}}\,, \quad \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\psi}}\,. \tag{3.17}$$

   Due to the sequence of computations to compute the distribution of a consecutive state (see Figure 3.8), the partial derivatives in equation (3.17) require repeated application of the chainrule. For $\pi(\mathbf{x}_{t-1}) = \mathbf{u}_{\max}\sin(\tilde{\pi}(\mathbf{x}_{t-1}, \boldsymbol{\psi}))$, we obtain

$$\begin{aligned}\frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\psi}} &= \frac{\partial\mathbb{E}_{\mathbf{x}_{t-1},\mathbf{u}_{t-1},f}[\Delta\mathbf{x}_{t-1}]}{\partial\boldsymbol{\psi}} = \frac{\partial\mathbb{E}_{\mathbf{x}_{t-1},\mathbf{u}_{t-1},f}[\Delta\mathbf{x}_{t-1}]}{\partial p(\pi(\mathbf{x}_{t-1}))}\frac{\partial p(\mathbf{u}_{\max}\sin(\tilde{\pi}(\,\cdot\,)))}{\partial\boldsymbol{\psi}} \\ &= \frac{\partial\mathbb{E}_{\mathbf{x}_{t-1},\mathbf{u}_{t-1},f}[\Delta\mathbf{x}_{t-1}]}{\partial p(\pi(\mathbf{x}_{t-1}))}\frac{\partial p(\mathbf{u}_{\max}\sin(\tilde{\pi}(\,\cdot\,)))}{\partial p(\tilde{\pi}(\,\cdot\,))}\frac{\partial p(\tilde{\pi}(\mathbf{x}_{t-1}, \boldsymbol{\psi}))}{\partial\boldsymbol{\psi}}\,,\end{aligned}$$

   for the first partial derivative in equation (3.17). Since all involved probability distributions are either exact Gaussian or approximate Gaussian, we informally write

$$\frac{\partial f(\mathbf{a})}{\partial p(\mathbf{a})}\frac{\partial p(\mathbf{a})}{\partial\boldsymbol{\psi}} = \frac{\partial f(\mathbf{a})}{\partial\mathbb{E}[\mathbf{a}]}\frac{\partial\mathbb{E}[\mathbf{a}]}{\partial\boldsymbol{\psi}} + \frac{\partial f(\mathbf{a})}{\partial\mathrm{cov}[\mathbf{a}]}\frac{\partial\mathrm{cov}[\mathbf{a}]}{\partial\boldsymbol{\psi}}$$

   to abbreviate the expressions. The second partial derivative in equation (3.17) can be obtained following the same scheme.

Example. We present two partial derivatives from equation (3.15) that are independent of the policy model. These derivatives can be derived from equation (2.16) and equation (2.17).

The derivative of the ath dimension of the predictive mean $\boldsymbol{\mu}_t$ with respect to the mean $\boldsymbol{\mu}_{t-1}$ of the input distribution[15] is given by

$$\frac{\partial\mu_t^{(a)}}{\partial\boldsymbol{\mu}_{t-1}} = \boldsymbol{\beta}_a^\top\left(\frac{\partial\mathbf{q}_a}{\partial\boldsymbol{\mu}_{t-1}}\right) = \frac{\alpha_a^2}{\sqrt{|\mathbf{R}\boldsymbol{\Lambda}_a^{-1}|}}\underbrace{(\boldsymbol{\beta}_a \odot \mathbf{q}_a)^\top}_{1\times n}\underbrace{\mathbf{T}^\top}_{n\times(D+F)}\,,$$

[14] To compute necessary multi-dimensional matrix multiplications, we used Jason Farquhar's tprod-toolbox for Matlab, which can be found at `http://www.mathworks.com/matlabcentral/fileexchange/16275`.

[15] Note that the input distribution is the joint distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ with $\mathbf{x}_{t-1} \in \mathbb{R}^D$ and $\mathbf{u}_{t-1} \in \mathbb{R}^F$.

where n is the size of the training set for the dynamics model, $\odot$ is a point-wise matrix product, $\mathbf{q}_a$ is defined in equation (2.17), and

$$\mathbf{R} \coloneqq \boldsymbol{\Sigma}_{t-1} + \boldsymbol{\Lambda}_a \in \mathbb{R}^{(D+F)\times(D+F)}\,,$$
$$\mathbf{T} \coloneqq (\mathbf{X} - [\mathbb{1}_{1\times n} \otimes \boldsymbol{\mu}_{t-1}])\mathbf{R}^{-1} \in \mathbb{R}^{(D+F)\times n}\,.$$

Here, $\mathbf{X}$ are the training inputs for the dynamics GP and $\mathbb{1}_{1\times n}$ is a $1 \times n$ matrix of ones. The operator $\otimes$ is a Kronecker product.

The derivative of the ath dimension of the predictive mean $\boldsymbol{\mu}_t$ with respect to the input covariance matrix $\boldsymbol{\Sigma}_{t-1}$ is given by

$$\frac{\partial \mu_t^{(a)}}{\boldsymbol{\Sigma}_{t-1}} = \frac{\alpha_a^2}{2\sqrt{|\mathbf{R}\boldsymbol{\Lambda}_a^{-1}|}} \mathbf{T} \underbrace{\left(\mathbf{T}^\top \odot [\mathbb{1}_{1\times(D+F)} \otimes (\boldsymbol{\beta}_a \odot \mathbf{q}_a)]\right)}_{n\times(D+F)} - \frac{\mu_t^{(a)}}{2}\mathbf{R}^{-1}\,.$$

The remaining derivatives

$$\underbrace{\frac{\partial \mu_t^{(a)}}{\partial \boldsymbol{\theta}}, \frac{\partial \mu_t^{(a)}}{\partial \mathbf{X}_\pi}, \frac{\partial \mu_t^{(a)}}{\partial \mathbf{y}_\pi}}_{= \frac{\partial \mu_t^{(a)}}{\partial \boldsymbol{\psi}}}, \frac{\partial \Sigma_t^{(a,b)}}{\partial \boldsymbol{\mu}_{t-1}}, \frac{\partial \Sigma_t^{(a,b)}}{\partial \boldsymbol{\Sigma}_{t-1}}, \underbrace{\frac{\partial \Sigma_t^{(a,b)}}{\partial \boldsymbol{\theta}}, \frac{\partial \Sigma_t^{(a,b)}}{\partial \mathbf{X}_\pi}, \frac{\partial \Sigma_t^{(a,b)}}{\partial \mathbf{y}_\pi}}_{= \frac{\partial \Sigma_t^{(a,b)}}{\partial \boldsymbol{\psi}}}$$

with respect to the hyper-parameters of the Gaussian basis functions and the measurement noise variance collected in $\boldsymbol{\theta}$, the pseudo-training inputs $\mathbf{X}_\pi$, and the pseudo-training targets $\mathbf{y}_\pi$ for the RBF policy in equation (3.10) are a bit lengthy, but straightforward to compute by repeated application of the chainrule and with the help of the partial derivatives given in Appendix A.2. The function values from which the derivatives can be derived, that is, the mean and the covariance of the predictive distribution $p(\mathbf{x}_t|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \boldsymbol{\psi})$, are given in equations (3.11) and (3.12), respectively.

3.6 Cost Function

In our learning problem, we assume that the immediate cost function c in equation (3.2) does not incorporate any solution-specific knowledge such as penalties on the control signal or speed variables (in regulator problems). An autonomous learner must be able to learn the remainder of the task by itself: If the system reaches the target state $\mathbf{x}_{\text{target}}$, but overshoots, the learning algorithm should account for this kind of failure in a next trial; the expected long-term cost for overshooting is higher than staying close to the target. We therefore employ a cost function that solely uses a geometric distance d of the current state to the target state. Thus, overshooting causes higher long-term cost than staying close to the target.

3.6.1 Saturating Cost

We propose to use the saturating immediate cost

$$c(\mathbf{x}) = 1 - \exp\left(-\tfrac{a^2}{2}\, d(\mathbf{x}, \mathbf{x}_{\text{target}})^2\right) \tag{3.18}$$

that is locally quadratic but which saturates at unity for large deviations d from the desired target $\mathbf{x}_{\text{target}}$ (blue function, solid, in Figure 3.11). In equation (3.18), the geometric distance from the state $\mathbf{x}$ to the target state is denoted by d, and the parameter $1/a$ controls the width of the cost function. In the context of sensorimotor control, the saturating cost function in equation (3.18) resembles the cost function in human reasoning as experimentally validated by Körding and Wolpert (2004b).

The immediate cost in equation (3.18) is an unnormalized Gaussian integrand with mean $\mathbf{x}_{\text{target}}$ and variance $1/a^2$ subtracted from unity. Therefore, the expected immediate cost can be computed analytically according to

$$\mathbb{E}_{\mathbf{x}}[c(\mathbf{x})] = 1 - \int c(\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1 - \int \exp\left(-\tfrac{1}{2}(\mathbf{x} - \mathbf{x}_{\text{target}})^\top \mathbf{T}^{-1}(\mathbf{x} - \mathbf{x}_{\text{target}})\right)p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,, \tag{3.19}$$

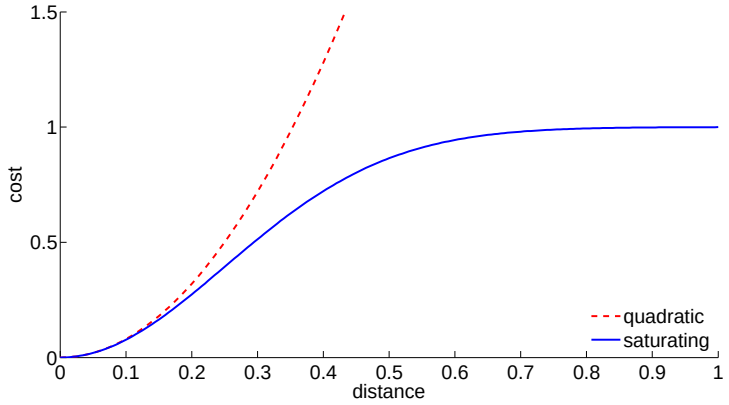
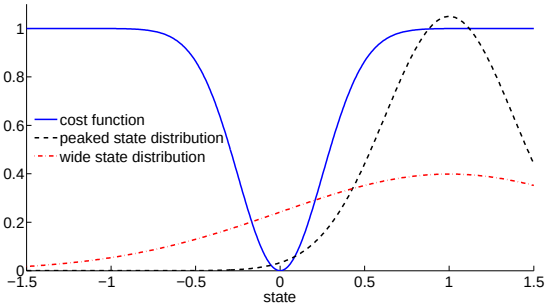
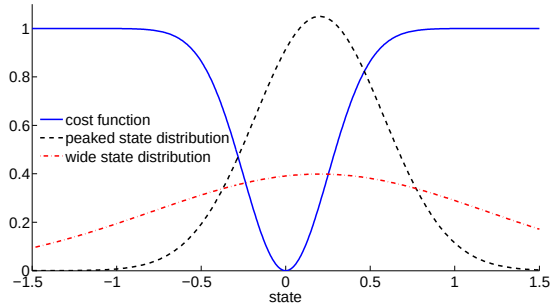


Figure 3.11: Quadratic (red, dashed) and saturating (blue, solid) cost functions. The x-axis shows the distance of the state to the target, the y-axis shows the corresponding immediate cost. Unlike the quadratic cost function, the saturating cost function can encode that a state is simply "far away" from the target. The quadratic cost function pays much attention to how "far away" the state really is.



(a) Initially, when the mean of the state is far away from the target, uncertain states (red, dashed-dotted) are preferred to more certain states with a more peaked distribution (black, dashed). This leads to initial exploration.



(b) Finally, when the mean of the state is close to the target, certain states with peaked distributions cause less expected cost and are therefore preferred to more uncertain states (red, dashed-dotted). This leads to exploitation once close to the target.

Figure 3.12: Automatic exploration and exploitation due to the saturating cost function (blue, solid). The x-axes describe the state space. The target state is the origin.

where $\mathbf{T}^{-1} = a^2\mathbf{C}^\top\mathbf{C}$ for suitable $\mathbf{C}$ is the precision matrix of the unnormalized Gaussian in equation (3.19).[16] If $\mathbf{x}$ is an input vector that has the same representation as the target vector, $\mathbf{T}^{-1}$ is a diagonal matrix with entries either unity or zero. Hence, for $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ we obtain the expected immediate cost

$$\mathbb{E}_{\mathbf{x}}[c(\mathbf{x})] = 1 - |\mathbf{I} + \boldsymbol{\Sigma}\mathbf{T}^{-1}|^{-1/2} \exp(-\tfrac{1}{2}(\boldsymbol{\mu} - \mathbf{x}_{\text{target}})^\top \tilde{\mathbf{S}}_1(\boldsymbol{\mu} - \mathbf{x}_{\text{target}}))\,, \tag{3.20}$$

$$\tilde{\mathbf{S}}_1 \coloneqq \mathbf{T}^{-1}(\mathbf{I} + \boldsymbol{\Sigma}\mathbf{T}^{-1})^{-1}\,. \tag{3.21}$$

Exploration and Exploitation

During learning (see Algorithm 1), the saturating cost function in equation (3.18) allows for "natural" exploration when the policy aims to minimizes the expected long-term cost in equation (3.2). This property is illustrated in Figure 3.12. If the mean of a state distribution $p(\mathbf{x}_t)$ is far away from the target $\mathbf{x}_{\text{target}}$, a wide state distribution is more likely to have substantial tails in some low-cost region than a fairly peaked distribution (Figure 3.12(a)). In the early stages of learning, the state uncertainty is essentially due to model uncertainty. If we encounter a state distribution in a high-cost region during internal simulation

[16] The covariance matrix does not necessarily exist and is not required to compute the expected cost. In particular, $\mathbf{T}^{-1}$ often does not have full rank.

(Figure 3.2(b) and layer two in Figure 3.3), the saturating cost then leads to automatic *exploration* by favoring uncertain states, that is, regions expectedly close to the target with a poor dynamics model. When visiting these regions in the interaction phase (Figure 3.2(a)), the subsequent model update (line 5 in Algorithm 1) reduces the model uncertainty.

If the mean of the state distribution is close to the target as in Figure 3.12(b), wide distributions are likely to have substantial tails in high-cost regions. By contrast, the mass of a peaked distribution is more concentrated in low-cost regions. In this case, the policy prefers peaked distributions close to the target, which leads to *exploitation*. Mathematically, the expected cost is a convolution of a Gaussian state distribution with the saturating cost function.

Hence, even for a policy aiming at the expected cost only, the combination of a probabilistic dynamics model and a saturating cost function leads to exploration as long as the states are far away from the target. Once close to the target, the policy does not substantially veer from a trajectory that lead the system to certain states close to the target.

One way to encourage further exploration is to modify the objective function in equation (3.2). Incorporation of the state uncertainty itself is an option, but this would lead to extreme designs as discussed by MacKay (1992). However, we are particularly interested in exploring promising regions of the state space, where "promising" is directly defined by value function V^π and the saturating cost function c in equation (3.18). Therefore, we consider the *variance of the predicted cost*

$$\operatorname{var}_{\mathbf{x}}[c(\mathbf{x})] = \mathbb{E}_{\mathbf{x}}[c(\mathbf{x})^2] - \mathbb{E}_{\mathbf{x}}[c(\mathbf{x})]^2\,, \quad \mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})\,, \tag{3.22}$$

where $\mathbb{E}_{\mathbf{x}}[c(\mathbf{x})]$ is given in equation (3.20). The second moment $\mathbb{E}_{\mathbf{x}}[c(\mathbf{x})^2]$ can be computed analytically and is given by

$$\mathbb{E}_{\mathbf{x}}[c(\mathbf{x})^2] = |\mathbf{I} + 2\boldsymbol{\Sigma}\mathbf{T}^{-1}|^{-1/2} \exp\big(-(\boldsymbol{\mu} - \mathbf{x}_{\text{target}})^\top \tilde{\mathbf{S}}_2 (\boldsymbol{\mu} - \mathbf{x}_{\text{target}})\big)\,, \tag{3.23}$$

$$\tilde{\mathbf{S}}_2 = \mathbf{T}^{-1}(\mathbf{I} + 2\boldsymbol{\Sigma}\mathbf{T}^{-1})^{-1}\,. \tag{3.24}$$

The variance of the cost (3.22) is then given by subtracting the square of equation (3.20) from equation (3.23).[17]

To encourage goal-directed exploration, we minimize the objective function

$$V^\pi(\mathbf{x}_0) = \sum_{t=0}^{T} \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] + b\,\sigma_{\mathbf{x}_t}[c(\mathbf{x}_t)]\,. \tag{3.25}$$

Here, $\sigma_{\mathbf{x}_t}$ is the standard deviation of the predicted cost. For $b < 0$ uncertainty in the cost is encouraged, for $b > 0$ uncertainty in the cost is penalized. Note that the modified value function in equation (3.25) is just an approximation to

$$\mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=0}^{T} c(\mathbf{x}_t)\right] + b\,\sigma_{\boldsymbol{\tau}}\left[\sum_{t=0}^{T} c(\mathbf{x}_t)\right]\,,$$

where the standard deviation of the predicted long-term cost along the trajectory $\boldsymbol{\tau}$ is considered, where $\boldsymbol{\tau} = (\mathbf{x}_0, \ldots, \mathbf{x}_T)$.

What is the difference between taking the variance of the state and the variance of the cost? The variance of the predicted cost at time t depends on the variance of the state: If the state distribution is fairly peaked, the variance of the corresponding cost is always small. However, an uncertain state does not necessarily cause a wide cost distribution: If the mean of the state distribution is in a high-cost region and the tails of the distribution do not substantially cover low-cost regions, the uncertainty of the predicted cost is very low. The only case the cost distribution can be uncertain is if a) the state is uncertain and b) a non-negligible part of the mass of the state distribution is in a low-cost region. Hence, using the uncertainty of the cost for exploration avoids extreme designs by solely exploring regions along trajectories passing regions that are somewhat close to the target—otherwise the objective function in equation (3.25) does not return small values.

[17] We represent the cost distribution $p(c(\mathbf{x}_t)|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ by its mean and variance. This representation is good when the mean is around 1/2, but can be fairly bad when the mean is close to a boundary, that is, zero or one. Then, the cost distribution resembles a Beta distribution with a one-sided heavy tail and a mode close to the boundary. By contrast, our chosen representation of the cost distribution can be interpreted to be a Gaussian distribution.

Partial Derivatives of the Saturating Cost

The partial derivatives

$$\frac{\partial}{\partial \boldsymbol{\mu}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)], \quad \frac{\partial}{\partial \boldsymbol{\Sigma}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]$$

of the immediate cost with respect to the mean and the covariance of the state distribution $p(\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, which are required in equation (3.14), are given by

$$\frac{\partial}{\partial \boldsymbol{\mu}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = -\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]\,(\boldsymbol{\mu}_t - \mathbf{x}_{\text{target}})^\top \tilde{\mathbf{S}}_1\,, \tag{3.26}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = \tfrac{1}{2}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]\left(\tilde{\mathbf{S}}_1(\boldsymbol{\mu}_t - \mathbf{x}_{\text{target}})(\boldsymbol{\mu}_t - \mathbf{x}_{\text{target}})^\top - \mathbf{I}\right)\tilde{\mathbf{S}}_1\,, \tag{3.27}$$

respectively, where $\tilde{\mathbf{S}}_1$ is given in equation (3.21). Additional partial derivatives are required if the objective function (3.25) is used to encourage additional exploration. These partial derivatives are

$$\frac{\partial}{\partial \boldsymbol{\mu}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)^2] = -2\,\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)^2](\boldsymbol{\mu} - \mathbf{x}_{\text{target}})^\top \tilde{\mathbf{S}}_2\,,$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)^2] = 2\,\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)^2]\tilde{\mathbf{S}}_2(\boldsymbol{\mu} - \mathbf{x}_{\text{target}})(\boldsymbol{\mu} - \mathbf{x}_{\text{target}})^\top \tilde{\mathbf{S}}_2\,,$$

where $\tilde{\mathbf{S}}_2$ is given in equation (3.24).

3.6.2 Quadratic Cost

A common cost function used in optimal control (particularly in combination with linear systems) is the quadratic cost (see red-dashed curve in Figure 3.11)

$$c(\mathbf{x}) = a^2\, d(\mathbf{x}, \mathbf{x}_{\text{target}})^2 \geq 0\,. \tag{3.28}$$

In equation (3.28), d is the distance from the current state to the target state and a^2 is a scalar parameter controlling the width of the cost parabola. In a general form, the quadratic cost, its expectation, and its variance are given by

$$c(\mathbf{x}) = a^2\, d(\mathbf{x}, \mathbf{x}_{\text{target}})^2 = (\mathbf{x} - \mathbf{x}_{\text{target}})^\top \mathbf{T}^{-1}(\mathbf{x} - \mathbf{x}_{\text{target}}) \tag{3.29}$$

$$\mathbb{E}_{\mathbf{x}}[c(\mathbf{x})] = \operatorname{tr}(\boldsymbol{\Sigma}\mathbf{T}^{-1}) + (\boldsymbol{\mu} - \mathbf{x}_{\text{target}})^\top \mathbf{T}^{-1}(\boldsymbol{\mu} - \mathbf{x}_{\text{target}}) \tag{3.30}$$

$$\operatorname{var}_{\mathbf{x}}[c(\mathbf{x})] = \operatorname{tr}(2\,\mathbf{T}^{-1}\boldsymbol{\Sigma}\mathbf{T}^{-1}\boldsymbol{\Sigma}) + 4\,(\boldsymbol{\mu} - \mathbf{x}_{\text{target}})^\top \mathbf{T}^{-1}\boldsymbol{\Sigma}\mathbf{T}^{-1}(\boldsymbol{\mu} - \mathbf{x}_{\text{target}})\,, \tag{3.31}$$

respectively, where $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma},)$ and $\mathbf{T}^{-1}$ is a symmetric matrix that also contains the scaling parameter a^2 in equation (3.29). In the quadratic cost function, the scaling parameter a^2 has theoretically no influence on the solution to the optimization problem: The optimum of V^π is always the same independent of a^2.[18]

Partial Derivatives of the Quadratic Cost

The partial derivatives

$$\frac{\partial}{\partial \boldsymbol{\mu}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)], \quad \frac{\partial}{\partial \boldsymbol{\Sigma}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]$$

of the quadratic cost with respect to the mean and the covariance of the state distribution $p(\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, which are required in equation (3.14), are given by

$$\frac{\partial}{\partial \boldsymbol{\mu}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = 2\,(\boldsymbol{\mu}_t - \mathbf{x}_{\text{target}})^\top \mathbf{T}^{-1}\,, \tag{3.32}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_t}\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = \mathbf{T}^{-1}\,, \tag{3.33}$$

[18] From a practical point of view, the gradient-based optimizer can be very sensitive to the choice of a^2.

respectively. Additional partial derivatives are required if the objective function (3.25) is used to encourage additional exploration. These partial derivatives are given by

$$\frac{\partial}{\partial \boldsymbol{\mu}_t} \mathrm{var}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = -8\,\mathbf{T}^{-1}\boldsymbol{\Sigma}_t\mathbf{T}^{-1}(\boldsymbol{\mu}_t - \mathbf{x}_{\mathrm{target}})^\top\,,$$
$$\frac{\partial}{\partial \boldsymbol{\Sigma}_t} \mathrm{var}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = 4\mathbf{T}^{-1}\boldsymbol{\Sigma}_t\mathbf{T}^{-1} + 4\mathbf{T}^{-1}(\boldsymbol{\mu}_t - \mathbf{x}_{\mathrm{target}})(\mathbf{T}^{-1}(\boldsymbol{\mu}_t - \mathbf{x}_{\mathrm{target}}))^\top\,,$$

respectively.

Potential Problems with the Quadratic Cost

A first problem with the quadratic cost function is that the expected long-term cost in equation (3.2) is highly dependent on the worst state along a predicted state trajectory: The state covariance $\boldsymbol{\Sigma}_t$ is an additive linear contribution to the expected cost in equation (3.30). By contrast, in the saturating cost function, the state uncertainty scales the distance between the mean $\boldsymbol{\mu}_t$ and the target in equation (3.19). Here, high variances can only occur close to the target.

A second problem with the quadratic cost is that the value function in equation (3.2) is highly sensitive to details of a distribution that essentially encode that the model has lost track of the state. In particular in the early stages of learning, the predictive state uncertainty may grow rapidly with the time horizon while the mean of predicted state is far from $\mathbf{x}_{\mathrm{target}}$. The partial derivative in equation (3.32) depends on the (signed) distance between the predicted mean and the target state. As opposed to the corresponding partial derivative in the saturating cost function (see equation (3.26)), the signed distance in equation (3.32) is *not* weighted by the inverse covariance matrix, which results in steep gradients even when the prediction is highly uncertain. Therefore, the optimization procedure is highly dependent on how much the predicted means $\boldsymbol{\mu}_t$ differ from the target state $\mathbf{x}_{\mathrm{target}}$ even when the model lost track of the state, which can be considered an arbitrary detail of the predicted state distribution $p(\mathbf{x}_t)$.

The desirable exploration properties of the saturating cost function that have been discussed in Section 3.6.1 cannot be directly transferred to the quadratic cost: The variance of the quadratic cost in equation (3.31) increases if the state $\mathbf{x}_t$ becomes uncertain and/or if the mean $\boldsymbol{\mu}_t$ of $\mathbf{x}_t$ is far away from the target $\mathbf{x}_{\mathrm{target}}$. Unlike the saturating cost function in equation (3.18), the variance of the quadratic cost function in equation (3.29) is unbounded and depends quadratically on the state covariance matrix. Moreover, the term involving the state covariance matrix is additively connected with a term that depends on the scaled squared distance between the mean $\boldsymbol{\mu}_t$ and the target state $\mathbf{x}_{\mathrm{target}}$. Thus, exploration using $\mathrm{var}_{\mathbf{x}}[c(\mathbf{x})]$ is not goal-directed: Along a predicted trajectory, uncertain states[19] and/or states far away from the target are favored. Hence, the variance of the quadratic cost function essentially boils down to grow unbounded with the state covariance, a setting that can lead to "extreme designs" (MacKay, 1992).

Due to these issues, we use the saturating cost in equation (3.18) instead.

3.7 Results

The algorithmic framework for efficient RL was applied to learning models and controllers for inherently unstable dynamic systems, where the applicable control signals were constrained. We present *typical* empirical results, that is, results that carry the flavor of a typical experiment we conducted. In all control tasks, we followed the high-level idea described in Algorithm 1.

Interactions

Pictorially, our algorithm used the learned model as an internal representation of the world as described by Figure 3.2. When we worked with hardware, the world was given by the mechanical system itself. Otherwise, the world was defined by the emulation of the mechanical systems. For this purpose, we used an ODE solver for numerical simulation of the corresponding equations of motion. The equations of motion were given by a set of coupled ordinary second-order differential equations $\ddot{\mathbf{s}} = f(\dot{\mathbf{s}}, \mathbf{s}, \mathbf{u})$. Then,

[19] This means either states that are far away from the current GP training set or states where the function value highly varies.

Algorithm 2 Detailed implementation of learning algorithm

1: **init:** $\boldsymbol{\psi}, p(\mathbf{x}_0), a, b, T_{\text{init}}, T_{\max}, N, \Delta_t$ ▷ initialization
2: $T \coloneqq T_{\text{init}}$ ▷ initial prediction horizon
3: generate initial training set $\mathcal{D} = \mathcal{D}_{\text{init}}$ ▷ initial interaction phase
4: **for** $j = 1$ to PS **do**
5: learn probabilistic dynamics model based on $\mathcal{D}$ ▷ update model (batch)
6: find $\boldsymbol{\psi}^*$ with $\pi^*_{\boldsymbol{\psi}^*} \in \arg\min_{\pi_{\boldsymbol{\psi}} \in \Pi} V^{\pi_{\boldsymbol{\psi}}}(\mathbf{x}_0) \,|\, \mathcal{D}$ ▷ policy search via internal simulation
7: compute $\mathcal{A}_t \coloneqq \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)|\pi^*_{\boldsymbol{\psi}^*}]$, $t = 1, \ldots, T$
8: **if** `task_learned`$(\mathcal{A})$ **and** $T < T_{\max}$ **then** ▷ if good solution predicted
9: $T \coloneqq 1.25\, T$ ▷ increase prediction horizon
10: **end if**
11: apply $\pi^*_{\boldsymbol{\psi}^*}$ to the system for T time steps, observe $\mathcal{D}_{\text{new}}$ ▷ interaction
12: $\mathcal{D} \coloneqq \mathcal{D} \cup \mathcal{D}_{\text{new}}$ ▷ update data set
13: **end for**
14: **return** π^* ▷ return learned policy

the ODE solver computed the state

$$\mathbf{x}_{t+1} \coloneqq \begin{bmatrix} \mathbf{s}(t+1) \\ \dot{\mathbf{s}}(t+1) \end{bmatrix} = \int_t^{t+\Delta_t} \begin{bmatrix} \dot{\mathbf{s}}(\tau) \\ f\big(\dot{\mathbf{s}}(\tau), \mathbf{s}(\tau), \mathbf{u}(\tau)\big) \end{bmatrix} \mathrm{d}\tau \tag{3.34}$$

during the interaction phase, where Δ_t is a time discretization constant (in seconds). Note that the system was simulated in continuous time, but the control decision could only be changed every Δ_t, that is, at the discrete time instances when the state of the system was measured.

Learning Procedure

Algorithm 2 describes a more detailed implementation of the high-level idea. In the following, we go through the lines of the algorithm. We distinguish between "automatic" steps that directly follow from the proposed learning framework and fairly general heuristics, which we used for computational convenience. Let us start with the automatic steps: First, the policy parameters $\boldsymbol{\psi}$ were initialized (line 1).[20] Moreover, we passed in the distribution $p(\mathbf{x}_0)$ of the initial state, the width $1/a$ of the saturating immediate cost function c in equation (3.18), the exploration parameter b, the prediction horizon T, the number PS of policy searches, and the time discretization constant Δ_t. An initial training set $\mathcal{D}_{\text{init}}$ for the dynamics model was generated by applying actions randomly (drawn uniformly from $[-\mathbf{u}_{\max}, \mathbf{u}_{\max}]$) to the system (line 3).[21] For PS policy searches, we followed the steps in the loop in Algorithm 2: A probabilistic model of the transition dynamics was learned using all previous experience collected in the data set $\mathcal{D}$ (line 5). Using this model, we simulated the dynamics forward internally, evaluated the objective function V^π (see Section 3.4), and optimized the policy parameters $\boldsymbol{\psi}$ for the current model (line 6 and Section 3.5). Then, the policy was applied to the system (line 11) and the data set was augmented by the new experience from this interaction phase (line 12). These steps were automatic steps and did not require any deep heuristic.

Thus far, we have not explained line 7 and the if-statement in line 8, where the latter one does involve a heuristic. In line 7, we computed the expected values of the predicted costs given by $\mathcal{A}_t \coloneqq \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)|\pi^*_{\boldsymbol{\psi}^*}]$, $t = 1, \ldots, T$, when following the optimized policy $\pi^*_{\boldsymbol{\psi}^*}$. The function `task_learned` uses $\mathcal{A}_T$ to determine whether a good path to the target is expected to be found (line 8): When the expected cost $\mathcal{A}_T$ at the end of the prediction horizon was small below 0.2, the system state at time T was predicted to be close to the target.[22] When `task_learned` reported success, we increased the current prediction horizon T by

[20] We initialized the policy parameters randomly: For the linear policy in equation (3.9), the parameters were sampled from a zero-mean Gaussian with variance 0.01. For the RBF policy in equation (3.10), the means of the basis functions were sampled from a Gaussian with variance 0.1 around $\mathbf{x}_0$. The hyper-parameters and the pseudo-training targets were sampled from a zero-mean Gaussian with variance 0.01.

[21] With the "system" we either mean a hardware realization or a computer program that emulates the mechanical system.

[22] The following simplified illustration might clarify our statement: Suppose a cost of 1 incurs if the task cannot be solved, and a cost of 0 incurs if the task can be solved. An expected value of 0.2 of this Bernoulli variable requires a predicted success rate of 0.8.

Table 3.1: Overview of conducted experiments.

experiment	section
single nonlinear controller	3.7.1, 3.7.2, 3.7.3
hardware applicability	3.7.1
importance of RL ingredients	3.7.1
multivariate controls	3.7.2, 3.7.4
different implementations of $\mathbf{u}(\tau)$	3.7.1

25% (line 9) and initialized the new task for the extended horizon by the policy that was expected to succeed for the shorter horizon. Initializing the learning task for a longer horizon by the solution for the shorter horizon can be considered a continuation method for learning. We refer to the paper by Richter and DeCarlo (1983) for details on continuation methods.

Stepwise increasing the horizon (line 9) mimics human learning for episodic tasks and simplifies artificial learning since the prediction and the optimization problems are easier for relatively short time horizons T: In particular in the early stages of learning when the dynamics model was very uncertain, most visited states along a long trajectory did not contribute much to goal-directed learning as they were almost random. Instead, they made learning the dynamics model more difficult since only the first seconds of a controlled trajectory conveyed valuable information for solving the task.

Overview of the Experiments

We applied our learning framework to four different nonlinear control tasks: the cart-pole problem (Section 3.7.1), the Pendubot (Section 3.7.2), the cart-double pendulum (Section 3.7.3), and the problem of riding a unicycle (Section 3.7.4). In all cases, we learned models of the system dynamics and controllers that solve the individual tasks.

The objective of this section is to shed light on the generality, applicability, and performance of our proposed framework by addressing the following questions in our experiments:

- Is the learning framework applicable to different tasks when only the parameter initializations (line 1 in Algorithm 2) can be changed?
- How many trials are necessary to learn the corresponding tasks?
- Is it possible to learn a single nonlinear controller, where standard methods apply multiple linear controllers?
- Is the learning framework applicable to hardware at all?
- Are the probabilistic model and/or the saturating cost function and/or the approximate inference algorithm using moment matching crucial for the success of our learning framework? What is the effect of replacing our suggested components in Figure 3.4?
- Can our algorithm naturally deal with multivariate control signals, which corresponds to having multiple actuators?
- What are the effects of different implementations of the continuous-time control signal $\mathbf{u}(\tau)$ in equation (3.34)?

Table 3.1 gives an overview in which sections these questions are addressed. Most of the questions are discussed in the context of the cart-pole problem (Section 3.7.1) and the Pendubot (Section 3.7.2). The hardware applicability was only tested on the cart-pole task since no other hardware equipment was available.

Algorithm 3 Evaluation setup

1: find policy π^* using Algorithm 2 ▷ learn policy
2: **for** $i = 1$ **to** 1,000 **do** ▷ loop over different rollouts
3: $\mathbf{x}_0^{(i)} \sim p(\mathbf{x}_0)$ ▷ sample initial state
4: observe trajectory $\boldsymbol{\tau}_i | \pi^*$ ▷ follow π^* starting from $\mathbf{x}_0^{(i)}$
5: evaluate π^* using $\boldsymbol{\tau}_i$ ▷ evaluate policy
6: **end for**

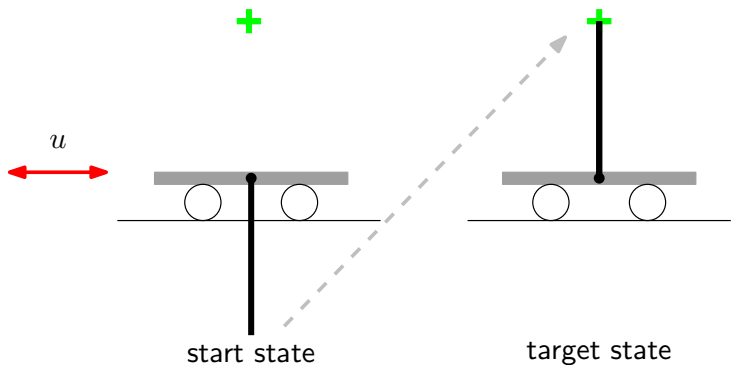


Figure 3.13: Cart-pole setup. The cart-pole system consists of a cart running on an infinite track and a freely swinging pendulum attached to the cart. Starting from the state where the pendulum hangs down, the task was to swing the pendulum up and to balance it in the inverted position at the cross by just pushing the cart to left and right. Note that the cart had to stop exactly below the cross in order to fulfill the task optimally.

Setup of the Evaluations

Algorithm 3 describes the setup that was used to evaluate our learning framework. For each task, initially, a policy π^* was learned by following Algorithm 2. Then, an initial state $\mathbf{x}_0^{(i)}$ was sampled from the state distribution $p(\mathbf{x}_0)$, where $i = 1, \ldots, 1,000$. Starting from $\mathbf{x}_0^{(i)}$, the policy was applied and the resulting state trajectory $\boldsymbol{\tau}_i = (\mathbf{x}_0^{(i)}, \ldots, \mathbf{x}_T^{(i)})$ was observed and used for the evaluation. Note that the policy was fixed in all rollouts used for the evaluation.

3.7.1 Cart Pole (Inverted Pendulum)

We applied our approach to learning a model and a controller for the under-actuated cart-pole problem, also called the inverted pendulum, whose setup is described in Figure 3.13. The system consists of a cart running on an infinite track and a pendulum attached to the cart. An external force can be applied to the cart, but not to the pendulum. The dynamics of the cart-pole system are derived from first principles in Appendix B.1.

The state $\mathbf{x}$ of the system was given by the position x_1 of the cart, the velocity $\dot{x}_1$ of the cart, the angle θ_2 of the pendulum, and the angular velocity $\dot{\theta}_2$ of the pendulum. The angle was measured anti-clockwise from hanging down. During simulation, the angle was represented as a complex number on the unit circle, that is, the angle was mapped to its sine and cosine. Therefore, the state was represented as

$$\mathbf{x} = \begin{bmatrix} x_1 & \dot{x}_1 & \dot{\theta}_2 & \sin\theta_2 & \cos\theta_2 \end{bmatrix}^\top \in \mathbb{R}^5 .$$

On the one hand we could exploit the periodicity of the angle that naturally comes along with this augmented state representation, on the other hand the dimensionality of the state increased by one (the number of angles).

Initially, the system was expected to be a state, where the pendulum hangs down ($\boldsymbol{\mu}_0 = \mathbf{0}$). By pushing the cart to the left and to the right, the objective was to swing the pendulum up and to balance it in the inverted position around at the target state (green cross in Figure 3.13), such that $x_1 = 0$ and $\theta_2 = \pi + 2k\pi$, $k \in \mathbb{Z}$. Doya (2000) considered a similar task, where the target position was upright, but

the target location of the cart was arbitrary. Instead of just balancing the pendulum, we additionally required the tip of the pendulum to be balanced at a specific location given by the cross in Figure 3.13. Optimally solving the task required the cart to stop at a particular position.

Note that a linear controller is incapable to swing the pendulum up and to balance it in the inverted position (Raiko and Tornio, 2009). Therefore, our system learned the nonlinear RBF policy given in equation (3.10). The cart-pole problem is non trivial to solve since sometimes actions have to be taken that temporarily move the pendulum further away from the target. Thus, optimizing a short-term cost does not lead to success.

In control theory, the cart-pole task is a classical benchmark for nonlinear optimal control. However, typically the task is solved using two controllers: one for the swing up and the second one for the balancing task. The control theory solution is based on an intricate understanding of the dynamics of the system (parametric system identification) and of how to solve the task (switching controllers), neither of which we assumed in this dissertation. Instead, our objective was to find a good policy without an intricate understanding of the system, which we consider expert knowledge. Unlike the classical solution, we attempted to *learn* a single nonlinear RBF controller to solve the entire cart-pole task.

The parameters used in the experiment are given in Appendix C.1. The chosen time discretization of $\Delta_t = 0.1\,\mathrm{s}$ corresponds to a rather slow sampling frequency of 10 Hz. Therefore, the control signal could only be changed every $\Delta_t = 0.1\,\mathrm{s}$, which made the control task fairly challenging.

Cost Function

Every $\Delta_t = 0.1\,\mathrm{s}$, the squared Euclidean distance

$$d(\mathbf{x}, \mathbf{x}_{\text{target}})^2 = x_1^2 + 2x_1 l \sin\theta_2 + 2l^2 + 2l^2 \cos\theta_2 \tag{3.35}$$

from the tip of the pendulum to the inverted position (green cross in Figure 3.13) was measured. Note, that d only depends on the position variables x_1, $\sin\theta_2$, and $\cos\theta_2$. In particular, it does not depend on the velocity variables $\dot{x}_1$ and $\dot{\theta}_2$. An approximate Gaussian joint distribution $p(\mathbf{j}) = \mathcal{N}(\mathbf{m}, \mathbf{S})$ of the involved state variables

$$\mathbf{j} \coloneqq \begin{bmatrix} x_1 & \sin\theta_2 & \cos\theta_2 \end{bmatrix}^\top \tag{3.36}$$

was computed using the results from Appendix A.1. The target vector in $\mathbf{j}$-space was $\mathbf{j}_{\text{target}} = [0, 0, -1]^\top$. The saturating immediate cost was then given as

$$c(\mathbf{x}) = c(\mathbf{j}) = 1 - \exp\left(-\tfrac{1}{2}(\mathbf{j} - \mathbf{j}_{\text{target}})^\top \mathbf{T}^{-1} (\mathbf{j} - \mathbf{j}_{\text{target}})\right) \in [0, 1]\,, \tag{3.37}$$

where the matrix $\mathbf{T}^{-1}$ in equation (3.19) was given by

$$\mathbf{T}^{-1} \coloneqq a^2 \begin{bmatrix} 1 & l & 0 \\ l & l^2 & 0 \\ 0 & 0 & l^2 \end{bmatrix} = a^2 \mathbf{C}^\top \mathbf{C} \qquad \text{with} \quad \mathbf{C} \coloneqq \begin{bmatrix} 1 & l & 0 \\ 0 & 0 & l \end{bmatrix}. \tag{3.38}$$

The parameter $1/a$ controlled the width of the saturating immediate cost function in equation (3.18). Note that when multiplying $(\mathbf{j} - \mathbf{j}_{\text{target}})$ from the left and the right to $\mathbf{C}^\top \mathbf{C}$, the squared Euclidean distance d^2 in equation (3.35) is recovered.

The length-scale of the cost function in equation (3.38) was set to $1/a = 0.25\,\mathrm{m}$. Thus, the immediate cost in equation (3.18) did not substantially differ from unity if the distance of the pendulum tip to the target was greater than l, the pendulum length, requiring the tip of the pendulum to move above horizontal: A distance l from the target was outside the $2\,\sigma$ interval of the immediate cost function.

Zero-order-hold Control

Following Algorithm 3, a policy π^* was learned using the proposed RL framework. Note that the policy was not modified during the evaluation. The control signal was implemented using *zero-order-hold* control. Zero-order hold converts a discrete-time signal $\mathbf{u}_{t-1}$ into a continuous-time signal $\mathbf{u}(t)$ by holding $\mathbf{u}_{t-1}$ for one time interval Δ_t.

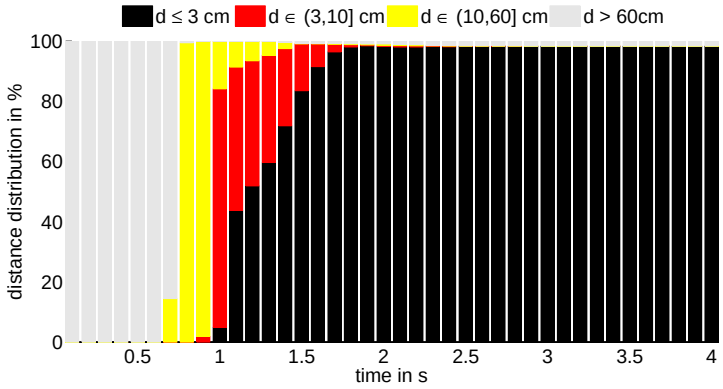


Figure 3.14: Histogram of the distances d from the tip of the pendulum to the target of 1,000 rollouts. The x-axis shows the time, the heights of the bars represent the percentages of trajectories that fell into the respective categories of distances to the target. After a transient phase, the controller could either solve the problem very well (black bars) or it could not solve it at all (gray bars).

Figure 3.14 shows a histogram of the empirical distribution of the distance d from the tip of the pendulum to the target over time when applying a learned policy. The histogram is based on 1,000 rollouts starting from states that were independently drawn from $p(\mathbf{x}_0) = \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_0)$. The figure distinguishes between four intervals of distances from the tip of the pendulum to the target position: $d(\mathbf{x}_t, \mathbf{x}_{\text{target}}) \leq 3\,\text{cm}$ (black), $d(\mathbf{x}_t, \mathbf{x}_{\text{target}}) \in (3, 10]$ cm (red), $d(\mathbf{x}_t, \mathbf{x}_{\text{target}}) \in (10, 60]$ cm (yellow), and $d(\mathbf{x}_t, \mathbf{x}_{\text{target}}) > 60\,\text{cm}$ (gray) for $t = 0\,\text{s}, \ldots, 4\,\text{s}$. The graph is cut at 4 s since the histogram does not change much for $t > 4\,\text{s}$. The heights of the bars at time t show the percentage of distances that fall into the respective intervals at this point in time. For example, after 1 s, in approximately 5% of the 1,000 rollouts, the pendulum tip was closer to the target than 3 cm (height of the black bar), in about 79% of the rollouts it was between 3 cm and 10 cm (height of the red bar), and in all other trajectories at time 1 s, the pendulum tip was between 10 cm and 60 cm away from the target. The gray bars show the percentage of trajectories at time t where the tip of the pendulum was further away from the target than the length of the pendulum (60 cm), which essentially caused full cost. At the beginning of each trajectory, all states were in a high-cost regime since in the initial state the pendulum hung down; the gray bars are full. From 0.7 s the pendulum started getting closer to the target: the yellow bars start appearing. After 0.8 s the gray bars almost disappear. This means that in essentially all rollouts the tip of the pendulum came closer to the target than the length of the pendulum. After 1 s the first trajectories got close to the target since the black bar starts appearing. The controller managed to stabilize the pendulum in the majority of the rollouts, which is shown by the increasing black bars between 1 s and 1.7 s. After 1.5 s, the size of the gray bars slightly increase again, which indicates that in a few cases the pendulum moved away from the target and fell over. In approximately 1.5% of the rollouts, the controller could not balance the pendulum close to the target state for $t \geq 2\,\text{s}$. This is shown by the gray bar that is approximately constant for $t \geq 2.5\,\text{s}$. The red and the yellow bars almost disappear after about 2 s. Hence, the controller could either (in 98.3% of the rollouts) solve the problem very well (height of the black bar) or it could not solve it at all (height of the gray bar).

In the 1,000 rollouts used for testing, the controller failed to solve the cart-pole task when the system encountered states that were not close to previously visited states captured by the training set. For instance, if the initial state of a trajectory was in a tail region of $\mathbf{\Sigma}_0$ it could happen that either the dynamics model was not good and/or the controller did not pay much attention to this region before. Note that the dynamics model was trained on twelve trajectories only. The controller's robustness can be increased if the data of a trajectory that led to a failure is incorporated into the dynamics model. However, we have not yet investigated this idea.

Figure 3.15 shows the medians (solid lines), the α-quantiles, and the $1 - \alpha$-quantiles, $\alpha = 0.1$, of the distribution of the predicted immediate cost (blue, dashed), $t = 0\,\text{s}, \ldots, 6.3\,\text{s} = T_{\text{max}}$, and the cor-

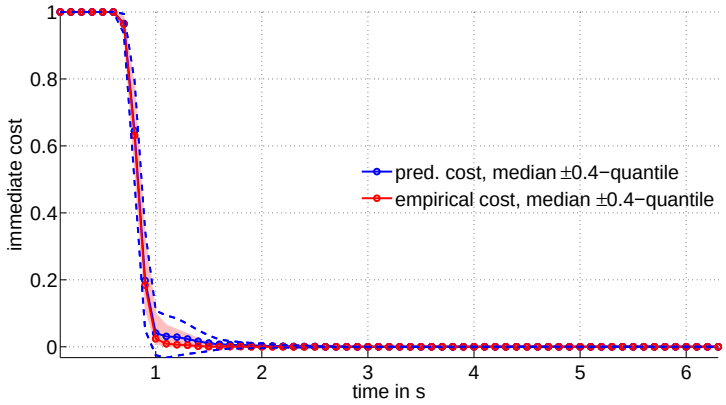


Figure 3.15: Medians and quantiles of the predictive immediate cost distribution (blue) and the empirical immediate cost distribution (red). The x-axis shows the time, the y-axis shows the immediate cost.

responding empirical cost distribution (red, shaded) after 1,000 rollouts.[23,24] The predictive immediate cost is based on the multiple-step ahead predictive distribution of $p(\mathbf{x}_0)$ when following the current policy π^* (intermediate layer in Figure 3.3). The medians of the distributions are close to each other. However, the error bars of the predictive distribution (blue, dashed) are larger than the error bars of the empirical distribution (red, shaded) when the predicted cost approaches zero at about $t = 1\,\mathrm{s}$. This is due to the fact that the predicted cost distribution is represented by its mean and its variance, but the empirical cost distribution at $t = 1\,\mathrm{s}$ rather resembles a Beta distribution with a one-sided heavy tail: In most cases, the tip of the pendulum was fairly close to the target (see also the black and red bars in Figure 3.14), but in the cases where the pendulum could not be stabilized, the tip of the pendulum went far away from the target incurring full immediate cost (gray bars in Figure 3.14). However, after about 1.6 s, the quantiles of both distributions converge toward the medians, and the medians are almost identical to zero. The median of the predictive distribution of the immediate cost implies that no failure was predicted.[25] Otherwise, the median of the Gaussian approximation of the predicted immediate cost would significantly differ from zero. The small bump in the error bars between 1 s and 1.6 s describes the time where the RBF controller switched from the swing-up action to balancing. Note, however, that only a single controller was learned.

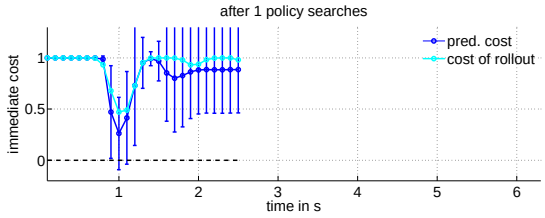
Figure 3.16 shows six examples of the predicted cost and the cost of an actual rollout during learning, that is after 1, 4, 6, 7, 8, and 12 policy searches. The predicted cost distributions $p(c(\mathbf{x}_t))$, $t = 1, \ldots, T$, are represented by their means and error bars denoting the corresponding 95% confidence intervals. The cost distributions $p(c(\mathbf{x}_t))$ are based on multiple-step ahead predictions of $p(\mathbf{x}_0)$ when following the currently optimal policy.

In Figure 3.16(a), that is, after one policy search, we see that for the first roughly 0.7 s the system did not enter states with a cost that significantly differed from unity. Between $t = 0.8\,\mathrm{s}$ and $t = 1\,\mathrm{s}$ a decline in cost was predicted. Simultaneously, a rapid increase of the predicted error bars can be observed. This reflects the initially poor dynamics model since the system never observed any data in this region: The dynamics training set so far was solely based on a single random initial trajectory. When applying the found policy to the system, we see (in the solid cyan line) that indeed the cost did decrease after about 0.8 s. The predicted error bars at around $t = 1.4\,\mathrm{s}$ were small and the mean of the predicted state was far from the target, that is, almost full cost was predicted. After the predicted fall-over, the system lost track of the state indicated by the increasing error bars and the mean of the predicted cost settling down at a value of approximately 0.8. More specifically, the system predicted that the pendulum swung through, but it simultaneously lost track of the phase; the GP model used to predict the angular velocity was very uncertain at this stage of learning. Thus, roughly speaking, to compute the mean and the variance of the

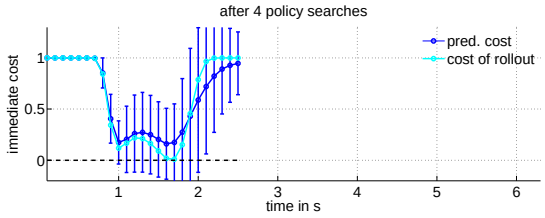
[23] For the purpose of the *evaluation*, we represent the distributions of the predicted immediate cost by Gaussians with the exact means and variances. The approximation of the cost distribution is not necessary during *learning* when only the means and the variances of the distributions, but not the distributions themselves, are required.

[24] Due to the Gaussian approximation of the cost distributions, the medians equal the means.

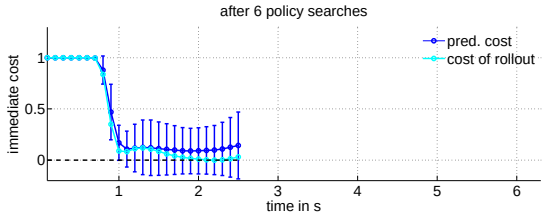
[25] Here, we use the fact that the median and the mean of a Gaussian distribution are identical.



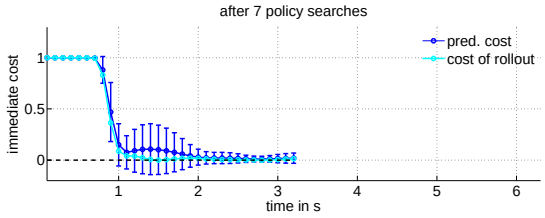
(a) Cost when applying a policy based on 2.5 s experience.



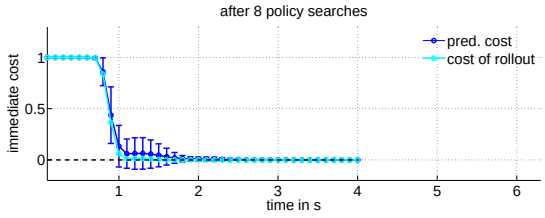
(b) Cost when applying a policy based on 10 s experience.



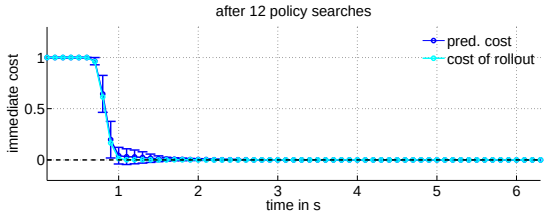
(c) Cost when applying a policy based on 15 s experience.



(d) Cost when applying a policy based on 18.2 s experience.



(e) Cost when applying a policy based on 22.2 s experience.



(f) Cost when applying a policy based on 46.1 s experience.

Figure 3.16: Predicted cost and incurred immediate cost during learning (after 1, 4, 6, 7, 8, and 12 policy searches, from top left to bottom right). The x-axes show the time in seconds, the y-axes show the immediate cost. The black dashed line is the minimum immediate cost (zero). The blue solid line is the mean of the predicted cost. The error bars show the corresponding 95% confidence interval of the predictions. The cyan solid line is the cost incurred when the new policy is applied to the system. Due to a heuristic (see line 9 in Algorithm 2), the prediction horizon T was increased when a low cost at the end of the current horizon was predicted (see line 9 in Algorithm 2). The cart-pole task can be considered learned after eight policy searches since the error bars at the end of the trajectories are negligible and do not increase when the prediction horizon increases.

immediate cost distribution for $t \geq 1.5\,\mathrm{s}$, the predictor had to average over all angles on the unit circle[26]. Since some of the angles were in a low-cost region, the mean of the corresponding predicted immediate cost settled down *below* unity. The trajectory of the incurred cost (cyan) confirms the prediction. The observed state trajectory led to an improvement in the subsequently updated dynamics model (see line 5 in Algorithm 2).

In Figure 3.16(b), the model had seven and a half seconds more experience, also including some states closer to the goal state. The trough of predicted low cost at $t = 1\,\mathrm{s}$ got extended to a length of approximately a second. After $t = 2\,\mathrm{s}$, the mean of the predicted cost increased and fell back close to unity at the end of the predicted trajectory. Compared to Figure 3.16(a), the error bars for $t \leq 1\,\mathrm{s}$ and for $t \geq 2.5\,\mathrm{s}$ got smaller due to an improved dynamics model. The actual rollout shown in cyan was in agreement with the prediction.

In Figure 3.16(c) (after six policy searches), the mean of the predicted immediate cost did no longer fall back to unity, but stayed at a value slightly below 0.2 with relatively small error bars. This means that the controller was fairly certain that the cart-pole task could be solved (otherwise the predicted mean would not have leveled out below 0.2), although the dynamics model still conveyed a fair amount of uncertainty. The actual rollout did not only agree with the prediction, but it solved the task better than

[26] The position of the cart was predicted to be close to its final destination with small uncertainty.

Table 3.2: Experimental results: cart-pole with zero-order-hold control.

interaction time	46.1 s
task learned (negligible error bars)	after 22.2 s (8 trials)
failure rate ($d > l$)	1.5%
success rate ($d \leq 3\,\text{cm}$)	98.3%
$V^{\pi^*}(\mathbf{x}_0)$, $\boldsymbol{\Sigma}_0 = 10^{-2}\mathbf{I}$	8.0

expected. Therefore, many data points close to the target state could be incorporated into the subsequent update of the dynamics model. Since the mean of the predicted cost at the end of the trajectory was smaller than 0.2, we employed the heuristic in line 9 of Algorithm 2 and increased the prediction horizon T increased by 25% for the next policy search.

The result after the next policy search and with the increased horizon $T = 3.2\,\text{s}$ is shown in Figure 3.16(d). Due to the improved dynamics model, the expected cost at the end of the trajectory declined to a value slightly above zero; the error bars shrank substantially compared to the previous predictions. The bump in the error bars between $t = 1\,\text{s}$ and $t = 2\,\text{s}$ is at the time when the RBF controller switched from swinging up to balancing: Slight oscillations around the target state could occur. Since the predicted mean at the end of the trajectory had a small value, using the heuristic in line 9 of Algorithm 2, the prediction horizon was increased again (to $T = 4\,\text{s}$) for the subsequent policy search. The predictions after the eighth policy search are shown in Figure 3.16(e). The error bars at the end of the trajectory collapsed, and the predicted mean leveled out at a value of zero. The task was essentially solved at this time.

Iterating the learning procedure for more steps resulted in a quicker swing-up action and in even smaller error bars both during the swing up (between 0.8 s and 1 s) and during the switch from the swing up to balancing (between 1 s and 1.8 s). From now onward, the prediction horizon was increased after each policy search until $T = T_{\text{max}}$. The result after the last policy search in our experiment is shown in Figure 3.16(f). The controller found a way to reduce the oscillations around the target, which is shown by the reduction of the bump after $t = 1\,\text{s}$. Furthermore, the error bars collapsed after $t = 1.6\,\text{s}$, and the mean of the predicted cost stayed at zero. The actual trajectory was in agreement with the prediction.

Besides the heuristic for increasing the prediction horizon, the entire learning procedure for finding a good policy was fully automatically. The heuristic employed was not crucial, but it made learning easier.

In case a trajectory did not agree with its predictive distribution of the trajectory, this "surprising" trajectory led to *learning*. When incorporating this information into the dynamics model, the model changed significantly in regions where the discrepancies between predictions and true states could not be explained by the error bars.

The obtained policy was not optimal[27], but it solved the task fairly well, and the system found it automatically using less than 30 s of interaction.[28] In a successful rollout when following this policy, the controller typically brought the angle exactly upright and kept the cart approximately 1 mm left from its optimal position.

Table 3.2 summarizes the results for the learned zero-order-hold controller. The total interaction time with the system was of the order of magnitude of a minute. However, to effectively learn the cart-pole task, that is, increasing the prediction horizon did not lead to increased error bars and a predicted failure, only about 20 s–30 s interaction time was necessary. The subsequent interactions were used to collect more data to further improve the dynamics model and the policy; the controller swung the pendulum up more quickly and became more robust: After eight policy searches[29] (when the task was essentially solved), the failure rate was about 10% and declined to about 1.5% after four more policy searches and roughly twice as much interaction time. Since only states that could not be predicted well contributed much to an improvement of the dynamics model, most data in the last trajectories were essentially not very valuable. Occasional failures can be explained by encountering states where the policy was not good, for example

[27] We sometimes found policies that could swing up quicker.

[28] The task could be considered solved after eight policy searches since the error bars of the predicted immediate cost vanished and the rollouts confirmed the prediction.

[29] Here, eight policy searches also correspond to eight trials when we include the random initial trial.

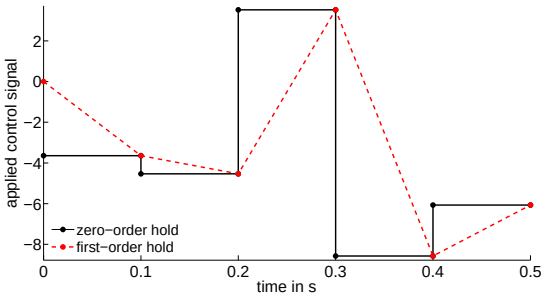


Figure 3.17: Zero-order-hold control (black, solid) and first-order-hold control (red, dashed). The x-axis shows the time in seconds, the y-axis shows the applied control signal. The control decision can be changed every $\Delta_t = 0.1\,\text{s}$. Zero-order-hold control applies a constant control signal for Δ_t to the system. First-order-hold control linearly interpolates between control decisions at time t and $t+1$ according to equation (3.39).

when the states were relatively far away from the states in the training set, which contained many states along a predicted optimal distribution over trajectories.

First-order-hold Control

Thus far, we considered control signals being constantly applied to the system for a time interval of Δ_t (zero-order hold). In many practical applications including robotics, this control method requires robustness of a physical system and the motor that applies the control signal: Instantaneously switching from a large positive control signal to a large negative control signal can accelerate attrition, for example. One way to increase the lifetime of the system is to implement a continuous-time control signal $\mathbf{u}(\tau)$ (see equation (3.34)) that smoothly interpolates between the control decisions $\mathbf{u}_{t-1}$ and $\mathbf{u}_t$ at time steps $t-1$ and t, respectively.

Suppose the control signal is piecewise linear. At each discrete time step, the controller decides on a new signal to be applied to the system. Instead of constantly applying it for Δ_t, the control signal interpolates linearly between the previous control decision $\pi(\mathbf{x}_{t-1}) = \mathbf{u}_{t-1}$ and the new control decision $\pi(\mathbf{x}_t) = \mathbf{u}_t$ according to

$$\mathbf{u}(\tau) = (1-\tau)\underbrace{\pi(\mathbf{x}_{t-1})}_{=\mathbf{u}_{t-1}} + \tau\underbrace{\pi(\mathbf{x}_t)}_{=\mathbf{u}_t}, \quad \tau \in [0, \Delta_t]\,. \tag{3.39}$$

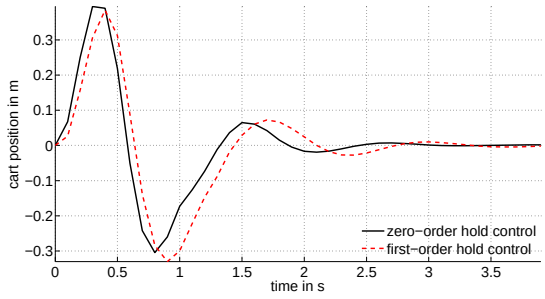
Here, $\mathbf{u}(\tau)$ is a continuous-time control signal and implements a *first-order-hold control*. The subscript t denotes discrete time. Figure 3.17 sketches the difference between zero-order hold and first-order hold. The control decision is changed every $\Delta_t = 0.1\,\text{s}$. The smoothing effect of first-order hold becomes apparent after 0.3 s: Instead of instantaneously switching from a large positive to a large negative control signal, first-order hold linearly interpolates between these values over a time span that corresponds to the sampling interval length Δ_t. It takes Δ_t for the first-order-hold signal to achieve the full effect of the control decision $\mathbf{u}_t$, whereas the zero-order-hold signal instantaneously switches to the new control signal. The smoothing effect of first-order hold diminishes in the limit $\Delta_t \to 0$.

We implemented the first-order-hold control using state augmentation. More precisely, the state was augmented by the previous control decision. With $\mathbf{u}_{-1} := \mathbf{0}$ we defined the augmented state at time step t

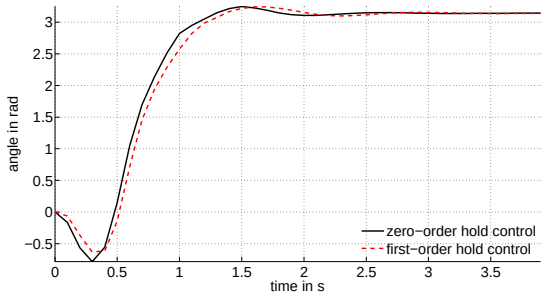
$$\mathbf{x}_t^{\text{aug}} := \begin{bmatrix} \mathbf{x}_t^\top & \mathbf{u}_{t-1}^\top \end{bmatrix}^\top,\ t \geq 0\,.$$

To learn a first-order-hold controller for the cart-pole problem, the same parameter setup as for the zero-order-hold controller was used (see Appendix C.1). In particular, the same sampling frequency $1/\Delta_t$ was chosen. We followed Algorithm 3 for the evaluation of a first-order-hold controller.

In Figure 3.18, typical state trajectories are shown, which are based on controllers applying zero-order-hold control and first-order-hold control starting from the same initial state $\mathbf{x}_0 = \mathbf{0}$. The first-order-hold controller induced a delay when solving the cart-pole task. This delay can be seen particularly well in the position of the cart. To stabilize the pendulum around equilibrium, the first-order-hold controller caused longer oscillations in both state variables than the zero-order-hold controller.



(a) Trajectory of the position of the cart.



(b) Trajectory of the angle of the pendulum.

Figure 3.18: Rollouts of four seconds for the cart position and the angle of the pendulum when applying zero-order-hold control and first-order-hold control. The delay induced by the first-order hold control can be observed in both state variables.

Table 3.3: Experimental results: cart-pole with first-order-hold control.

interaction time	57.8 s
task learned (negligible error bars)	after 17.2 s (5 trials)
failure rate ($d > l$)	7.6%
success rate ($d \leq 3\,\text{cm}$)	91.8%
$V^{\pi^*}(\mathbf{x}_0)$, $\boldsymbol{\Sigma}_0 = 10^{-2}\mathbf{I}$	10.5

Table 3.3 summarizes the results for the learned first-order-hold controller. Compared to the zero-order-hold controller, the expected long-term cost V^{π^*} was a bit higher due to the delay induced by the first-order hold. However, in this experiment, the first-order-hold controller learned the cart-pole task a bit quicker than the zero-order-hold controller.

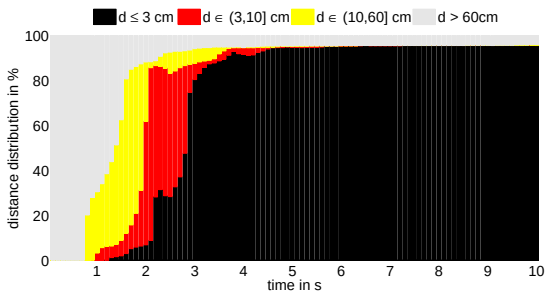
Position-independent Controller

Let us now discuss the case where the initial position of the cart is very uncertain. For this case, we set the marginal variance of the position of the cart to a value, such that about 95% of the possible initial positions of the cart were in the interval $[-4.5, 4.5]$ m. Due to the width of this interval, we informally call the controller for this problem "position independent". Besides the initial state uncertainty and the number of basis functions for the RBF controller, which we increased to 200, the basic setup was equivalent to the setup for the zero-order-hold controller earlier in this section. The full set of parameters are given in Appendix C.1

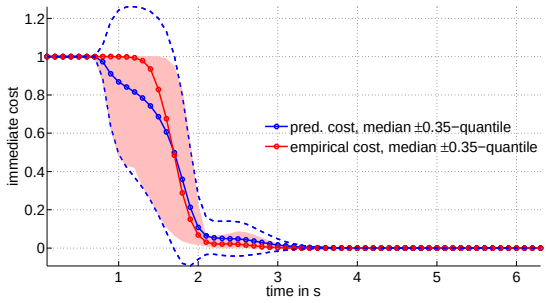
Directly performing the policy search with a large initial state uncertainty is a difficult optimization problem. To simplify this step, we employed ideas from continuation methods (see the tutorial paper by Richter and DeCarlo (1983) for detailed information): Initially, a controller was learned for a fairly peaked initial state distribution with $\boldsymbol{\Sigma}_0 = 10^{-2}\mathbf{I}$. When success was predicted[30], the initial state distribution was broadened and learning was continued. The found policy for the narrower initial state distribution served as the parameter initialization of the policy to be learned for the broadened initial state distribution. This"problem shaping" is typical for a continuation method and simplified the policy search, that is, the optimization problem.

For the evaluation, we followed the steps described in Algorithm 3. Figure 3.19(a) shows a histogram of the empirical distribution of the distance d to the target state. Following the learned policy π^* 1,000 rollouts were started from random states sampled independently from $p(\mathbf{x}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$. The figure is the position-independent counterpart of Figure 3.14. Compared to Figure 3.14, more time was required to solve the task, that is, when the black bars level out: Sometimes the cart had to drive a long way to the location from where the pendulum is swung up. We call this location the "swing-up location".

[30] To predict success, we employed the `task_learned` function in Algorithm 2 as a heuristic.



(a) Histogram of the distances d of the tip of the pendulum to the target. After a fairly long transient phase due to the widespread initial positions of the cart, after a transient phase, the controller could either solve the cart-pole problem very well (height of the black bar) or it could not solve it at all (height of the gray bar).



(b) Medians and quantiles of the predictive immediate cost distribution (blue) and the empirical immediate cost distribution (red). The fairly large uncertainty between $t = 0.7\,\mathrm{s}$ and $t = 2\,\mathrm{s}$ is due to the potential offset of the cart, which led to different arrival times at the "swing-up location": Typically, the cart drove to a particular location from where the pendulum was swung up.

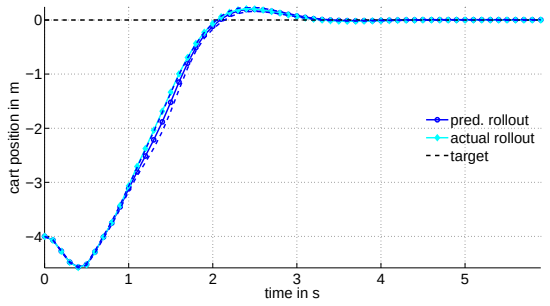
Figure 3.19: Cost distributions using the position-independent controller after 1,000 rollouts.

The error rate at the end of the prediction horizon was 4.4% (compared to 1.5% for the smaller initial distribution). The higher failure rate can be explained by the fact the wide initial state distribution $p(\mathbf{x}_0)$ was not sufficiently well covered by the trajectories in the training set. In both Figure 3.14 and Figure 3.19(a), in most rollouts, the controller brought the tip of the pendulum closer than 10 cm to the target within 1 s, that is from $t = 1\,\mathrm{s}$ to $t = 2\,\mathrm{s}$ (if we add the height of the black bars to the height of the red bars in both figures). Hence, the position-independent controller could solve the task fairly well within the first two seconds, but required more time to stabilize the pendulum. Like for the smaller initial distribution, at the end of each rollout, the controller either brought the system close to the target, or it failed completely (see the constant height of the gray bars). The position-independent controller took about 4 s to solve the task reliably. Two seconds of these four seconds were needed to stabilize the pendulum (from $t = 2\,\mathrm{s}$ to $t = 4\,\mathrm{s}$) around the target state (red bars turn into black bars), which is about 1 s longer than in Figure 3.14. The longer stabilization period can be explained by the higher uncertainty in the system. Figure 3.19(b) supports this statement: Figure 3.19(b) shows the α-quantile and the $(1-\alpha)$-quantile, $\alpha = 0.15$, of approximate Gaussian distributions[31] of the predicted immediate costs $c(\mathbf{x}_t)$, $t = 0\,\mathrm{s}, \ldots, 6.3\,\mathrm{s} = T_{\mathrm{max}}$, after the last policy search (blue) and the median and the quantiles of the corresponding empirical cost distribution (red) after 1,000 rollouts. The medians are described by the solid lines. Between $t = 0.7\,\mathrm{s}$ and $t = 2\,\mathrm{s}$ both the predictive distribution and the empirical distribution are very broad. This effect is caused by the almost arbitrary initial position of the cart: When the position of the cart was close to zero, the controller implemented a policy that resembled the policy found for a small initial distribution (see Figure 3.15). Otherwise the cart reached the low-cost region with a "delay". According to Figure 3.19(b), this delay could be up to 1 s. However, after about 3.5 s, the quantiles of both the predicted distribution and the empirical distribution of the immediate cost converge toward the medians, and the medians are almost identically zero.

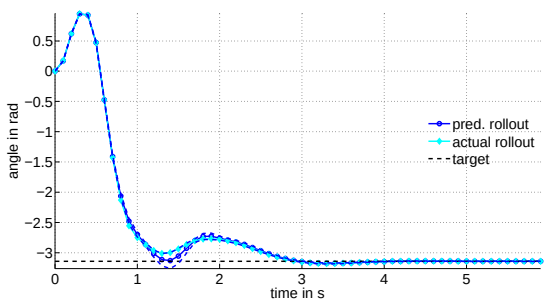
In a typical successful rollout (when following π^*), the controller swung the pendulum up and balanced it exactly in the inverted position while the cart had an offset of 2 mm from the optimal cart position just below the cross in Figure 3.13.

Figure 3.20 shows predictions of the position of the cart and the pendulum angle starting from $\mathbf{x}_0 = [-4, 0, 0, 0]^\top$. We passed the initial state distribution $p(\mathbf{x}_0)$ and the learned controller implementing π^* to the model of the transition dynamics. The model predicted the state 6 s ahead without any evidence from measurements to refine the predicted state distribution. However, the learned policy π^* was incorporated explicitly into these predictions following the approximate inference algorithm described in Section 3.4. The figure illustrates that the predictive dynamics model was fairly accurate and not overconfident since the true trajectories were within the predicted 95% confidence bounds. The error bars in the angle grow slightly around $t = 1.3\,\mathrm{s}$ when the controller decelerated the pendulum to stabilize

[31] The Gaussian approximation of the predicted immediate cost distribution was only used for the evaluation, but not during learning.



(a) Predicted position of the cart and true latent position. The cart is 4 m away from the target. It took about 2 s to move to the origin. After a small overshoot, which was required to stabilize the pendulum, the cart stayed close to the origin.



(b) Predicted angle of the pendulum and true latent angle. After swinging the pendulum up, the controller balanced the pendulum.

Figure 3.20: Predictions (blue) of the position of the cart and the angle of the pendulum when the position of the cart was far away from the target. The true trajectories (cyan) were within the 95% confidence interval of the predicted trajectories (blue). Note that even in predictions without any state update the uncertainty decreased (see panel (b) between $t \in [1, 2]$ s.) since the system was being controlled.

Table 3.4: Experimental results: position-independent controller (zero-order-hold control).

interaction time	53.7 s
task learned (negligible error bars)	after 17.2 s (5 trials)
failure rate ($d > l$)	4.4%
success rate ($d \leq 3$ cm)	95.5%
$V^{\pi^*}(\mathbf{x}_0)$, $\mathbf{\Sigma}_0 = \text{diag}([5,\ 10^{-2},\ 10^{-2},\ 10^{-2}])$	15.8

it. Note that the initial position of the cart was 4 m away from its optimal target position. Table 3.4 summarizes the results for the learned position-independent controller. Compared to the results of the "standard" setup (see Table 3.2), the failure rate and the expected long-term cost are higher. In particular, the higher value of V^π originates from the high uncertainty of the initial position of the cart. Interaction time and the speed of learning do not differ much compared to the results of the standard setup.

When the position-independent controller was applied to the simpler problem with $\mathbf{\Sigma}_0 = 10^{-2}\mathbf{I}$, we obtained a failure rate of 0% after 1,000 trials compared to 1.5% when we directly learned a controller for this tight distribution (see Table 3.2). The increased robustness is due to the fact that the tails of the tighter distribution with $\mathbf{\Sigma}_0 = 10^{-2}\mathbf{I}$ were better covered by the position-independent controller since during learning the initial states were drawn from the wider distribution. The swing up was not performed as quickly as shown in Figure 3.15, though.

Hardware Experiment

We applied our proposed learning framework based on a probabilistic dynamics model to a hardware realization of the cart-pole system. The setup of the apparatus is shown in Figure 3.21. The apparatus consists of a pendulum attached to a cart, which can be pulled up and down a (finite) track by a wire attached to a torque motor.

The zero-order-hold control signal $u(\tau)$ was the voltage to the power amplifier, which then produced a current in the torque motor. The observations were the position of the cart, the velocity of the cart, the angle of the pendulum, the angular velocity of the pendulum, and the motor current. The values returned for the measured system state were very accurate, that is almost noise free, such that we considered them exact. The graphical model in Figure 3.1 was employed although it is only approximately correct (see Section 3.8.2 for a more detailed discussion).

Table 3.5 reports some physical parameters of the cart-pole system in Figure 3.21. Note that none of

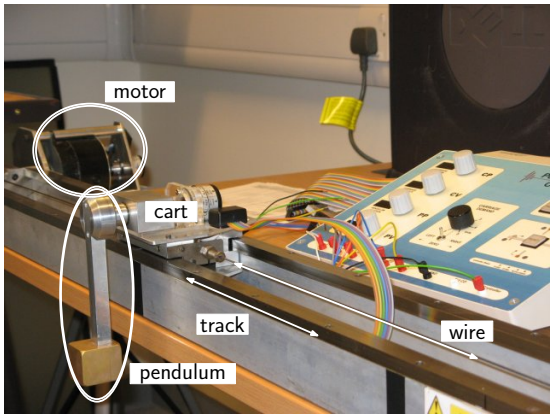


Figure 3.21: Hardware setup of the cart-pole system.

Table 3.5: Parameters of the cart-pole system (hardware).

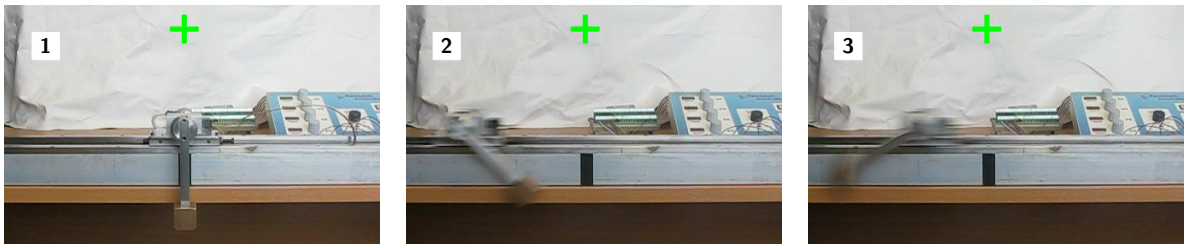
length of the pendulum	$l = 0.125\,\mathrm{m}$
mass of the pendulum	$m = 0.325\,\mathrm{kg}$
mass of the cart	$M = 0.7\,\mathrm{kg}$

the parameters in Table 3.5 was directly required to apply our learning framework. However, we used the length of the pendulum for heuristics to determine the width $1/a$ of the cost function in equation (3.37) and the sampling frequency $1/\Delta_t$. The width of the cost function encodes what states were considered "far" from the target. If a state $\mathbf{x}$ was further away from the target than twice the width $1/a$ of the cost function in equation (3.37), the state was considered "far away". The pendulum length was also used to find the characteristic frequency of the system.[32] Since $l = 125\,\mathrm{mm}$, we set the sampling frequency to $10\,\mathrm{Hz}$, which is about seven times faster than the characteristic frequency of the system. Furthermore, we chose the cost function in equation (3.18) with $1/a \approx 0.07\,\mathrm{m}$, such that the cost incurred did not substantially differ from unity if the distance between the pendulum tip and the target state was greater than the length of the pendulum.

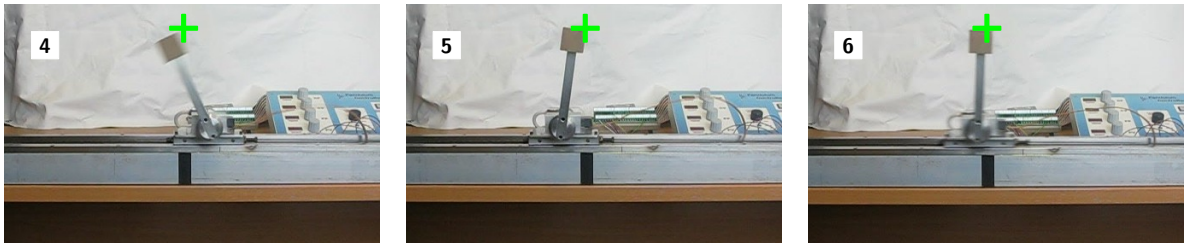
Unlike classical control methods, our algorithm learned a probabilistic non-parametric model of the system dynamics in equation (3.1) and a good controller from data. It was therefore not necessary to provide a complicated parametric description of the transition dynamics that might have included parameters, such as friction coefficients, motor constants, and delays. The torque motor limits implied force constraints of $u \in [-10, 10]\,\mathrm{N}$. The length of each trial was constant and set to $T_{\mathrm{init}} = 2.5\,\mathrm{s} = T_{\mathrm{max}}$, that is, we did not use the heuristic to stepwise increase the prediction horizon.

Following the automatic procedure described in Algorithm 1, the learning algorithm was initialized with two trials of length $T = 2.5\,\mathrm{s}$ each, where actions (horizontal forces to the cart) randomly drawn from $\mathcal{U}[-10, 10]\,N$ were applied. The five seconds of data collected in these trials were used to train an initial probabilistic dynamics model. The RL algorithm used this model to simulate the cart-pole system internally and to optimize the parameters of the RBF controller (line 6 in Algorithm 1). In the third trial, which was the first non-random trial, the RBF policy was applied to the system. The controller managed to keep the cart in the middle of the track, but the pendulum did not go beyond horizontal—the system never experienced states before with the pendulum being above horizontal. In the subsequent model update (line 5), these observations were taken into account. Due to the incorporated new experience, the uncertainty in the predictions decreased and a good policy for the updated model was found. Then, the new policy was applied to the system for another $2.5\,\mathrm{s}$, which led to the fourth trial where the controller swung the pendulum up. The pendulum drastically overshot, but for the first time states close to the target state were encountered. In the iteration, these observations were taken into account by the updated dynamics model and the corresponding controller was learned. In the fifth trial, the controller learned

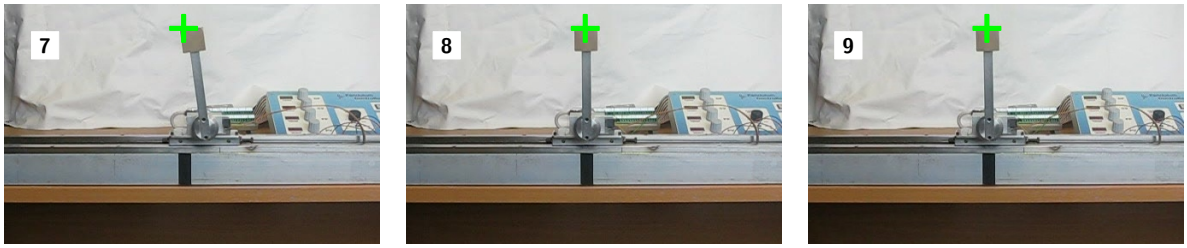
[32] The swing period of the pendulum is approximately $2\pi\sqrt{l/g} \approx 0.7\,\mathrm{s}$, where g is the acceleration of gravity and l is the length of the pendulum.



(a) Test trajectory, $t = 0.000\,\mathrm{s}$. (b) Test trajectory, $t = 0.300\,\mathrm{s}$. (c) Test trajectory, $t = 0.466\,\mathrm{s}$.



(d) Test trajectory, $t = 0.700\,\mathrm{s}$. (e) Test trajectory, $t = 1.100\,\mathrm{s}$. (f) Test trajectory, $t = 1.300\,\mathrm{s}$.



(g) Test trajectory, $t = 1.500\,\mathrm{s}$. (h) Test trajectory, $t = 3.666\,\mathrm{s}$. (i) Test trajectory, $t = 7.400\,\mathrm{s}$.

Figure 3.22: Inverted pendulum in hardware; snapshots of a controlled trajectory after having learned the task. The pendulum was swung up and balanced in the inverted position close to the target state (green cross). To solve this task, our algorithm required only 17.5 s of interaction with the physical system.

to reduce the angular velocity close to the target since falling over led to high expected cost. After two more trials, the learned controller solved the cart-pole task based on a total of 17.5 s experience only. Furthermore, the controller and the corresponding dynamics model were found fully automatically by simply following the steps in Algorithm 1. Figure 3.22 shows snapshots of a test trajectory of 20 s length. A video showing the entire learning process can be found at `http://mlg.eng.cam.ac.uk/marc/`.[33]

Our learning algorithm is very general and worked immediately when we applied it to the hardware problem. Since we could derive all required parameters (the width of the cost function and the time sampling frequency) from the length of the pendulum, no parameter tuning was necessary.

Table 3.6 summarizes the results of the hardware experiment. Although we used two random initial trials, only seven trials in total were required to learn the task. The interaction time required to solve the task was far less than a minute, an unprecedented learning efficiency for this kind of problem. Note that only system identification in a classical sense requires more data from interaction.

[33]I thank James N. Ingram and Ian S. Howard for technical support. I am grateful to Jan Maciejowski and the Control Group at the University of Cambridge for providing the hardware equipment.

Table 3.6: Experimental results: hardware experiment.

interaction time	17.5 s
task learned (negligible error bars)	after 17.5 s (7 trials)
$V^{\pi^*}(\mathbf{x}_0)$, $\mathbf{\Sigma}_0 = 10^{-2}\mathbf{I}$	8.2

Importance of the RL Components

As described in Figure 3.4, finding a good policy requires the successful interplay of the three components: the cost function, the dynamics model, and the RL algorithm. To analyze the importance of either of the components in Figure 3.4, in the following, we substitute classical components for the probabilistic model, the saturating cost function, and the approximate inference algorithm. We provide some evidence of the relevance of the probabilistic dynamics model, the saturating cost function, and the approximate inference algorithm using moment matching.

Deterministic model. Let us start with replacing the probabilistic dynamics model by a deterministic model while keeping the cost function and the learning algorithm the same.

The deterministic model employed was a "non-parametric RBF network": Like in the previous sections, we trained a GP model using the collected experience in form of tuples (state, action, successor state). However, we set the model uncertainty to zero. The resulting model was an RBF network (equivalent to the prior mean function of the GP), where the number of basis functions increased with the size of the data set. Although this model is deterministic, it could still be used to propagate (state) uncertainty as detailed in Section 3.4.2 in the context of the policy model. Hence, the only difference to the probabilistic GP model was that the "non-parametric RBF network" employed was degenerate, that is, the predictive variances vanished far away from the data.

The degeneracy of the model was crucial and led to a failure in solving the cart-pole problem: The initial training set for the dynamics model was essentially random and did not contain states close to the target state. The model was overconfident when the predicted states left the region of the training set. Since the model's extrapolation eventually fell back to the mean function (with zero variance), the model predicted no change in state (independent of the control applied) when moving away from the data.[34] The model never "saw" states close to the target and finally decided on a policy that kept the pendulum in the downward position.

We found two ways of making the deterministic model work. Either basis functions were placed along trajectories that led to the target or a constant noise term was added to the predictive variances. Both ways of "healing" the deterministic model were successful to learn the cart-pole task although learning the task took a bit longer. The problem with the first approach is that a path to the target had to be known in advance, that is, expert knowledge was required.[35] Therefore, this option defeats our purpose of finding solutions in the absence of expert knowledge. The second option of adding a constant noise term essentially shapes the deterministic model back toward a probabilistic model: The noise is interpreted as a constant term that emulates the model uncertainty. A problem with the constant noise term is that the right order of magnitude has to be found and depends on the task. However, when we simply made the noise term dependent on the variance of the function value, we were almost back to the probabilistic GP model since the GP learns exactly this relationship from data to set the signal variance (together with the other hyper-parameters).

From these experiments, we conclude that the probabilistic dynamics model in Figure 3.4 is crucial for learning motor control tasks in the absence of expert knowledge.

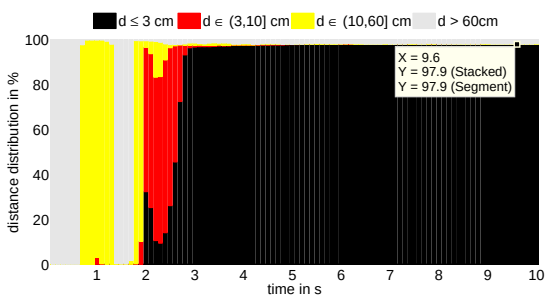
Quadratic cost function. In our next experiment, we chose the quadratic cost function instead of the saturating cost function. We kept the probabilistic dynamics model and the indirect policy search RL algorithm. This means, the second component of the three essential components in Figure 3.4 was replaced.

We considered the quadratic immediate cost function in equation (3.29), where d is the Euclidean distance (in meters) from the tip of the pendulum to the target position and a^2 is a scalar parameter controlling the width of the cost parabola. The squared distance is given in equation (3.35) and in equation (3.38). For the evaluation, we followed the steps in Algorithm 3.

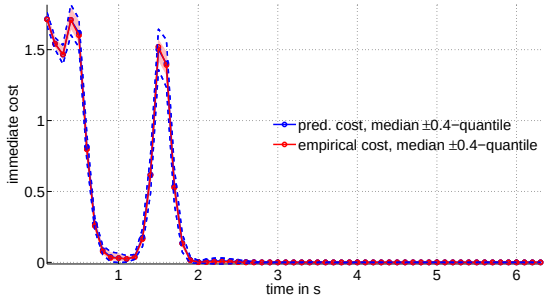
Figure 3.23 shows a distance histogram and the quantiles of the predicted and empirical cost distributions. Let us first consider Figure 3.23(a): Initially, the controller attempted to bring the pendulum closer to the target to avoid the high-cost region encoded by the gray bars. A region of lower cost (yellow bars) was reached for many trajectories at 0.8 s. After that, the pendulum tip fell over (gray bars appear again)

[34] Note that the model was trained on differences $\mathbf{x}_{t+1} - \mathbf{x}_t$, see equation (3.3).

[35] To generate this expert knowledge, we used a working policy (learned by using a probabilistic dynamics model) to generate the initial training set (instead of random actions).



(a) Histogram of the distances d of the tip of the pendulum to the target of 1,000 rollouts. The x-axis shows the time in seconds, the colors encode distances to the target. The heights of the bars correspond to their respective percentages in the 1,000 rollouts.



(b) Quantiles of the predictive immediate cost distribution (blue) and the empirical immediate cost distribution (red). The x-axis shows the time in seconds, the y-axis shows the immediate cost.

Figure 3.23: Distance histogram in panel 3.23(a) and quantiles of cost distributions in panel 3.23(b). The controller was learned using the quadratic immediate cost in equation (3.29).

Table 3.7: Experimental results: quadratic-cost controller (zero-order hold).

interaction time	46.1 s
task learned (negligible error bars)	after 22.2 s (6 trials)
failure rate ($d > l$)	1.9%
success rate ($d \leq 3\,\text{cm}$)	97.8%
$V^{\pi^*}(\mathbf{x}_0)$, $\boldsymbol{\Sigma}_0 = 10^{-2}\mathbf{I}$	13.8 (quadratic cost)

and came again close at $t = 1.8\,\text{s}$. Then, the pendulum fell over for the second time and finally swung up and balanced after approximately 3 s. Instantaneously, the heights of the gray bars shrink to a small value. While ensuring that the pendulum did not fall over, the tip of the pendulum was slowly brought closer to the target. Compared to the histograms of the distances to the target shown in Figures 3.14 and 3.19(a), the histogram in Figure 3.23(a) seems "delayed" by about a second. The trough of the black bar at approximately 2.3 s can be explained by the computation of the expected cost in equation (3.30): Through the trace operator, the covariance of the state distribution is an additive linear contribution (scaled by $\mathbf{T}^{-1}$) to the expected cost in equation (3.30). By contrast, the distance of the mean of the state to the target contributes quadratically to the expected cost (scaled by $\mathbf{T}^{-1}$). Hence, when the mean of the predictive state distribution came close to the target, say, $d \leq 0.1\,\text{m}$, the main contribution to the expected cost was the uncertainty of the predicted state. Thus, it could be suboptimal for the controller to bring the pendulum from the red region to the black region when the uncertainty increases at the same time.

Figure 3.23(b) shows the median and the 0.1-quantiles and the 0.9-quantiles of both the predicted immediate cost and the empirical immediate cost at time t for $t = \Delta_t = 0.1\,\text{s}$ to $t = 6.3\,\text{s} = T_{\max}$. The high-cost regions in the plot are within the first 3 s. The expected long-term cost based on the quadratic cost function is fairly sensitive to predicted state distribution within this time span. By contrast, the saturating cost function in equation (3.18) does not much distinguish between states where the tip of the pendulum is further away from the target state than one meter: The cost is unity with very small variance.

Table 3.7 summarizes the results of the learning algorithm when using the quadratic cost function in equation (3.29). No exploration was employed, that is, the exploration parameter b in equation (3.25) equals zero. Compared to the saturating cost function, learning with the quadratic cost function was often slower since the quadratic cost function payed much attention to essentially minor details of the state distribution when the state was far away from the target. Our experience is that if a solution was found when using the quadratic cost function, it was often worse than the solution with a saturating

Algorithm 4 Policy evaluation with PEGASUS

for $i = 1$ to S **do** ▷ for all scenarios
  sample $\mathbf{x}_0^{(i)}$ from $p(\mathbf{x}_0)$ ▷ sample start state from initial distribution
  load fixed random numbers $q_{0i}, \dots, q_{Ti}$
  for $t = 0$ to T **do** ▷ for all time steps
    compute $p\big(\mathbf{x}_{t+1}^{(i)} | \mathbf{x}_t^{(i)}, \pi^*\big(\mathbf{x}_t^{(i)}\big)\big)$ ▷ distribution of successor state
    determine $\mathbf{x}_{t+1}^{(i)} | q_{ti}$ ▷ "sample" successor state deterministically
  end for
  $V^\pi(\mathbf{x}_0) = \frac{1}{S} \sum_{i=1}^{S} V^\pi(\mathbf{x}_0^{(i)})$ ▷ overall value function
end for

cost function. It also seemed that the optimization using the quadratic cost suffered more severely from local optima. Better solutions existed, though: When plugging in the controller that was found with the saturating cost, the expected sum of immediate quadratic costs could halve.

Summarizing, the saturating cost function was not crucial but helpful for the success of the learning framework: The saturating immediate cost function typically led to faster learning and better solutions than the quadratic cost function.

Approximate inference algorithm. In our next experiment, we used the PEGASUS algorithm for Monte Carlo policy evaluation instead of the proposed approximate inference algorithm based on moment matching (see Section 3.4). We kept the probabilistic GP dynamics model and the saturating cost function fixed. This means, the third component of the general model-based setup depicted in Figure 3.4 was replaced.

PEGASUS is a sampling-based policy search algorithm proposed by Ng and Jordan (2000). The central idea of PEGASUS is to build a deterministic simulator to generate sample trajectories from an initial state distribution. By essentially fixing the random seed, even a stochastic model can be converted into a deterministic simulative model by state space augmentation.

The PEGASUS algorithm assumes that a generative model of the transition dynamics is given. If the model is probabilistic, the successor state $\mathbf{x}_t$ of a current state-action $(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ pair is always given by a probability distribution $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_{t-1})$. PEGASUS addresses the problem of sampling trajectories in this setup efficiently. Typically, the successor state $\mathbf{x}_t$ is sampled from $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ and propagated forward and determined by an internal random number generator. This standard sampling procedure is inefficient and cannot be used for gradient-based policy search methods using a small number of samples only. The key idea in PEGASUS is to draw a sample from the augmented state distribution $\tilde{p}(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_{t-1}, q)$, where q is a random number provided *externally*. If q is given externally, the distribution $\tilde{p}(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_{t-1}, q)$ collapses to a deterministic function of q. Therefore, a *deterministic simulative model* is obtained that is very powerful as detailed by Ng and Jordan (2000). For each rollout during the policy search, the same random numbers are used.

Since PEGASUS solely requires a generative model for the transition dynamics, we used our probabilistic GP model for this purpose and performed the policy search with PEGASUS. To optimize the policy parameters for an initial distribution $p(\mathbf{x}_0)$, we sampled S start states $\mathbf{x}_0^{(i)}$, so called *scenarios*, from which we started the sample trajectories determined by PEGASUS. The value function $V^\pi(\mathbf{x}_0)$ in equation (3.2) is then approximated by a Monte Carlo sample average over $V^\pi(\mathbf{x}_0^{(i)})$. The derivatives of the value function with respect to the policy parameters (required for the policy search) can be computed analytically for each scenario. Algorithm 4 summarizes the policy evaluation step in Algorithm 8 with PEGASUS. Note that the predictive state distribution $p\big(\mathbf{x}_{t+1}^{(i)} | \mathbf{x}_t^{(i)}, \pi^*\big(\mathbf{x}_t^{(i)}\big)\big)$ is a standard GP predictive distribution where $\mathbf{x}_t^{(i)}$ is given *deterministically*, which saves computations compared to our approximate inference algorithm that requires predictions with uncertain inputs. However, PEGASUS performs these computations for S scenarios, which can easily become computationally cumbersome.

It turned out to be difficult to learn a good policy for the cart-pole task using PEGASUS for the policy evaluation. The combination of PEGASUS with the saturating cost function and the gradient-based policy search using a deterministic gradient-based optimizer often led to slow learning if a good policy was found at all. In our experiments, we used between 10 and 100 scenarios, which might not be enough to compute V^π sufficiently well. From a computational perspective, we were not able to sample more scenarios, since

Table 3.8: Some cart-pole results in the literature (using no expert knowledge).

citation	interaction	swing up	balancing	dyn. model
Kimura and Kobayashi (1999)	144,000 s	yes	yes	none
Doya (2000)	52,000 s	yes	yes	none
Doya (2000)	16,000 s	yes	yes	RBF
Coulom (2002)	500,000 s	yes	yes	none
Wawrzynski and Pacut (2004)	900 s	yes	yes	none
Riedmiller (2005)	289 s–576 s	no	yes	none
Raiko and Tornio (2005, 2009)	125 s–150 s	yes	yes	MLP
Deisenroth and Rasmussen (2009)	17.5 s	yes	yes	GP

the learning algorithm using PEGASUS and 25 policy searches took about a month.[36] Sometimes, we could learn a policy that led the pendulum close to the target point, but it was not yet able to stabilize. However, we believe that PEGASUS would have found a good solution with more policy searches, which would have taken another couple of weeks.[37]

A difference between PEGASUS and our approximate inference algorithm using moment matching is that PEGASUS does not approximate the distribution of the state $\mathbf{x}_t$ by a Gaussian. Instead, the distribution of a state at time t is represented by a set of samples. This representation might be more accurate (at least in the case of many scenarios), but it does not necessarily lead to quicker learning. Using the sample-based representation, PEGASUS does not average over unseen states, which is possible when explicitly averaging according to a (Gaussian) state distribution $p(\mathbf{x}_t)$. This fact might also be a reason why exploration with PEGASUS can be difficult—and exploration is clearly required in our setup since we do not rely on expert knowledge.

Summarizing, although not tested heavily, the PEGASUS algorithm seemed not a very successful approximate inference algorithm in the context of the cart-pole problem. More efficient code would allow for more possible scenarios. However, we are doubtful whether PEGASUS eventually can reach the efficiency of our approximate inference algorithm.

Results in the Literature

Table 3.8 lists some successes in learning the cart-pole task in the absence of expert knowledge. In all papers it was assumed that all state variables can be measured. Although the setups of the cart-pole task across the papers slightly differ, an impression of the improvements over the last decade can be obtained. The different approaches are distinguished by interaction time, the type of the problem (balancing only or swing up plus balancing), and whether a dynamics model is learned or not. Kimura and Kobayashi (1999) employed a hierarchical RL approach composed of local linear controllers and Q-learning to learn both the swing up and balancing. Doya (2000) applied both model-free and model-based algorithms to learn the cart-pole task. The value function was modeled by an RBF network. If the dynamics model was learned (using a different RBF network), Doya (2000) showed that the resulting algorithm was more efficient than the model-free learning algorithm. Coulom (2002) presented a model-free TD-based algorithm that approximates the value function by a multilayer perceptron (MLP). Although the method looks rather inefficient compared to the work by Doya (2000), better value function models can be obtained. Wawrzynski and Pacut (2004) used multilayer perceptrons to approximate the value function and the randomized policy in an actor-critic context. Riedmiller (2005) applied the *Neural Fitted Q Iteration* (NFQ) to the cart-pole balancing problem without swing up. NFQ is a generalization of Q-learning by Watkins (1989), where the Q-state-action value function is modeled by an MLP. The range

[36] The data sets after 25 policy searches were fairly large, which slowed the learning algorithm down. We did not switch to sparse GP approximations to exclude a possible source of errors.

[37] In a personal communication, Andrew Ng suggested a stochastic gradient descent method using a single scenario per descent step but a different scenario for each gradient step. We have not yet investigated this.

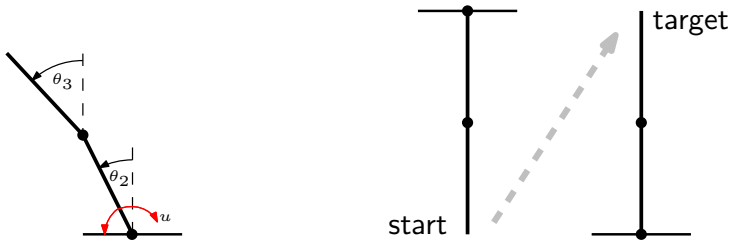


Figure 3.24: Pendubot system. The Pendubot is an under-actuated two-link arm, where the inner link can exert torque. The goal is to swing up both links and to balance them in the inverted position.

of interaction times in Table 3.8 depends on the quality of the Q-function approximation.[38] Raiko and Tornio (2005, 2009) employed a model-based learning algorithm to solve the cart-pole task. The system model was learned using the *Nonlinear dynamical factor analysis* (NDFA) proposed by Valpola and Karhunen (2002). Raiko and Tornio (2009) used NDFA for system identification in partially observable Markov decision processes (POMDPs), where MLPs were used to model both the system equation and the measurement equation.[39] The results in Table 3.8 are reported for three different control strategies, direct control, optimistic inference control, and nonlinear model predictive control, which led to different interaction times required to solve the task. In this report, we showed that our learning framework requires 17.5 s only to learn the cart-pole task. This means, we require at least one order of magnitude less interaction than any other RL algorithm we found in the literature.

3.7.2 Pendubot

We applied our RL framework to learning a dynamics model and a controller for the Pendubot task depicted in Figure 3.24. The Pendubot is a two-link, under-actuated robotic arm and was introduced by Spong and Block (1995). The inner joint (attached to the ground) exerts a torque u, but the outer joint cannot. The system has four continuous state variables: two joint angles and two joint angular velocities. The angles of the joints, θ_2 and θ_3, are measured anti-clockwise from the upright position. The dynamics of the Pendubot are derived from first principles in Appendix B.2.

The state of the system was given by $\mathbf{x} = [\dot{\theta}_2, \dot{\theta}_3, \theta_2, \theta_3]^\top$, where θ_2, θ_3 are the angles of the inner pendulum and the outer pendulum, respectively (see Figure 3.24), and $\dot{\theta}_2, \dot{\theta}_3$ are the corresponding angular velocities. During simulation, the state was represented as

$$\mathbf{x} = \begin{bmatrix} \dot{\theta}_2 & \dot{\theta}_3 & \sin\theta_2 & \cos\theta_2 & \sin\theta_3 & \cos\theta_3 \end{bmatrix}^\top \in \mathbb{R}^6 \,.$$

Initially, the system was expected to be in a state, where both links hung down ($\theta_2 = \pi = \theta_3$). By applying a torque to the inner joint, the objective was to swing both links up and to balance them in the inverted position around the target state ($\theta_2 = 2k_2\pi$, $theta_3 = 2k_3\pi$, where $k_2, k_3 \in \mathbb{Z}$) as depicted in the right panel of Figure 3.24. The Pendubot system is a chaotic and inherently unstable system. A linear controller is not capable to solve the Pendubot task. Furthermore, a myopic strategy does not lead to success either.

Typically, two controllers are employed to solve the Pendubot task, one to solve the swing up and a linear controller for the balancing (Spong and Block, 1995; Orlov et al., 2008). Unlike this engineered solution, the proposed RL framework *learned* a single nonlinear RBF controller to solve both subtasks.

The parameters used for the computer simulation are given in Appendix C.2. The chosen time discretization $\Delta_t = 0.075\,\mathrm{s}$ corresponds to a fairly slow sampling frequency of $13.\bar{3}\,\mathrm{Hz}$: For comparison, O'Flaherty et al. (2008) chose a sampling frequency of $2,000\,\mathrm{Hz}$.

[38] NFQ code is available online at `http://sourceforge.net/projects/clss/`.
[39] The system identification with NDFA goes beyond the scope of this report and addresses an important field of research.

Cost Function

Every $\Delta_t = 0.075\,\mathrm{s}$, the squared Euclidean distance

$$\begin{aligned} d(\mathbf{x}, \mathbf{x}_{\text{target}})^2 &= (-l_2 \sin\theta_2 - l_3 \sin\theta_3)^2 + (l_2 + l_3 - l_2 \cos\theta_2 - l_3 \cos\theta_3)^2 \\ &= l_2^2 + l_3^2 + (l_2 + l_3)^2 + 2 l_2 l_3 \sin\theta_2 \sin\theta_3 - 2(l_2 + l_3) l_2 \cos\theta_2 \\ &\quad - 2(l_2 + l_3) l_3 \cos\theta_3 + 2 l_2 l_3 \cos\theta_2 \cos\theta_3 \end{aligned} \tag{3.40}$$

from the tip of the outer pendulum to the target state was measured. Here, the lengths of the two pendulums are denoted by l_i, $i = 2, 3$.

Note that the distance d in equation (3.40) and, therefore, the cost function in equation (3.18), only depends on the sines and cosines of the angles θ_i. In particular, it does not depend on the angular velocities $\dot{\theta}_i$ and the torque u. An approximate Gaussian joint distribution $p(\mathbf{j}) = \mathcal{N}(\mathbf{m}, \mathbf{S})$ of the involved states

$$\mathbf{j} \coloneqq \begin{bmatrix} \sin\theta_2 & \cos\theta_2 & \sin\theta_3 & \cos\theta_3 \end{bmatrix}^\top \tag{3.41}$$

was computed using the results from Section A.1. The target vector in $\mathbf{j}$-space was $\mathbf{j}_{\text{target}} = [0, 1, 0, 1]^\top$. The matrix $\mathbf{T}^{-1}$ in equation (3.19) was given by

$$\mathbf{T}^{-1} \coloneqq a^2 \begin{bmatrix} l_2^2 & 0 & l_2 l_3 & 0 \\ 0 & l_2^2 & 0 & l_2 l_3 \\ l_2 l_3 & 0 & l_3^2 & 0 \\ 0 & l_2 l_3 & 0 & l_3^2 \end{bmatrix} = a^2 \mathbf{C}^\top \mathbf{C} \qquad \text{with} \quad \mathbf{C}^\top \coloneqq \begin{bmatrix} l_2 & 0 \\ 0 & l_2 \\ l_3 & 0 \\ 0 & l_3 \end{bmatrix}, \tag{3.42}$$

where $1/a$ controlled the width of the saturating immediate cost function in equation (3.18). Note that when multiplying $(\mathbf{j} - \mathbf{j}_{\text{target}})$ from the left and the right to $\mathbf{C}^\top \mathbf{C}$, the squared Euclidean distance d^2 in equation (3.40) is recovered. The saturating immediate cost was then given as

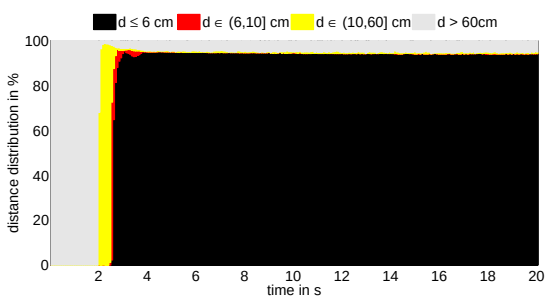
$$c(\mathbf{x}) = c(\mathbf{j}) = 1 - \exp\left(-\tfrac{1}{2}(\mathbf{j} - \mathbf{j}_{\text{target}})^\top \mathbf{T}^{-1} (\mathbf{j} - \mathbf{j}_{\text{target}})\right) \in [0, 1]\,. \tag{3.43}$$

The width $1/a = 0.5\,\mathrm{m}$ of the cost function in equation (3.42) was chosen, such that the immediate cost was about unity as long as the distance between the tip of the outer pendulum and the target state was greater than the length of both pendulums. Thus, the tip of the outer pendulum had to move above horizontal to reduce the immediate cost significantly from unity.

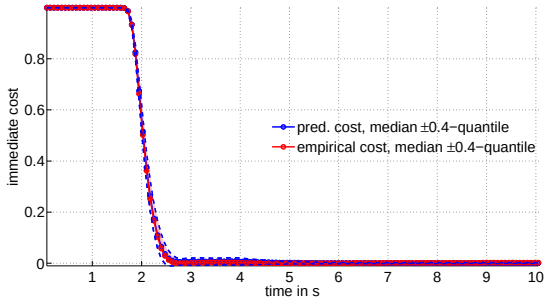
Zero-order-hold Control

By following the steps of Algorithm 2, our RL framework learned a zero-order-hold controller, where the control decision could be changed every $\Delta_t = 0.075\,\mathrm{s}$. When following the learned policy π^*, Figure 3.25(a) shows a histogram of the empirical distribution of the distance d from the tip of the outer pendulum to the inverted position based on 1,000 rollouts from start positions randomly sampled from $p(\mathbf{x}_0)$ (see Algorithm 3). It took about 2 s to leave the high-cost region represented by the gray bars. After about 2 s, the tip of the outer pendulum was closer to the target than its own length in most of the rollouts. In these cases, the tip of the outer pendulum was certainly above horizontal. After about 2.5 s, the tip of the outer pendulum came close to the target in the first rollouts, which is illustrated by the increasing black bars. After about 3 s the black bars "peak" meaning that at this time point the tip of the outer pendulum was close to the target in almost all trajectories. The decrease of the black bars and the increase of the red bars between 3.1 s and 3.5 s is due to a slight over-swing of the Pendubot. The RBF-controller essentially had to switch from swinging up to balancing. However, the pendulums typically did not fall over. After 3.5 s, the red bars vanish, and the black bars level out at 94%. Like for the cart-pole task (Figure 3.14), the controller either brought the system close to the target, or it failed completely.

Figure 3.25(b) shows the α-quantiles and the $1 - \alpha$-quantiles, $\alpha = 0.1$, of a Gaussian approximation of the distribution of the predicted immediate costs $c(\mathbf{x}_t)$, $t = 0\,\mathrm{s}, \ldots, 10\,\mathrm{s} = T_{\max}$ (using the controller implementing π^* after the last policy search), and the corresponding empirical cost distribution after 1,000 rollouts. The medians are described by the solid lines. The quantiles of the predicted cost (blue,



(a) Histogram of the distances d from the tip of the outer pendulum to the upright position of 1,000 rollouts. At the end of the horizon, the controller could either solve the problem very well (black bar) or it could not solve it at all, that is, $d > l_3$ (gray bar).



(b) Quantiles of the predictive immediate cost distribution (blue) and the empirical immediate cost distribution (red).

Figure 3.25: Cost distributions for the Pendubot task (zero-order-hold control).

dashed) and the empirical quantiles (red, shaded) are similar to each other. The quantiles of the predicted cost cover a larger area than the empirical quantiles due to the Gaussian representation of the immediate cost. The Pendubot required about 1.8 s for the immediate cost to fall below unity. After about 2.2 s, the Pendubot was balanced in the inverted position. The error bars of both the empirical immediate cost and the predicted immediate cost declined to close to zero for $t \geq 5$ s.

Figure 3.26 shows six examples of the predicted cost and the real cost during learning the Pendubot task. In Figure 3.26(a), after 23 trials, we see that the controller managed to bring the Pendubot close to the target state. This took about 2.2 s. After that, the error bars of the prediction increased. The prediction horizon was increased for the next policy search as shown in Figure 3.26(b). Here, the error bars increased when the time exceeds 4 s. It was predicted that the Pendubot could not be stabilized. The actual rollout shown in cyan, however, did not incur much cost at the end of the prediction horizon and was therefore surprising. The rollout was not explained well by the prediction, which led to learning as discussed in Section 3.7.1. In Figure 3.26(c), it was predicted (with high confidence) that the Pendubot could be stabilized, which was confirmed by the actual rollout. In Figures 3.26(d)–3.26(f), the prediction horizon keeps increasing until $T = T_{\max}$ and the error bars are getting even smaller. The Pendubot task was considered solved after 26 policy searches.

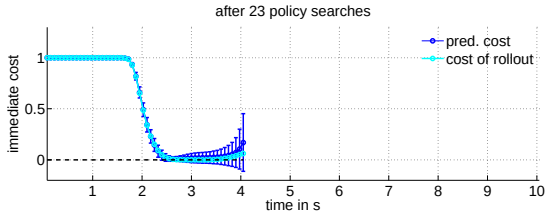
Figure 3.27 illustrates a learned solution to the Pendubot task. The learned controller attempted to keep both pendulums aligned. Substantial reward was gained after crossing the horizontal. From the viewpoint of mechanics, alignment of the two pendulums increases the total moment of inertia leading to a faster swing-up movement. However, it might require more energy for swinging up than a strategy where the two pendulums are not aligned.[40] Since the torque applied to the inner pendulum was constrained, but not penalized in the cost function defined in equations (3.42) and (3.43), alignment of the two pendulums was therefore an efficient strategy of solving the Pendubot task. In a typical successful rollout, the controller swung the Pendubot up and balanced it in an almost exactly inverted position: the inner joint had a deviation of up to 0.072 rad (4.125 °), the outer joint had a deviation of up to −0.012 rad (0.688 °) from the respective inverted positions. This non-optimal solution (also shown in Figure 3.27) was maintained by the inner joint exerting small (negative) torques.

Table 3.9 summarizes the results of the Pendubot learning task for a zero-order-hold controller. Learning the task required between one and two minutes, which is longer than the time required to learn the cart-pole task. This is essentially due to the more complicated dynamics requiring for more training data to learn a good model. Like in the cart-pole task, the learned controller for the Pendubot was fairly robust.

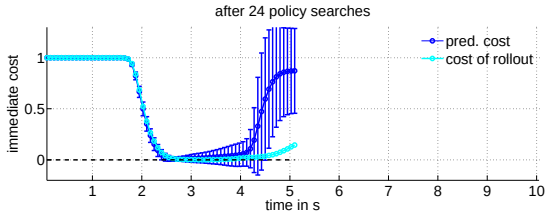
Zero-order-hold Control with Two Actuators

In the following, we consider the Pendubot system with an additional actuator for the outer link to demonstrate the applicability of our learning approach to multiple actuators (multivariate control signals).

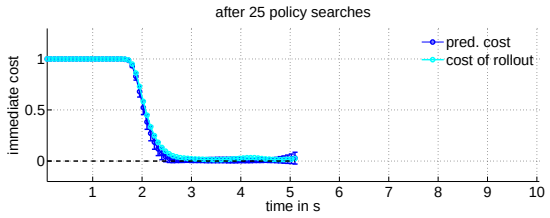
[40] I thank Toshiyuki Ohtsuka for pointing this relationship out.



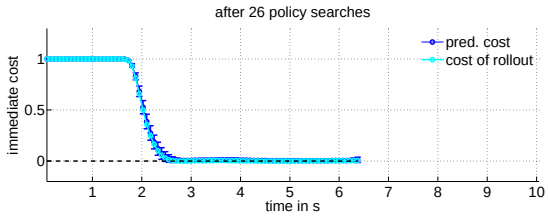
(a) Cost when applying a policy based on 66.3 s experience.



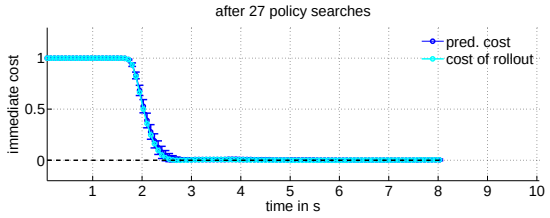
(b) Cost when applying a policy based on 71.4 s experience.



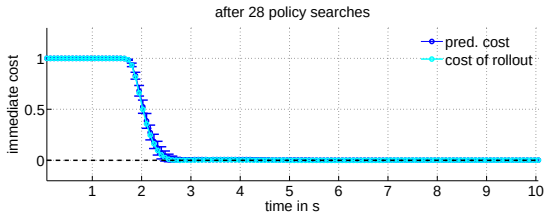
(c) Cost when applying a policy based on 77.775 s experience.



(d) Cost when applying a policy based on 85.8 s experience.



(e) Cost when applying a policy based on 95.85 s experience.



(f) Cost when applying a policy based on 105.9 s experience.

Figure 3.26: Predicted cost and incurred immediate cost during learning the Pendubot task (after 23, 24, 25, 26, 27, and 28 policy searches, from top left to bottom right). The x-axis is the time in seconds, the y-axis is the immediate cost. The black dashed line is the minimum immediate cost. The blue solid line is the mean of the predicted cost. The error bars show the 95% confidence interval. The cyan solid line is the cost incurred when the new policy is applied to the system. The prediction horizon T increases when a low cost at the end of the current horizon was predicted (see line 9 in Algorithm 2). The Pendubot task could be considered solved after 26 policy searches.

This two-link arm with two actuators, one for each pendulum, is no longer under-actuated.

In principle, the generalization of a univariate control signal to a multivariate control signal is straightforward: For each control dimension, we train a different policy, that is, either a linear function or an RBF network according to equations (3.9) and (3.10), respectively. We assume that the control dimensions are conditionally independent given a particular state (see Figure 2.5 for a graphical model in a GP context). However, when the state is uncertain, the control dimensions covary. The covariance between the control dimensions plays a central role in the simulation of the dynamic system when uncertainty is propagated forward (see Section 3.4). Both the linear controller and the RBF controller allow for the computation of a fully joint (Gaussian) distribution of the control signals to take the covariance between the signals into account. In case of the RBF controller, we closely follow the computations in Section 2.3.2. Given the fully joint distribution $p(\mathbf{u})$, we can also compute the fully joint Gaussian distribution $p(\mathbf{x}, \mathbf{u})$, which is required to cascade short-term predictions (see Figure 3.6(b)).

Compared to the Pendubot task with a single actuator, we increased the time discretization to $\Delta_t = 0.1\,\mathrm{s}$ and reduced the applicable torques to both joints to make the task more challenging:

- Using $\Delta_t = 0.075\,\mathrm{s}$ for the Pendubot with two actuators, the dynamics model was easier to learn than in the previous setup, where only a single actuator was available. However, using a time discretization of $\Delta_t = 0.1\,\mathrm{s}$ for the (standard) Pendubot task with a *single* actuator did not always lead to successful dynamics learning.

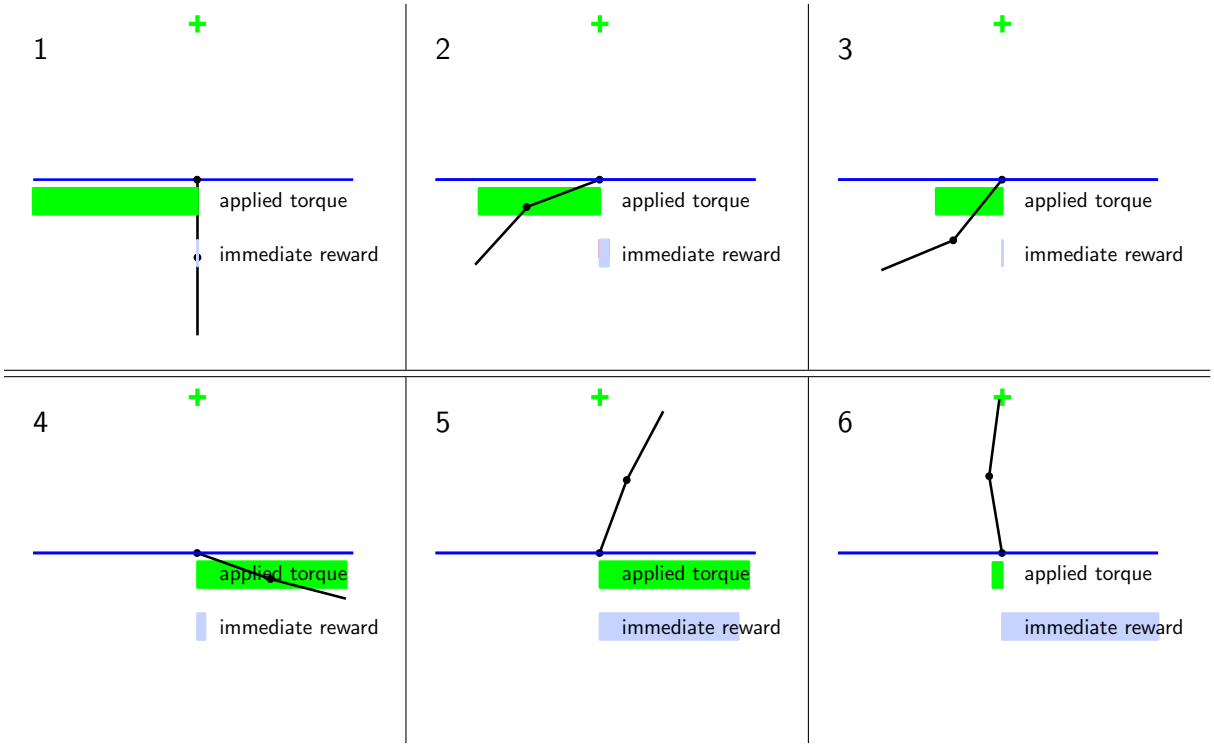


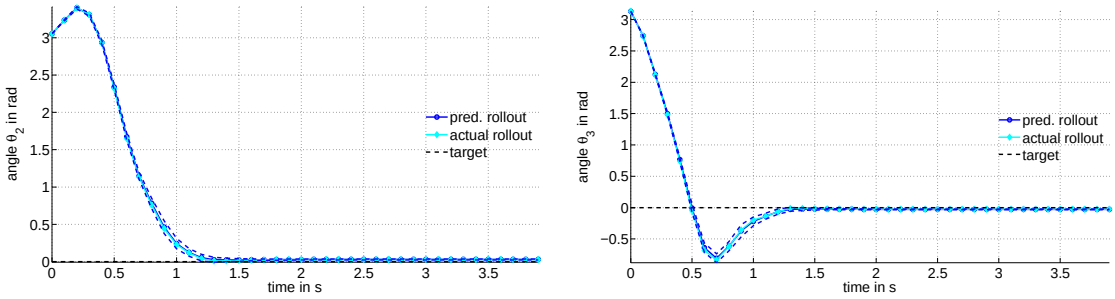
Figure 3.27: Illustration of the learned Pendubot task. Six snapshots of the swing up (top left to bottom right) are shown. The cross marks the target state of the tip of the outer pendulum. The green bar shows the torque exerted by the inner joint. The gray bar shows the reward (unity minus immediate cost). The learned controller attempts to keep the pendulums aligned.

Table 3.9: Experimental results: Pendubot (zero-order-hold control).

interaction time	136.05 s
task learned (negligible error bars)	after 85.8 s (26 trials)
failure rate ($d > l_3$)	5.4%
success rate ($d \leq 6$ cm)	94%
$V^{\pi^*}(\mathbf{x}_0)$, $\boldsymbol{\Sigma}_0 = 10^{-2}\mathbf{I}$	28.34

- Without torque reduction to 2 Nm per joint, the Pendubot could swing both links up directly. By contrast, a torque of 2 Nm was insufficient to swing a single joint directly up. However, when synergetic effects of the joints were exploited, the direct swing up of the outer pendulum was possible. Figure 3.28 illustrates this effect by showing typical trajectories of the angles of the inner and outer pendulum. As shown in Figure 3.28(a), the inner pendulum attached to the ground swung left and then right and up. By contrast, the outer pendulum, attached to the tip of inner one, swung up directly (Figure 3.28(b)). This direct swing-up of the outer joint was only possible since the synergetic effect of both pendulums was exploited. Note that both pendulums did not reach the target state (black dashed lines) exactly; both joints exerted small torques to maintain the slightly bent posture. In this posture, the tip of the outer pendulum was very close to the target, which means, that it was not costly to maintain the posture.

Table 3.10 summarizes the results of the Pendubot learning task for a zero-order-hold RBF controller with two actuators when following the evaluation setup in Algorithm 3. With an interaction time of about three minutes, our algorithm successfully learned a fairly robust controller. Note that the task was essentially learned after 10 trials or 40 s, which is less than half the trials and about half the interaction time required to learn the Pendubot task with a single actuator (see Table 3.9).



(a) Trajectories of the angle of the inner joint.

(b) Trajectories of the angle of the outer joint.

Figure 3.28: Example trajectories of the two angles for the two-link arm with two actuators when applying the learned controller. The x-axis shows the time, the y-axis shows the angle in radians. The blue solid lines are predicted trajectories when starting from a given state. The corresponding error bars show the 95% confidence intervals. The cyan lines are the actual trajectories when applying the controller. In both cases, the predictions are very certain, that is, the error bars are small. Moreover, the actual rollouts are in correspondence with the predictions.

Table 3.10: Experimental results: Pendubot with two actuators (zero-order-hold control).

time discretization	$\Delta_t = 0.1\,\mathrm{s}$
torque constraints	$u_1 \in [-2\,\mathrm{Nm}, 2\,\mathrm{Nm}]$, $u_2 \in [-2\,\mathrm{Nm}, 2\,\mathrm{Nm}]$
interaction time	192.9 s
task learned (negligible error bars)	after 40 s (10 trials)
failure rate ($d > l_3$)	1.5%
success rate ($d \leq 6\,\mathrm{cm}$)	98.5%
$V^{\pi^*}(\mathbf{x}_0)$, $\mathbf{\Sigma}_0 = 10^{-2}\mathbf{I}$	6.14

3.7.3 Cart-Double Pendulum

We followed Algorithm 2 to learning a dynamics model and a controller for the cart-double pendulum task (see Figure 3.29).

The cart-double pendulum dynamic system consists of a cart running on an infinite track and an attached double pendulum, which swings freely in the plane (see Figure 3.29). The cart can move horizontally when an external force u is applied to it. The pendulums are not actuated In Appendix B.3, the corresponding equations of motion are derived from first principles.

The state of the system was given by the position x_1 of the cart, the corresponding velocity $\dot{x}_1$, and the angles θ_2, θ_3 of the two pendulums with the corresponding angular velocities $\dot{\theta}_2, \dot{\theta}_3$, respectively. The angles were measured anti-clockwise from the upright position. For the simulation, the internal state representation was

$$\mathbf{x} = \begin{bmatrix} x_1 & \dot{x}_1 & \dot{\theta}_2 & \dot{\theta}_3 & \sin\theta_2 & \cos\theta_2 & \sin\theta_3 & \cos\theta_3 \end{bmatrix}^\top \in \mathbb{R}^8 .$$

Initially, the cart-double pendulum was expected to be in a state where the cart was below the green cross in Figure 3.29 and the pendulums hung down ($x_1 = 0, \theta_2 = \pi = \theta_3$). The objective was to learn a policy to swing the double pendulum up and to balance the tip of the outer pendulum at the target state in the inverted position (green cross in Figure 3.29) by applying forces to the cart only. In order to solve this task optimally, the cart had to stop at the position exactly below the cross. The cart-double pendulum task is challenging since the under-actuated dynamic system is inherently unstable. Moreover, the dynamics are chaotic. A linear controller is not capable to solve the cart-double pendulum task.

A standard control approach to solving the swing up plus balancing problem is to design two controllers, one for the swing up and one linear controller for the balancing task (Alamir and Murilo, 2008;

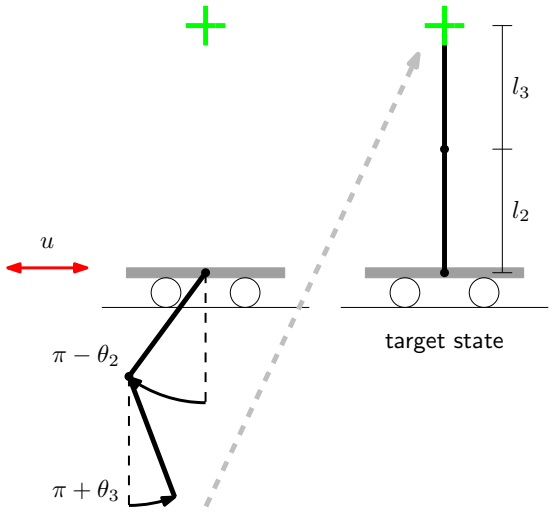


Figure 3.29: Cart with attached double pendulum. The cart can be pushed to the left and to the right in order to swing the double pendulum up and to balance it in the inverted position. The target state of the tip of the outer pendulum is denoted by the green cross.

Zhong and Röck, 2001; Huang and Fu, 2003; Graichen et al., 2007). Unlike this engineered solution, in our approach, a single nonlinear RBF controller was *learned* to solve both subtasks together.

The parameter settings for the cart-double pendulum system are given in Appendix C.3. The chosen sampling frequency of $13.\bar{3}\,\text{Hz}$ is fairly slow for this kind of problem. For example, both Alamir and Murilo (2008) and Graichen et al. (2007) sampled with $1,000\,\text{Hz}$ and Bogdanov (2004) sampled with $50\,\text{Hz}$ to control the cart-double pendulum, where Bogdanov (2004), however, solely considered the stabilization of the system, a problem where the system dynamics are fairly slow.

Cost Function

Every $\Delta_t = 0.075\,\text{s}$, the squared Euclidean distance

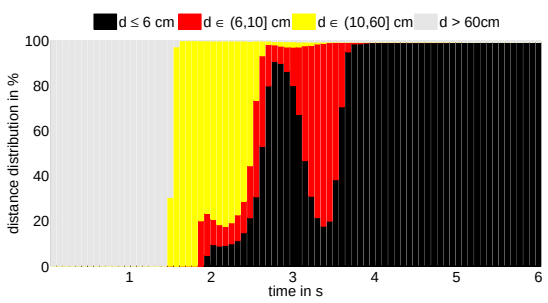
$$\begin{aligned} d(\mathbf{x}, \mathbf{x}_{\text{target}})^2 &= \left(x_1 - l_2 \sin\theta_2 - l_3 \sin\theta_3\right)^2 + \left(l_2 + l_3 - l_2 \cos\theta_2 - l_3 \cos\theta_3\right)^2 \\ &= x_1^2 + l_2^2 + l_3^2 + (l_2 + l_3)^2 - 2x_1 l_2 \sin\theta_2 - 2x_1 l_3 \sin\theta_3 + 2l_2 l_3 \sin\theta_2 \sin\theta_3 \\ &\quad - 2(l_2 + l_3) l_2 \cos\theta_2 - 2(l_2 + l_3) l_3 \cos\theta_3 + 2l_2 l_3 \cos\theta_2 \cos\theta_3 \end{aligned} \tag{3.44}$$

from the tip of the outer pendulum to the target state was measured, where $l_i = 0.6\,\text{m}$, $i = 2, 3$, are the lengths of the pendulums. The relevant variables of the state $\mathbf{x}$ were the position x_1 and the sines and the cosines of the angles θ_i. An approximate Gaussian joint distribution $p(\mathbf{j}) = \mathcal{N}(\mathbf{m}, \mathbf{S})$ of the involved parameters

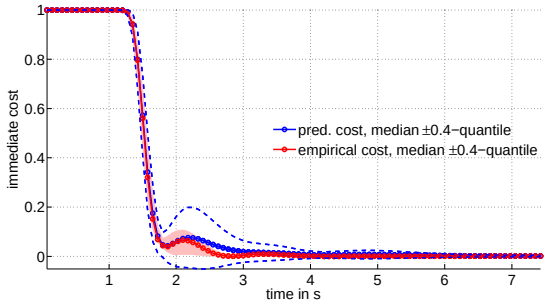
$$\mathbf{j} \coloneqq \begin{bmatrix} x_1 & \sin\theta_2 & \cos\theta_2 & \sin\theta_3 & \cos\theta_3 \end{bmatrix}^\top \tag{3.45}$$

was computed using the results from Appendix A.1. The target vector in $\mathbf{j}$-space was $\mathbf{j}_{\text{target}} = [0, 0, 1, 0, 1]^\top$. The first coordinate of $\mathbf{j}_{\text{target}}$ is the optimal position of the cart when both pendulums are in the inverted position. The matrix $\mathbf{T}^{-1}$ in equation (3.19) was given by

$$\mathbf{T}^{-1} = a^2 \begin{bmatrix} 1 & -l_2 & 0 & -l_3 & 0 \\ -l_2 & l_2^2 & 0 & l_2 l_3 & 0 \\ 0 & 0 & l_2^2 & 0 & l_2 l_3 \\ -l_3 & l_2 l_3 & 0 & l_3^2 & 0 \\ 0 & 0 & l_2 l_3 & 0 & l_3^2 \end{bmatrix} = a^2 \mathbf{C}^\top \mathbf{C}, \qquad \mathbf{C}^\top = \begin{bmatrix} 1 & 0 \\ -l_2 & 0 \\ 0 & l_2 \\ -l_3 & 0 \\ 0 & l_3 \end{bmatrix},$$



(a) Histogram of the distances of the tip of the outer pendulum to the target of 1,000 rollouts.



(b) Quantiles of the predictive immediate cost distribution (blue) and the empirical immediate cost distribution (red).

Figure 3.30: Cost distribution for the cart-double pendulum problem (zero-order-hold control).

where $1/a$ controlled the width of the saturating immediate cost function in equation (3.18). The saturating immediate cost was then

$$c(\mathbf{x}) = c(\mathbf{j}) = 1 - \exp\left(-\tfrac{1}{2}(\mathbf{j} - \mathbf{j}_{\text{target}})^\top \mathbf{T}^{-1}(\mathbf{j} - \mathbf{j}_{\text{target}})\right) \in [0, 1]\,. \tag{3.46}$$

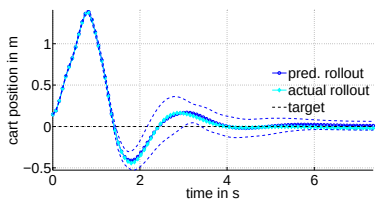
The width $1/a = 0.5\,\text{m}$ of the cost function in equation (3.18) was chosen, such that the immediate cost was about unity as long as the distance between the tip of the outer pendulum and the target state was greater than both pendulums together. Thus, the tip of the outer pendulum had to move above horizontal to reduce the immediate cost significantly from unity.

Zero-order-hold Control

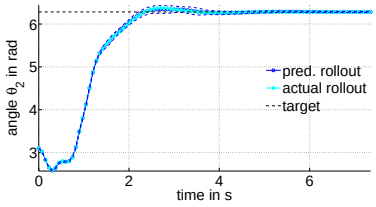
As described by Algorithm 3, we considered 1,000 controlled trajectories of 20 s length each to evaluate the performance of the learned controller. The start states of the trajectories were independent samples from $p(\mathbf{x}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$, the distribution for which the controller was learned.

Figure 3.30 shows cost distributions for the cart-double pendulum learning task. Figure 3.30(a) shows a histogram of the empirical distribution of the distance d of the tip of the outer pendulum to the target over 6 s after 1,000 rollouts from start positions randomly sampled from $p(\mathbf{x}_0)$. The histogram is cut at 6 s since the cost distribution looks alike for $t \geq 6\,\text{s}$. It took the system about 1.5 s to leave the high-cost region denoted by the gray bars. After about 1.5 s, in many trajectories, the tip of the outer pendulum was closer to the target than its own length $l_3 = 60\,\text{cm}$ as shown by the appearing yellow bars. This means, the tip of the outer pendulum was certainly above horizontal. After about 2 s, the tip of the outer pendulum was close to the target for the first rollouts, which is illustrated by the increasing black bars. After about 2.8 s the black bars "peak" meaning that at this time point in many trajectories the tip of the outer pendulum was very close to the target state. The decrease of the black bars and the increase of the red bars between 2.8 s and 3.2 s is due to a slight overshooting of the cart to reduce the energy in the system; the RBF controller switched from swinging up to balancing. However, the pendulums typically did not fall over. After 4 s, the red bars vanish, and the black bars level out at 99.1%. Like for the cart-pole task (Figure 3.14), the controller either brought the system close to the target, or it failed completely.

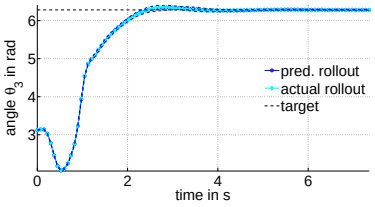
Figure 3.30(b) shows the median and the lower and upper 0.1-quantiles of both a Gaussian approximation of the predicted immediate cost and the empirical immediate cost over 7.5 s. For the first approximately 1.2 s, both immediate cost distributions are at unity without variance. Between 1.2 s and 1.875 s the cost distributions transition from a high-cost regime to a low-cost regime with increasing uncertainty. At 1.875 s, the medians of both the predicted and the empirical cost distributions have a local minimum. Note that at this point in time the red bars in the cost histogram in Figure 3.30(a) start appearing. The uncertainty in both the predicted and the empirical immediate cost in Figure 3.30(b) significantly increased between 1.8 s and 2.175 s since the controller had to switch from the swing up to decelerating the speeds of the cart and both pendulums and balancing. After $t = 2.175\,\text{s}$ the error bars and the medians declined toward zero. The error bars of the predicted immediate cost were generally



(a) Position of the cart. The initial uncertainty is very small. After about 1.5 s the cart was slowed down and the predicted uncertainty increased. After approximately 4 s, the uncertainty decreased again.



(b) Angle of the inner pendulum.



(c) Angle of the outer pendulum.

Figure 3.31: Example trajectories of the cart position and the two angles of the pendulums for the cart-double pendulum when applying the learned controller. The x-axes show the time, the y-axes show the cart position in meters and the angles in radians. The blue solid lines are the predicted trajectories when starting from a given state. The dashed blue lines show the 95% confidence intervals. The cyan lines are the actual trajectories when applying the controller. The actual rollouts agree with the predictions. The increase in the predicted uncertainty in all three state variables between $t = 1.5\,\mathrm{s}$ and $t = 4\,\mathrm{s}$ indicates the time interval when the controller removed energy from the system to stabilize the double pendulum at the target state.

larger than the error bars of the empirical immediate cost for two reasons: First, the model uncertainty was taken into account. Second, the predictive immediate cost distribution was always represented its mean and variance. By contrast, as shown in Figure 3.30(a), the true distribution of the immediate cost had a strong peak close to zero and some outliers where the controller did not succeed. These outliers were not predicted by the dynamics model, otherwise the predicted mean would have been shifted toward unity. The dynamics model did not predict failings, which could occur when the start state was in a tail of $p(\mathbf{x}_0)$; the dynamics model had no or only sparse training data in these regions.

Let us consider a single trajectory starting from a given position $\mathbf{x}_0$. For this case, Figure 3.31 shows the corresponding predicted trajectories $p(\mathbf{x}_t)$ and the actual trajectories of the position of the cart, the angle of the inner pendulum, and the angle of the outer pendulum, respectively. Note that for the predicted state distributions $p(\mathbf{x}_t)$ the dynamics model did a multiple-step ahead prediction using the learned controller for internal simulation—before applying the policy to the real system (updates of the predicted state distributions were impossible). In all three cases, the actual rollout agreed with the predictions. In particular in the position of the cart in Figure 3.31(a), it can be seen that the predicted uncertainty grew and declined although no new additional information was incorporated. The uncertainty increase was exactly during the phase where the controller switched from swinging the pendulums up to balancing them in the inverted position. Figure 3.31(b) and Figure 3.31(c) nicely show that the angles of the inner and outer pendulums were very much aligned from 1 s onward.

The learned RBF-controller implemented a policy that attempted to align both pendulums. From the viewpoint of mechanics, alignment of two pendulums increases the total moment of inertia leading to a faster swing-up movement. However, it might require more energy for swinging up than a strategy where the two pendulums are not aligned. Since the force applied to the cart was constrained, but not penalized in the cost function defined in (3.45) and (3.46), alignment of the two pendulums presumably yielded a lower long-term cost V^π than any other configuration.

Figure 3.32 shows snapshots of a typical trajectory when applying the learned controller. The learned policy payed more attention to the angles of the pendulums than to the position of the cart: At the end of a typical rollout, both pendulums were exactly upright, but the position of the cart was about 2 cm off to the left side. This makes intuitive sense since the angles of the pendulums can only be controlled indirectly via the force applied to the cart. Hence, correcting the angle of a pendulum requires to change the position of the cart. Not correcting the angle of the pendulum would lead to a fall-over. By contrast, if the cart position is slightly off, maintaining the cart position does not lead to a complete failure but only to a suboptimal solution.

Table 3.11 summarizes the experimental results of the cart-double pendulum learning task. With

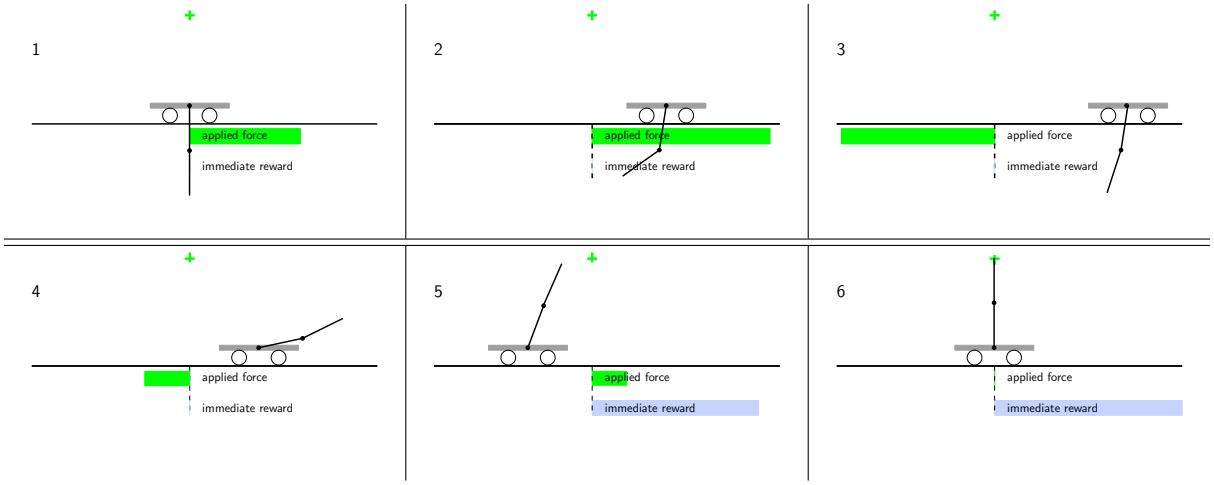


Figure 3.32: Sketches of the learned cart-double pendulum task (top left to bottom right). The green cross marks the target state of the tip of the outer pendulum. The green bars show the force applied to the cart. The gray bars show the reward (unity minus immediate cost). To reach the target exactly, the cart has to be directly below the target. The ends of the track the cart is running on denote the maximum applicable force and the maximum reward (at the right-hand side).

Table 3.11: Experimental results: cart-double pendulum (zero-order hold).

interaction time	98.85 s
task learned (negligible error bars)	after 84 s (23 trials)
failure rate ($d > l_3$)	0.9%
success rate ($d \leq 6\,\text{cm}$)	99.1%
$V^{\pi^*}(\mathbf{x}_0)$, $\boldsymbol{\Sigma}_0 = 10^{-2}\mathbf{I}$	6.14

an interaction time of between one and two minutes, our algorithm successfully learned a very robust controller. Occasional failures can be explained by encountering states where the policy and/or the dynamics model are not particularly good.

3.7.4 Robotic Unicycle

The final application in this chapter is to apply our learning framework to learning a dynamics model and a controller for balancing a robotic unicycle. A unicycle system is composed of a unicycle and a rider. This system is inherently unstable. However, if the rider is skilled, he can balance the unicycle without falling. We apply our learning framework to the task of riding a robotic unicycle. The human rider is replaced by two torque motors, one of which is used to drive the unicycle forwards (instead of using pedals), the second motor is used to prevent the unicycle from falling sideways and mimics twisting. A robotic unicycle can be considered a nonlinear control system similar to an inverted pendulum moving in a two-dimensional space with a unicycle cart as its base.

Figure 3.33 is an illustration of the considered unicycle system. Two torques can be applied to the system: The first torque u_w is applied directly on the wheel and corresponds to a human rider using pedals. The torque produces longitudinal and tilt accelerations. Lateral stability of the wheel can be maintained by either steering the wheel toward the direction in which the unicycle is falling and/or by applying a torque u_t to the turntable.

The target application we have in mind is to learn a stabilizing controller for balancing the robotic unicycle in the Department of Engineering, University of Cambridge, UK. A photograph of the robotic unicycle, which is assembled according to the descriptions above, is shown in Figure 3.34.[41] The equations

[41] Detailed information about the project can be found at `http://www.roboticunicycle.info/`.

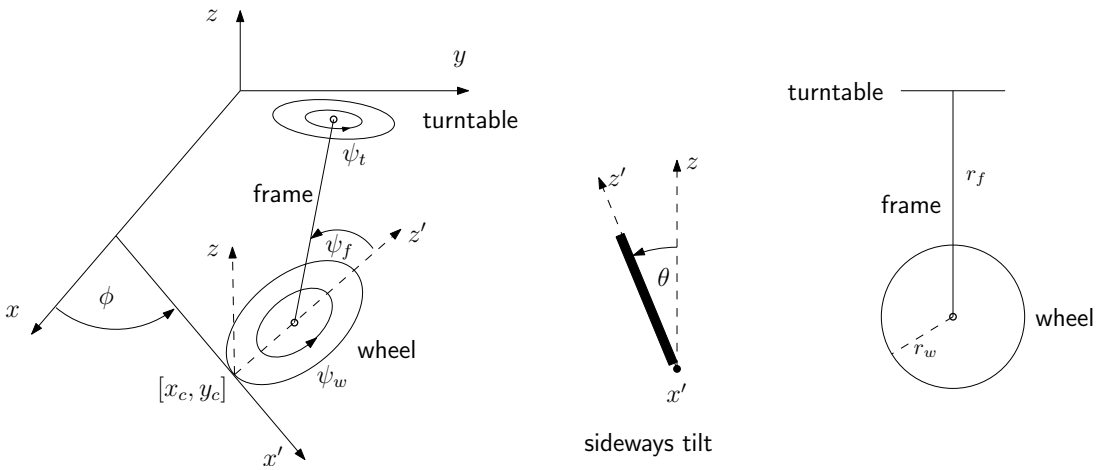


Figure 3.33: Sketch of the robotic unicycle. The robotic unicycle consists of a wheel, a frame, and a turntable (flywheel). The wheel of the unicycle rolls without slip on a horizontal plane along the x'-axis, which is the x-axis rotated by the angle ϕ around the z-axis of the world-coordinate system. The contact point of the wheel with the surface is given by the Euclidean coordinates $[x_c, y_c]^\top$. The wheel can fall sideways, that is, it can be considered a body rotating around the new x-axis along which the wheel is rolling. This sideways tilt is denoted by θ. The frame can fall forward and/or backward in the hyperplane of the wheel. The angle of the frame with respect to the axis z' in Figure 3.33 is denoted by ψ_f. The turntable (flywheel) is mounted perpendicular to the frame. The rotational angle of the wheel and the turntable are given by ψ_w and ψ_t, respectively.

of motion for this robotic unicycle can be found in the thesis by Forster (2009). In the following, however, we only consider an idealized *computer simulation* of the robotic unicycle. Learning the controller using data from the hardware realization of the unicycle remains to future work.

The robotic unicycle is a challenging control problem due to its intrinsically nonlinear dynamics. Moreover, it is non-minimum-phase and non-holonomic[42]. Without going into details, following a Lagrangian approach to deriving the equations of motion, the non-holonomic constraints on the speed variables $[\dot{x}_c, \dot{y}_c]$ were incorporated into the remaining state variables. The state ignores the absolute position of the contact point of the unicycle, which is irrelevant for stabilization. The state of the system was then given as

$$\mathbf{x} = \begin{bmatrix} \dot{\theta} & \dot{\phi} & \dot{\psi}_w & \dot{\psi}_f & \dot{\psi}_t & \theta & \phi & \psi_w & \psi_f & \psi_t \end{bmatrix} \in \mathbb{R}^{10} .$$

Thus, the dynamics of the robotic unicycle were described by ten coupled second-order ordinary differential equations, which were used for numerical simulation. During simulation, the state was represented as an $\mathbb{R}^{15}$-vector

$$\mathbf{x} = \left[\dot{\theta},\, \dot{\phi},\, \dot{\psi}_w,\, \dot{\psi}_f,\, \dot{\psi}_t,\, \sin\theta,\, \cos\theta,\, \sin\phi,\, \cos\phi,\, \sin\psi_w,\, \cos\psi_w,\, \sin\psi_f,\, \cos\psi_f,\, \sin\psi_t,\, \cos\psi_t\right]^\top$$

since we mapped the angles to their sines and cosines. The objective was to balance the unicycle, that is, to prevent it from falling over. The location of the unicycle was not represented and irrelevant for solving the task.

We employed a linear controller for the stabilization of the robotic unicycle and followed the steps of Algorithm 2. In contrast to the previous learning tasks, however, 15 trajectories with random torques were used to initialize the dynamics model. Furthermore, we aborted the simulation when the sideways tilt θ of the wheel or the angle ψ_f of the frame exceeded an angle of $\pi/3$. For $\theta = \pi/2$ the unicycle would lay flat on the ground. The fifteen initial trajectories were typically short since the unicycle quickly fell over when applying torques randomly to the wheel and the turntable.

[42] Roughly speaking, the controllable degrees of freedom are less than the total degrees of freedom.

Figure 3.34: Robotic unicycle in the Department of Engineering, University of Cambridge, UK. With permission borrowed from `http://www.roboticunicycle.info/`.

Cost Function

The objective was to balance the unicycle. Therefore, the tip of the unicycle should have the z-coordinate in a three-dimensional Cartesian coordinate system defined by the radius r_w of the wheel plus the length r_f of the frame. Every $\Delta_t = 0.05\,\mathrm{s}$, the squared Euclidean distance

$$\begin{aligned} d(\mathbf{x}, \mathbf{x}_{\text{target}})^2 &= \big(\overbrace{r_w + r_f}^{\text{upright}} - r_w \cos\theta - r_f \cos\theta \cos\psi_f\big)^2 \\ &= \big(r_w + r_f - r_w \cos\theta - \tfrac{r_f}{2}\cos(\theta - \psi_f) - \tfrac{r_f}{2}\cos(\theta + \psi_f)\big)^2 \end{aligned} \tag{3.47}$$

from the tip of the unicycle to the upright position (a z-coordinate of $r_w + r_f$) was measured. The squared distance in equation (3.47) did not penalize the position of the contact point of the unicycle since the task was to balance the unicycle *somewhere*. Note that d only depends on the angles θ (sideways tilt of the wheel) and ψ_f (tilt of the frame in the hyperplane of the tilted wheel). In particular, d does not depend on the angular velocities and the applied torques $\mathbf{u}$.

The state variables that were relevant to compute the squared distance in equation (3.47) were the cosines of θ and the difference/sum of the angles θ and ψ_f. Therefore, we defined $\chi := \theta - \psi_f$ and $\xi := \theta + \psi_f$. An approximate Gaussian distribution $p(\mathbf{j}) = \mathcal{N}(\mathbf{m}, \mathbf{S})$ of the state variables

$$\mathbf{j} = \begin{bmatrix} \cos\theta & \cos\chi & \cos\xi \end{bmatrix}^\top$$

that are relevant for the computation of the cost function was computed using the results from Appendix A.1. The target vector in $\mathbf{j}$-space was $\mathbf{j}_{\text{target}} = [1, 1, 1]^\top$. The matrix $\mathbf{T}^{-1}$ in equation (3.19) was

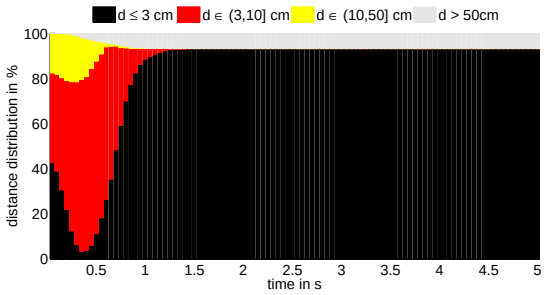


Figure 3.35: Histogram of the distances from the top of the unicycle to the fully upright position after 1,000 test rollouts.

given by

$$\mathbf{T}^{-1} = a^2 \begin{bmatrix} r_w^2 & \frac{r_w r_f}{2} & \frac{r_w r_f}{2} \\ \frac{r_w r_f}{2} & \frac{r_f^2}{4} & \frac{r_f^2}{4} \\ \frac{r_w r_f}{2} & \frac{r_f^2}{4} & \frac{r_f^2}{4} \end{bmatrix} = a^2 \mathbf{C}^\top \mathbf{C}\,, \quad \mathbf{C} = \begin{bmatrix} r_w & \frac{r_f}{2} & \frac{r_f}{2} \end{bmatrix},$$

where $1/a = 0.1\,\mathrm{m}$ controlled the width of the saturating cost function in equation (3.18). Note that almost full cost incurred if the tip of the unicycle exceeded a distance of 20 cm from the upright position.

Zero-order-hold Control

We followed the steps described in Algorithm 2 to automatically learn a dynamics model and a (linear) controller to balance the robotic unicycle.

According to Algorithm 3, the controller was tested in 1,000 independent runs of 30 s length each starting from a randomly drawn initial state $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, 0.25^2\mathbf{I})$. Note that the distribution $p(\mathbf{x}_0)$ of the initial state was fairly wide. The (marginal) standard deviation for each angle was $0.25\,\mathrm{rad} \approx 15\,^\circ$. A test run was aborted in case the unicycle fell over.

Figure 3.35 shows a histogram of the empirical distribution of the distance d from the top of the unicycle to the upright position over 5 s after 1,000 rollouts from random start positions (sampled from $p(\mathbf{x}_0)$). The histogram is cut at $t = 5\,\mathrm{s}$ since the cost distribution does not change afterward. The histogram distinguishes between states close to the target (black bars), states fairly close to the upright position (red bars), states that might cause a fall-over (yellow bars), and states, where the unicycle already fell over or could not be prevented from falling over (gray bars). The initial distribution of the distances gives an intuition of how far the random initial states were from the upright position: In approximately 80% of the initial configurations, the top of the unicycle was closer than ten centimeters to the upright position. About 20% of the states had a distance between ten and fifty centimeters to the upright position. Within the first 0.4 s, the distances to the target state grew for many states that used to be in the black regime. Often, this depended on the particular realization of the state including the sampled angular velocities. Most of the states that were previously in the black regime moved to the red regime. Additionally, some states from the red regime became parts of the yellow regime of states. In some cases, the initial configuration was too bad that a fall-over could not be prevented, which is indicated by the gray error bars. Between 0.4 s and 0.7 s, the black bars increase and the yellow bars almost vanish. The yellow bars vanish since either the state could be controlled (yellow becomes red) or it could not and the unicycle fell over (yellow becomes gray). The heights of the black bars increase since some of the states in the red regime got closer to the target again. After about 1.2 s, the result is essentially binary: Either the unicycle fell over or the linear controller managed to balance it very close to the desired upright position. The success rate was approximately 93%.

Table 3.12 summarizes the results of the unicycle task. The interaction time of 32.85 s was sufficient to learn a fairly robust (linear) policy. After 23 trials (15 of which were random) the task was essentially solved. In about 7% of 1,000 test runs (each of length 30 s) the controller was incapable to balance the unicycle starting from a randomly drawn initial state $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, 0.25^2\mathbf{I})$. Note however, that the covariance matrix $\boldsymbol{\Sigma}_0$ allowed for some initial states where the angles deviate by more than $30\,^\circ$ from the

Table 3.12: Experimental results: unicycle (zero-order hold).

interaction time	32.85 s
task learned (negligible error bars)	after 17.75 s (23 trials)
failure rate (fall-over)	6.8%
success rate (stabilization)	93.2%
$V^{\pi^*}(\mathbf{x}_0)$, $\boldsymbol{\Sigma}_0 = 0.25^2\mathbf{I}$	6.02

upright position. Bringing the unicycle upright from these extreme angles can be physically impossible with the linear controller due to the torque constraints.

In a typical successful run, the learned controller kept the unicycle upright, but drove it straight ahead with relatively high speed. Intuitively, this solution makes sense: Driving the unicycle straight ahead leads to more mechanical stability than just keeping it upright due to the conservation of the angular momentum. The same effect can be experienced in ice-skating or riding a bicycle, for example. When just keeping the unicycle upright (without rolling), the unicycle can fall into all directions. By contrast, a unicycle rolling straight ahead is unlikely to fall sideways.

3.8 Practical Considerations

When facing real-world applications, we are typically facing two major problems: large data sets and noisy (and partial) observations of the state of the dynamic system. In the following, we briefly touch upon these topics.

3.8.1 Large Data Sets

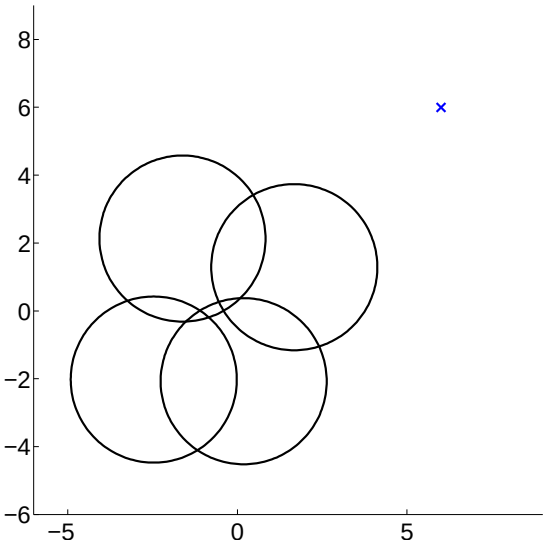
Although training a GP with a training set with 500 data points can be done in short time (see Section 2.3.4 for the computational complexity), *repeated predictions* in the approximate inference step (Section 3.4) and the computation of the derivatives for the gradient-based policy search (Section 3.5) become computationally expensive: On top of the computations required for multivariate predictions with uncertain inputs (see Section 2.3.4), computing the derivatives of the predictive covariance matrix $\boldsymbol{\Sigma}_t$ with respect to the covariance matrix $\boldsymbol{\Sigma}_{t-1}$ of the input distribution and with respect to the policy parameters $\boldsymbol{\psi}$ of the RBF policy requires $\mathcal{O}(F^2n^2D)$ operations per time step, where F is the dimensionality of the control signal to be applied, n is the size of the training set, and D is the dimension of the training inputs. Hence, repeated prediction and derivative computation for approximate inference and optimization become very demanding although the computational effort scales linearly with the prediction horizon T.[43] Therefore, we use sparse approximations to speed up training, predicting, and the computation of the required derivatives.

Sparse Approximations for Episodic Learning in Control

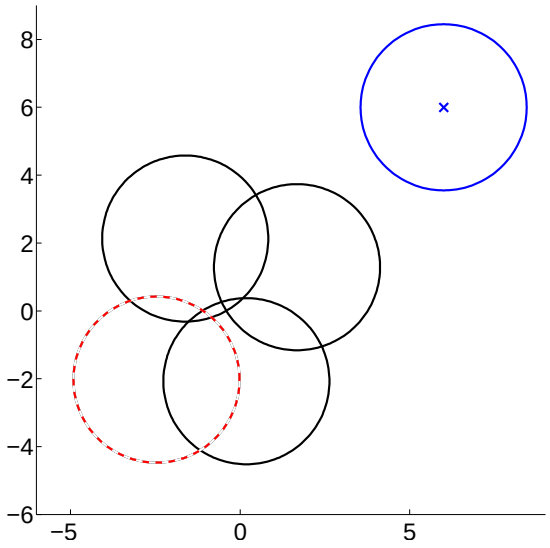
In the following, we briefly discuss sparse approximations in the context of the control learning task, where we face the problem of acquiring data sequentially, that is, after each interaction with the system (see Algorithm 1). The state-of-the-art sparse GP algorithms by Snelson and Ghahramani (2006), Snelson (2007), and Titsias (2009) are based on the concept of inducing inputs (see Section 2.4 for a brief introduction). However, they are typically used in the context of a fixed data set. We identified two main problems with these sparse approximations in the context of our learning framework:

- When the locations of the inducing inputs and the kernel hyper-parameters are optimized jointly, the FITC sparse approximation proposed by Snelson and Ghahramani (2006) and Snelson (2007)

[43] In case of *stochastic transition dynamics*, that is $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) + \mathbf{w}_t$, where $\mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_w)$ is a random offset that affects the state of the system, the derivatives with respect to the distribution of the previous state still require $\mathcal{O}(E^2n^2D)$ arithmetic operations per time step, where E is the dimension of the GP training targets. However, when using a *stochastic policy* the derivatives with respect to the policy parameters require $\mathcal{O}(F^2n^2D + Fn^3)$ arithmetic operations per time step.



(a) Initial placement of basis functions. The blue cross is a new data point.



(b) A better model can require to move the red-dashed basis function to the location of the desired basis function (blue).

Figure 3.36: The FITC sparse approximation encounters optimization problems when moving the location of an unimportant basis function "through" other basis functions if the locations of these other basis functions are crucial for the model quality. These problems are due to the gradient-based optimization of the basis functions locations.

is good in fitting the model, but can be poor in predicting. Our experience is that it can suffer from overfitting indicated by a too small (by several orders of magnitude) learned hyper-parameter for the noise variance. By contrast, the recently proposed algorithm by Titsias (2009) attempts to avoid overfitting but can suffer from underfitting.

As mentioned in the beginning of this section, the main computational burden arises from repeated predictions and computations of the derivatives, but not necessarily from training the GPs. To avoid the issues with overfitting and underfitting, we train the full GP to obtain the hyper-parameters. After that, we optimize the locations of the pseudo-inputs while keeping the hyper-parameters from the full GP model fixed.

- Besides the overfitting and underfitting problems, we ran into problems that have to do with the sequential nature of the data set for the dynamics GP in the light of our learning framework. A sparse approximation for the dynamics GP "compresses" collected experience. The collected experience consists of trajectories that always start from the same initial state distribution $p(\mathbf{x}_0)$. Therefore, the first parts of the rollouts are very similar to each other. After learning a good controller that can solve the problem, the last parts of the rollouts almost equal each other as well. This experience blows the size of the data set up, but does not yield much new information. The sparse methods by Snelson and Ghahramani (2006), Snelson (2007), and Titsias (2009) did not have difficulties to represent these bits of the data set. However, there is other experience that is more challenging to model: Suppose we collected some experience around the initial position and now observe states along a new rollout trajectory that substantially differs from the experience so far. In order to model these data, the locations of the pseudo-inputs $\bar{\mathbf{X}}$ in the sparse model have to be moved (with the simplifying assumption that the hyper-parameters do not change). Figure 3.36 illustrates this setting. The left panel shows possible initial locations of four black basis functions, which are represented by ellipses. These basis functions represent the GP model given the data *before* obtaining the new experience. The blue cross represents new data that is not sufficiently covered by the black basis functions. A *full* GP would simply place a new function at the location of the blue cross. For the sparse GP, the number of basis functions is typically fixed a priori. The panel on the right-hand side of Figure 3.36 contains the same four basis functions, one of which is

Algorithm 5 Sparse swap

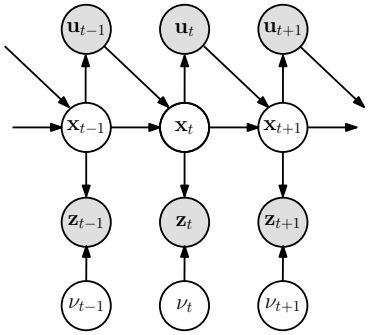
```
1:  init: X̄, X                                   ▷ pseudo inputs and training set
2:  nlml = sparse_nlml(X̄)                        ▷ neg. log-marginal likelihood for pseudo inputs
3:  loop
4:      for i = 1 to M do                        ▷ for all pseudo training inputs
5:          e1(i) = sparse_nlml(X̄ \ x̄_i)         ▷ neg. log-marginal likelihood for reduced set
6:      end for
7:      i* = arg min_i e1
8:      X̄ := X̄ \ x̄_{i*}                          ▷ delete least important pseudo input
9:      for j = 1 to PS do                       ▷ for all inputs of the full training set
10:         e2(j) = sparse_nlml(X̄ ∪ x_j)         ▷ neg. log-marginal likelihood for augmented set
11:     end for
12:     j* = arg min_j e2
13:     if e2(j*) < nlml then
14:         X̄ := X̄ ∪ x_{j*}                      ▷ add best input x_j from real data set as new pseudo input
15:         nlml := e2(j*)
16:     else
17:         X̄ := X̄ ∪ x̄_{i*}                      ▷ recover pseudo training set from previous iteration
18:         return X̄                             ▷ exit loop
19:     end if
20: end loop
```

dashed and red. Let us assume that the blue basis function is the optimal location of the red-dashed basis function in order to model the data set *after* obtaining new experience. If the black basis functions are crucial to model the latent function, it is very difficult to move the red-dashed basis function to the location of the blue basis function using a gradient-based optimizer. To sidestep this problem, we follow Algorithm 5. The main idea of the heuristic in Algorithm 5 is to replace some pseudo training inputs $\bar{\mathbf{x}}_i \in \bar{\mathbf{X}}$ with inputs $\mathbf{x}_j \in \mathbf{X}$ from the full training set if the model improves. In the context of Figure 3.36, the red Gaussian corresponds to $\bar{\mathbf{x}}_{i^*}$ in line 8. The blue Gaussian is an input $\mathbf{x}_j$ of the full training set and represents a potentially "better" location of a pseudo input. The quality of the model is evaluated by the `sparse_nlml`-function (lines 2, 5, and 10) that computes the negative log-marginal likelihood (negative log-evidence) in equation (2.29) for the sparse GP approximation. The log-marginal likelihood can be evaluated efficiently since swapping a data point in or out only requires a rank-one-update of the low-rank approximation of the kernel matrix.

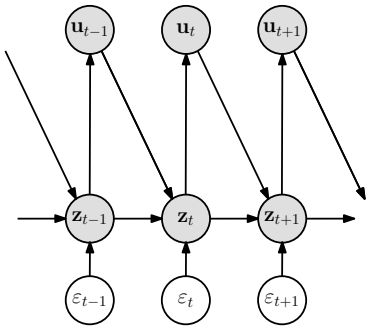
We emphasize that the problems in the sequential-data setup stem from the locality of the Gaussian basis function. Stepping away from local basis functions to global basis functions should avoid the problems with sequential data completely. Trigonometric basis functions could be a reasonable choice. The evaluation of this approach remains to future work.

We identified two problems with sparse GP approximations: First, there is the problem of overfitting or underfitting related to the hyper-parameters. Second, there is the problem of moving basis functions around when data is collected sequentially. To account for both problems, we employ the following method when the size of the data set exceeds 250 data points: We use the hyper-parameters of the full GP and solely optimize the pseudo-input locations in the sparse model. We switch to the modified sparse representation of the dynamics GP given in Algorithm 5 to account for the sequential-data problem. Typically, the algorithm swaps a lot in the early stages of learning when the characteristics of the data set vary most. When a good controller has been found and the new trajectories look very similar; the model no longer changes significantly and no more swapping is required.

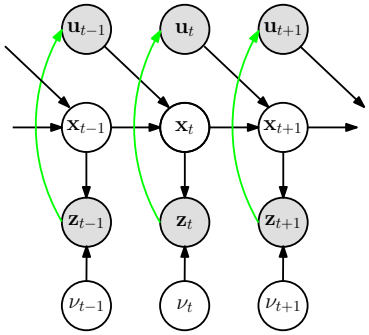
Recently, a new algorithm for sparse GPs was provided by Walder et al. (2008), who generalize the FITC approximation by Snelson and Ghahramani (2006) to the case where the basis functions centered at the inducing input locations can have different length-scales. However, we have not yet thoroughly investigated their multi-scale sparse GPs.



(a) The true problem is a POMDP with deterministic latent transitions. The hidden states $\mathbf{x}_t$ form a Markov chain. The measurements $\mathbf{z}_t$ of the hidden states are corrupted by additive Gaussian noise $\boldsymbol{\nu}$. The applied control signal is a function of the state $\mathbf{x}_t$ and is denoted by $\mathbf{u}_t$.



(b) Stochastic MDP. There are no latent states $\mathbf{x}$ in this model (in contrast to panel (a)). Instead it is assumed that the measured states $\mathbf{z}_t$ form a Markov chain and that the effect of the control signal $\mathbf{u}_t$ directly affects the measurement $\mathbf{z}_{t+1}$. The measurements are corrupted by Gaussian system noise $\boldsymbol{\varepsilon}$, which makes the assumed MDP stochastic.



(c) Implications to the true POMDP. The control decision $\mathbf{u}$ does not directly depend on the hidden state $\mathbf{x}$, but on the observed state $\mathbf{z}$. However, the control does affect the latent state $\mathbf{x}$ in contrast to the simplified assumption in panel (b). Thus, the measurement noise from panel (b) propagates through as system noise.

Figure 3.37: True POMDP, simplified stochastic MDP, and its implication to the true POMDP (without incurring costs). Hidden states, observed states, and control signals are denoted by $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{u}$, respectively. Panel (a) shows the true POMDP. Panel (b) is the graphical model when the simplifying assumption of the absence of a hidden layer is employed. This means, the POMDP with deterministic transitions is transformed into an MDP with stochastic transitions. Panel (c) illustrates the effect of this simplifying assumption to the true POMDP.

3.8.2 Noisy Measurements of the State

When working with hardware, such as the robotic unicycle in Section 3.7.4 or the hardware cart-pole system discussed in Section 3.7.1, we cannot assume that full and noise-free access to the state $\mathbf{x}$ is given: Typically, sensors measure the state (or parts of it), and these measurements $\mathbf{z}$ are noisy. In the following, we briefly discuss a simple extension of our approach that can deal with noisy measurements, if the noise is small. However, we still need full access to the state.

Let us consider the case where we no longer have direct access to the state $\mathbf{x}$. Instead, we receive a noisy measurement $\mathbf{z}$ of the state $\mathbf{x}$. In particular, we consider the state space model

$$\begin{aligned} \mathbf{x}_t &= f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \,, \\ \mathbf{z}_t &= \mathbf{x}_t + \boldsymbol{\nu}_t \,, \quad \boldsymbol{\nu}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\nu) \,, \end{aligned} \tag{3.48}$$

where $\boldsymbol{\nu}_t$ is white Gaussian measurement noise with uncorrelated dimensions. The corresponding graphical model is given in Figure 3.37(a). Note the difference to the graphical model we considered in the previous sections (Figure 3.1): The problem here is no longer an MDP, but a *partially observable Markov decision process* (POMDP). Inferring a generative model governing the latent Markov chain is a hard problem that is closely related to nonlinear system identification in a control context. If we assume the measurement function in equation (3.48) and a small covariance matrix $\boldsymbol{\Sigma}_\nu$ of the noise, we pretend the hidden layer of states $\mathbf{x}_t$ in Figure 3.37(a) does not exist. Thus, we approximate the POMDP by an MDP, where the (autoregressive) transition dynamics are given by

$$\mathbf{z}_t = \tilde{f}(\mathbf{z}_{t-1}, \mathbf{u}_{t-1}) + \boldsymbol{\varepsilon}_t \,, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\varepsilon) \,, \tag{3.49}$$

where $\boldsymbol{\varepsilon}_t$ is white independent Gaussian noise. The graphical model for this setup is shown in Figure 3.37(b). When the model in equation (3.49) is used, the control signal $\mathbf{u}_t$ is directly related to the (noisy) *observed* state $\mathbf{z}_t$, and no longer a function of the hidden state $\mathbf{x}_t$. Furthermore, in the model in Figure 3.37(b), the control signal $\mathbf{u}_t$ directly influences the consecutive observed state $\mathbf{z}_{t+1}$. Therefore, the noise in the observation at time t directly translates into noise in the control signal. If this noise is additive, the measurement noise $\boldsymbol{\nu}_t$ in equation (3.48) can be considered *system noise* $\boldsymbol{\varepsilon}_t$ in equation (3.49).

Hence, we approximate the POMDP in equation (3.48) by a stochastic MDP in equation (3.49). Figure 3.37(c) illustrates the effect of this simplified model on the true underlying POMDP in Figure 3.37(a). The control decision $\mathbf{u}_t$ is based on the observed state $\mathbf{z}_t$. However, unlike in the assumed model in Figure 3.37(b), the control in Figure 3.37(c) does not directly affect the next consecutive observed state $\mathbf{z}_{t+1}$, but only indirectly through the *hidden* state $\mathbf{x}_{t+1}$. When the simplified model in equation (3.49) is employed, both $\mathbf{z}_{t-1}$ *and* $\mathbf{z}_t$ can be considered samples, either from $\mathcal{N}(f(\mathbf{x}_{t-2}, \mathbf{u}_{t-2}), \boldsymbol{\Sigma}_\nu)$ or from $\mathcal{N}(f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}), \boldsymbol{\Sigma}_\nu)$. Thus, the variance of the noise $\boldsymbol{\varepsilon}$ in Figure 3.37(b) must be larger than the variance of the measurement noise $\boldsymbol{\nu}$ in Figure 3.37(c). In particular, $\boldsymbol{\Sigma}_\varepsilon = 2\,\boldsymbol{\Sigma}_\nu + 2\,\mathrm{cov}[\mathbf{z}_{t-1}, \mathbf{z}_t]$, which makes the learning problem harder compared to having direct access to the hidden state. Note that the measurements $\mathbf{z}_{t-1}$ and $\mathbf{z}_t$ are not uncorrelated since the noise $\boldsymbol{\varepsilon}_{t-1}$ in state $\mathbf{z}_{t-1}$ *does* affect $\mathbf{z}_t$ through the control signal $\mathbf{u}_{t-1}$ (Figure 3.37(c)).

Hence, the presented approach of approximating the POMDP with deterministic latent transitions by an MDP with stochastic transitions is only applicable if the covariance $\boldsymbol{\Sigma}_\nu$ is small and all state variables are measured.

3.9 Further Reading

General introductions to reinforcement learning and optimal control are given by Bertsekas and Tsitsiklis (1996), Bertsekas (2007), Kaelbling et al. (1996), and Sutton and Barto (1998).

In RL, we distinguish between direct and indirect learning algorithms. Direct (model-free) reinforcement learning algorithms include Q-learning proposed by Watkins (1989), TD-learning proposed by Barto et al. (1983), or SARSA proposed by Rummery and Niranjan (1994), which were originally not designed for continuous-valued state spaces. Extensions of model-free RL algorithms to continuous-valued state spaces are for instance the Neural Fitted Q-iteration by Riedmiller (2005) and, in a slightly more general form, the Fitted Q-iteration by Ernst et al. (2005). A drawback of model-free methods is that they typically require many interactions with the system/world to find a solution to the considered RL problem. In real-world problems, hundreds of thousands or millions of interactions with the system are often infeasible due to physical, time, and/or cost constraints. Unlike model-free methods, indirect (model-based) approaches can make more efficient use of limited interactions. The experience from these interactions is used to learn a model of the system, which can be used to generate arbitrarily much *simulated* experience. However, model-based methods may suffer if the model employed is not a sufficiently good approximation to the real world. This problem was discussed by Atkeson and Santamaría (1997) and Atkeson and Schaal (1997).

To overcome the problem of policies for inaccurate models, Abbeel et al. (2006) added a bias term to the model when updating the model after gathering real experience. Poupart and Vlassis (2008) learned a probabilistic model for a finite-state POMDP to incorporate observations into prior knowledge. This model was used in a value iteration scheme to determine an optimal policy. However, a principled and rigorous way of building non-parametric generative models that consistently quantify knowledge in continuous spaces, did not yet appear in the RL and/or control literature to our best knowledge. In our approach, we used flexible non-parametric probabilistic models to reap the benefit of the indirect approach while reducing the problems of model bias.

Traditionally, solving even relatively simple tasks in the absence of expert knowledge have been considered "daunting", (Schaal, 1997), in the absence of strong task-specific prior assumptions. In the context of robotics, one popular solution employs prior knowledge provided by a human expert to restrict the space of possible solutions. Successful applications of this kind of learning in control were described by Atkeson and Schaal (1997), Abbeel et al. (2006), Schaal (1997), Abbeel and Ng (2005), Peters and Schaal (2008b), Nguyen-Tuong et al. (2009), and Kober and Peters (2009). A human demonstrated a possible solution to the task. Subsequently, the policy was improved locally by using RL methods. This kind of learning is known as *learning from demonstration*, *imitation learning*, or *apprenticeship learning*.

Engel et al. (2003, 2005), Engel (2005), and Deisenroth et al. (2008, 2009) used probabilistic GP models to describe the RL value function. While Engel et al. (2003, 2005) and Engel (2005) focused on model-free TD-methods to approximate the value function, Deisenroth et al. (2008, 2009) focused on model-based algorithms in the context of dynamic programming/value iteration using GPs for the transition dynamics. Kuss (2006) and Rasmussen and Kuss (2004) considered model-based policy iteration using probabilistic dynamics models. Rasmussen and Kuss (2004) derived the policy from the value function, which was

modeled globally using GPs. The policy was not a parameterized function, but the actions at the support points of the value function model were directly optimized. Kuss (2006) additionally discussed Q-learning, where the Q-function was modeled by GPs. Moreover, Kuss proposed an algorithm without an explicit global model of the value function or the Q-function. He instead determined an open-loop sequence of T actions $(\mathbf{u}_1, \ldots, \mathbf{u}_T)$ that optimized the expected reward along a predicted trajectory.

Ng and Jordan (2000) proposed PEGASUS, a policy search method for large MDPs and POMDPs, where the transition dynamics were given by a stochastic model. Bagnell and Schneider (2001), Ng et al. (2004a), Ng et al. (2004b), and Michels et al. (2005) successfully incorporated PEGASUS into learning complicated control problems.

Indirect policy search algorithms often require the gradients of the value function with respect to the policy parameters. If the gradient of the value function with respect to the policy parameters cannot be computed analytically, it has to be estimated. To estimate the gradient, a range of policy gradient methods can be applied starting from a finite difference approximation of the gradient to more efficient gradient estimation using Monte-Carlo rollouts as discussed by Baxter et al. (2001). Williams (1992) approximated the value function V^π by the immediate cost and discounted future costs. Kakade (2002) and Peters et al. (2003) derived the *natural policy gradient*. A good overview of policy gradient methods with estimated gradients and their application to robotics is given in the work by Peters and Schaal (2006, 2008a,b) and Bhatnagar et al. (2009).

Several probabilistic models have been used to address the exploration-exploitation issue in RL. R-MAX by Brafman and Tennenholtz (2002) is a model-based RL algorithm that maintains a complete, but possibly inaccurate model of the environment and acts based on the model-optimal policy. R-MAX relies on the assumption that acting optimally with respect to the model results either in acting (close to) optimally according to the real world or in learning by encountering "unexpected" states. R-MAX distinguishes between "unknown" and "known" states and explores under the assumption that unknown states deliver maximum reward. Therefore, R-MAX uses an implicit approach to address the exploration/exploitation tradeoff as opposed to the E^3 algorithm by Kearns and Singh (1998). R-MAX assumes probabilistic transition dynamics, but is mainly targeted toward finite state-action domains, such as games. The myopic BOSS algorithm by Asmuth et al. (2009) samples multiple models from a posterior over models. These models were merged to an optimistic MDP via action space augmentation. A greedy policy was used for action selection. Exploration was essentially driven by maintaining the posterior over models. Similarly, Strens (2000) maintained a posterior distribution over models, sampled MDPs from it, and solved the MDPs via dynamic programming, which yielded an approximate distribution over best actions.

Murray-Smith et al. (2003), Kocijan et al. (2003), Kocijan et al. (2004), Grancharova et al. (2007), and Grancharova et al. (2008) used GPs for nonlinear system identification and model predictive control (receding horizon control). The uncertainty of the predictions was used in the context of robust and cautious control. In the context of RL, Ko et al. (2007) used GPs to learn the residuals between an idealized parameterized model and the observed data.

Approximate inference for planning purposes was proposed by Attias (2003). Here, planning was done by computing the posterior distribution over actions conditioned on reaching the goal state within a specified number of steps. Along with increasing computational power, planning via approximate inference is catching more and more attention. In the context of optimal control, Toussaint and Storkey (2006) and Toussaint (2008) used Bayesian inference to compute optimal policies.

Control of robotic unicycles has been studied for example by Naveh et al. (1999), Bloch et al. (2003), and Kappeler (2007), for example. In contrast to the unicycle system in the book by Bloch et al. (2003), we did *not* assume that the extension of the frame always passes through the contact point of the wheel with the ground. This simplifying assumption is equivalent to assuming that the frame cannot fall forward or backward. In 2008, the Murata Company designed the *MURATA GIRL*, a robot that could balance the unicycle[44].

[44] See `http://www.murata.com/new/news_release/2008/0923` and `http://www.murataboy.com/ssk-3/en/` for further information.

Chapter 4

Discussion

Current limitations. Our current approach learns very fast in terms of the amount of experience (interactions with the system) required to solve a task, but the computational demand is not negligible. In our current implementation, one policy search for a typically-sized data set takes on the order of thirty minutes CPU time on a standard PC. Performance can certainly be improved by writing more efficient and parallel code. Recently, graphics processing units (GPUs) have been shown promising for demanding computations in machine learning. Catanzaro et al. (2008) used them in the context of support vector machines whereas Raina et al. (2009) applied them to large-scale deep learning. Nevertheless, it is not obvious that our algorithms can necessarily learn in real time. However, once the policy has been learned, the computational requirements of applying the policy to a control task are fairly trivial and real-time capable as demonstrated in Section 3.7.1.

Strengths. The major strength of our framework is that it is general, coherent, and fully Bayesian. Therefore, it can be used to learn various episodic motor control tasks in the absence of expert knowledge; only general prior knowledge is required. The framework is based on fairly simple but well-understood approximations: For instance, all predictive distributions are approximated by Gaussian distributions, one of the simplest approximation one could make. The learning framework is successful not because of the Gaussian approximations, but because of the full incorporation of all kinds of uncertainty.

The proposed framework is not restricted to comprehensible mechanical control problems, but it can theoretically also be applied to control of complicated mechanical control systems, biological and chemical process control, and medical processes, for example. In these cases, we profit from the generality of the framework and from the fact that we do not rely on expert knowledge: Modeling slack, protein interaction, or responses of humans to drug treatments are examples, where non-parametric models can be superior to parametric approaches although the physical and biological interpretation is not given.

The three ingredients required for finding an optimal policy (see Figure 3.4), namely the probabilistic dynamics model, the saturating cost function, and the indirect policy search algorithm form a successful and efficient RL framework. Although it is difficult to tear them apart, we provided some evidence that the probabilistic dynamics model and in particular the incorporation of the model uncertainty into the decision-making process, are essential for successful learning.

Our learning framework was able to learn complicated nonlinear control tasks from scratch. To our best knowledge, we can achieve an unprecedented learning efficiency (in terms of required interactions) for either control task presented in this chapter. To our best knowledge, our learning framework is the first one that can learn the cart-double pendulum problem a) without expert knowledge and b) with only a single nonlinear controller. We demonstrated that the learning framework can directly be applied to hardware and tasks with multiple actuators.

Deterministic simulator. Although the model of the transition dynamics f in equation (3.1) is probabilistic, the internal simulation is fully deterministic: For a given policy parameterization and an initial state distribution $p(\mathbf{x}_0)$ the approximate inference computes predictive probability distributions deterministically and does not require any sampling. This property still holds if the transition dynamics f and/or the policy π are stochastic. Due to the deterministic simulative model, any optimization method for deterministic functions can be employed for the policy search.

Parameters. The parameters to be set for each task are essentially described in the upper half of Table C.1: the time discretization Δ_t, the width $1/a$ of the immediate cost, the exploration parameter b, and the prediction horizon. We give some rule-of-thumb heuristics how we chose these parameters although the algorithms are fairly robust against other parameter choices. The key problem is to find the right order of magnitude of the parameters.

The time discretization is set somewhat faster than the characteristic frequency of the system. The tradeoff with the Δ_t-parameter is that for small Δ_t the dynamics can be learned fairly easily, but more prediction steps are needed resulting in higher computational burden. Thus, we attempt to set Δ_t to a large value, which presumable requires more interaction time to learn the dynamics. The width of the saturating cost function should be set in a way that the cost function can easily distinguish between a "good" state and a "bad" state. Making the cost function too wide can result in numerical problems. In the experiments discussed in this dissertation, we typically set the exploration parameter to a small negative value, say, $b \in [-0.5, 0]$. Besides the exploration effect, a negative value of b smoothes the value function out and simplifies the optimization problem. First experiments with the cart-double pendulum indicate that a negative exploration parameter simplifies learning. Since we have no representative results yet, we do not discuss this issue thoroughly in this report. The (initial) prediction horizon T_{init} should be set in a way that the controller can approximately solve the task. Thus, T_{init} is also related to the characteristic frequency of the system. Since the computational effort increases linearly with the length of the prediction horizon, shorter horizons are desirable in the early stages of learning when the dynamics model is still fairly poor. Furthermore, the learning task is difficult for long horizons since it is easy to lose track of the state in the early stages of learning.

Value function model. Our learning algorithm does not use an explicit model of the value function V^π. Instead, it evaluates the value function for a finite set of initial states $\mathbf{x}_0$. Although global value function models are often used for efficiently deriving an optimal policy, they are an additional source of errors: Optimal value functions are often discontinuous in the case of deterministic transitions. Most (linear) function approximators smooth these discontinuities out. Although in many interesting cases, an optimal policy is related to the gradient of the optimal value function as discussed by Bertsekas (2005), the direct consequence of an error in the value function model on the policy is often unclear. It is true that the optimal policy is also discontinuous. However, there often exists a close-to-optimal policy in the class of smooth functions.

Linearity of the policy. Strictly speaking, the policy based on a linear model (see equation (3.9)) is nonlinear since we squash the linear function (the preliminary policy) through the sine function to account for constrained control signals in a fully Bayesian way (we can compute predictive distributions after squashing the preliminary policy).

Noise in the policy training set and policy parameterization. The pseudo-training targets $\mathbf{y}_\pi = \tilde{\pi}(\mathbf{X}_\pi) + \boldsymbol{\varepsilon}_\pi,\ \boldsymbol{\varepsilon}_\pi \sim \mathcal{N}(0, \boldsymbol{\Sigma}_\pi)$, for the RBF policy in equation (3.10) are considered noisy. We optimize the training targets (and the corresponding noise variance), such that the fitted RBF policy minimizes the expected long-term cost in equation (3.2) or likewise in equation (3.25). The pseudo-targets $\mathbf{y}_\pi$ do not necessarily have to be noisy since they are not real observations. However, noisy pseudo-targets smooth the latent function out and make the optimization problem, that is, the policy search, easier.

The parameterization of the RBF policy via the mean function of a GP is unusual. A "standard" RBF is usually given as

$$\sum_{i=1}^{N} \beta_i k(\mathbf{x}_i, \mathbf{x}_*)\,, \tag{4.1}$$

where k is a Gaussian basis function with mean $\mathbf{x}_i$; β_i are coefficients, and $\mathbf{x}_*$ is a test point. The parameters in this representation are simply the values β_i, the means $\mathbf{x}_i$, and the widths of the Gaussian basis functions. In our representation (see equation (3.10)), the vector $\boldsymbol{\beta}$ of coefficients is not directly determined, but indirectly since it depends on the (inverse) kernel matrix, the training targets, and the noise levels $\boldsymbol{\Sigma}_\pi$. This leads to an over-parameterization with one parameter, which corresponds to the signal-to-noise ratio. Also, the dependency on the inverse kernel matrix (plus a noise ridge) can lead to numerical instabilities. However, algebraically the standard RBF parameterization via the mean

function of the GP is as expressive as the standard RBF parameterization in equation (4.1): since the matrix $\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I}$ has full rank. Despite the over-parameterization, in our experiments overfitting did not happen. It is not clear yet whether treating β_i as direct parameters makes the optimization easier since we have not yet investigated this issue thoroughly.

Different policy representations. We did not thoroughly investigate other (preliminary) policy representations than a linear function and an RBF network. Alternative representations include Gaussian processes and neural networks (with cumulative Gaussian activation functions). Note, however, that any representation of a preliminary policy must satisfy the constraints discussed in Section 3.4.2, which include the analytic computation of the mean and the variance of the preliminary policy if the state is Gaussian distributed. The optimization of the policy parameters could profit from a GP policy (with a fixed number of basis functions) due to the smoothing effect of the model uncertainty, which is not present in the RBF policy representation used in this report. First results with the GP policy are looking promising.

Model of the transition dynamics. From a practical perspective, the main challenge in learning for control seems to be finding a good model of the transition dynamics, which can be used for internal simulation. Many model-based RL algorithms can be applied when an "accurate" model is given. However, if the model does not coherently describe the system, the policy found by the RL algorithm can be arbitrarily bad. The probabilistic GP model appropriately represents the transition dynamics: Since the dynamics GP can be considered a distribution over all models that plausibly explain the experience (collected in the training set), incorporation of novel experience does usually[1] not make previously plausible models implausible. By contrast, if a deterministic model is used, incorporation of novel experience always changes the model and, therefore, makes plausible models implausible and vice versa. We observed that this model change can have a strong influence on the optimization procedure and is an additional reason why deterministic models and gradient-based policy search algorithms do not fit well together.

The dynamics GP model, which models the general input-output behavior can be considered an efficient machine learning approach to non-parametric system identification. All involved parameters are implicitly determined. A drawback (from a system engineer's point of view) of a GP is that the hyper-parameters in a non-parametric model do not usually yield a mechanical or physical interpretation.

If some parameters in system identification cannot be determined with certainty, classic robust control (minimax/$\mathcal{H}_\infty$-control) aims to minimize the worst-case error. This methodology often leads to suboptimal and conservative solutions. Possibly, a fully probabilistic Gaussian process model of the system dynamics can be incorporated as follows: As the GP model coherently describes the uncertainty about the underlying function, it implicitly covers all transition dynamics that plausibly explain observed data. By Bayesian averaging over all these models, we appropriately treat uncertainties and can potentially bridge the gap between optimal and robust control. GPs for system identification and robust model predictive control have been employed for example by Kocijan et al. (2004), Murray-Smith et al. (2003), Murray-Smith and Sbarbaro (2002), Grancharova et al. (2008), or Kocijan and Likar (2008).

Moment matching approximation of densities. Let q_1 be the approximate Gaussian distribution that is analytically computed by moment matching using the results from Section 2.3.2. The moment matching approximation minimizes the Kullback-Leibler (KL) divergence $\mathrm{KL}(p||q_1)$ between the true predictive distribution p and its approximation q_1. Minimizing $\mathrm{KL}(p||q_1)$ ensures that q_1 is non-zero where the true distribution p is non-zero. This is an important issue in the context of coherent predictions and, therefore, robust control: The approximate distribution q_1 is not overconfident, but can be too cautious since it tries to capture all of the modes of the true distribution as shown by Kuss and Rasmussen (2006). However, if we can learn a controller using the admittedly conservative moment-matching approximation, the controller is expected to be robust. By contrast, a variational approximate distribution q_2 that minimizes the KL divergence $\mathrm{KL}(q_2||p)$ ensures that q_2 is zero where p is zero. This approximation often leads to overconfident results and is presumably not well-suited for optimal and/or robust control. More information and insight about the KL divergence and approximate distributions is given in the book by

[1] In the very early stages of learning, it can happen that the model substantially changes when data are added that are far away from the old training set.

Bishop (2006, Chapter 10.1). The moment-matching approximation employed is equivalent to a unimodal approximation using Expectation Propagation (Minka, 2001).

Unimodal distributions are usually fairly bad representations of state distributions at symmetry-breaking points. Consider for example a pendulum in the inverted position: It can fall to the left and to the right with equal probability. We approximate this bimodal distribution by a Gaussian with high variance. When we predict, we have to propagate this Gaussian forward and we lose track of the state very quickly. However, we can control the state by applying actions to the system. We are interested in minimizing the expected long-term cost. High variances are therefore not favorable in the long term. Our experience is that the controller ensures that it decides on one mode and completely ignores the other mode of the bimodal distribution. Hence, the symmetry can be broken by applying actions to the system.

The projection of the predictive distribution of a GP with uncertain inputs onto a unimodal Gaussian is a simplification in general since the true distribution can easily be multi-modal (see Figure 2.6). If we want to consider and propagate multi-modal distributions in a time series, we need to compute a multi-modal predictive distribution from a multi-modal input distribution. Suppose a multi-modal distribution $p(\mathbf{x})$. It is possible to compute a multi-modal predictive distribution following the results from 2.3.2 for each mode. Expectation Correction by Barber (2006) leads into this direction. However, the multi-modal predictive distribution is generally not an optimal[2] multi-modal approximation of the true predictive distribution. Finding an optimal multi-modal approximation of the true distribution is an open research problem. Only in the unimodal case, we can easily find a unimodal approximation of the predictive distribution in the exponential family that minimizes the Kullback-Leibler divergence between the true distribution and the approximate distribution: the Gaussian with the mean and the covariance of the true predictive distribution.

Separation of system identification and control. Our probabilistic framework iterates between system identification (learning the dynamics model) and optimization of the controller parameters, where we condition on the probabilistic model of the transition dynamics. This approach contrasts many traditional approaches in control, where the controller optimization is deterministically conditioned on the learned model, that is, a point estimate of the model parameters is employed when optimizing the controller parameters.

Incorporation of prior knowledge. Prior knowledge about the policy and the transition dynamics can be incorporated easily: A good-guess policy or a demonstration of the task can be used instead of a random initialization of the policy. In a practical application, if idealized transition dynamics are known, they can be used as a prior mean function as proposed for example by Ko et al. (2007) and Ko and Fox (2008, 2009a,b) in the context of RL and mechanical control. In this report, we used a prior mean that was zero everywhere.

Curriculum learning. Humans and animals learn much faster when they learn in small steps: A complicated task can be learned faster if it is similar to an easy task that has been learned before. In the context of machine learning, Bengio et al. (2009) call this type of learning *curriculum learning*. Curriculum learning can be considered a continuation method across tasks. In continuation methods, a difficult optimization problem is solved by first solving a much simpler initial problem and then, step by step, shaping the simple problem into the original problem by tracking the solution to the optimization problem. Details on continuation methods can be found in the paper by Richter and DeCarlo (1983). Bengio et al. (2009) hypothesize that curriculum learning improves the speed of learning and the quality of the solution to the complicated task.

In the context of learning motor control, we can apply curriculum learning by learning a fairly easy control task and initialize a more difficult control task with the solution of the easy problem. For example, if we first learn to swing up and balance a single pendulum, we can exploit this knowledge when learning to swing up and balance a double-pendulum. Curriculum learning for control tasks remains to future work.

[2] In this context, "optimality" corresponds to a minimal Kullback-Leibler divergence between the true distribution and the approximate distribution.

Extension to partially observable Markov decision processes. We have demonstrated learning in the special case where we assume that the full state is measured. In principle, there is nothing to hinder the use of the algorithm when not all state variables are observed and the measurements $\mathbf{z}$ are noisy. In this case, we need to learn a generative model for the latent state Markovian process

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

and the measurement function g

$$\mathbf{z}_t = g(\mathbf{x}_t) + \mathbf{v}_t \,,$$

where $\mathbf{v}_t$ is a noise term. Suppose a GP models for f is given. For the internal simulation of the system (Figure 3.2(b) and intermediate layer in Figure 3.3), our learning framework can be applied without any practical changes: We simply need to predict multiple steps ahead when the initial state is uncertain—this is done already in the current implementation. The difference is simply where the initial state distribution originates from. Right now, it represents a set of possible initial states; in the POMDP case it would be the prior on the initial state. During the interaction phase (see Figure 3.2(a)), where we obtain noisy and partial measurements of the latent state $\mathbf{x}_t$, it is advantageous to update the predicted state distribution $p(\mathbf{x}_t)$ using the measurements $\mathbf{z}_{1:t}$. To do so, we require efficient filtering algorithms suitable for GP transition functions and potentially GP measurement functions. The distribution $p(\mathbf{x}_t|\mathbf{o}bs_{1:t})$ can be used to compute a control signal applied to the system (for example the mean of this distribution). The remaining problem is to determine the GP models for the transition function f (and potentially the measurement function g). This problem corresponds to nonlinear (non-parametric) system identification. Parameter learning and system identification go beyond the scope of this report and are left to future work.

Chapter 5

Summary

We proposed a general framework for efficient model-based reinforcement learning in the context of motor control problems. The key ingredient of this framework is a fully probabilistic model for the transition dynamics that mimics two important features of biological learners: the ability to generalize and the explicit incorporation of uncertainty into the decision-making process. From a control perspective, the learning framework touches upon (non-parametric) nonlinear system identification for control, model-based nonlinear optimal control, robust control, iterative learning control, and dual control.

The proposed framework is conceptually simple and relies on well-established ideas. A decisive difference to common RL algorithms including the Dyna architecture proposed by Sutton (1990) is that we explicitly require fully non-degenerate probabilistic models of the transition dynamics for coherent predictions and to reduce model bias. We presented an efficient implementation of the framework based on Gaussian process models. The strengths of our method is its ability to learn dynamics models and policies for fairly complicated control tasks in the absence of task-specific prior knowledge. This makes our RL framework generally applicable to episodic tasks with continuous states and actions. The success of our algorithm stems from the principled approach to handling the model's uncertainty.

Our learning framework was successfully applied to controlling several inherently unstable nonlinear dynamic systems, the cart-pole problem, the Pendubot, the cart-double pendulum problem, and the robotic unicycle. The same learning framework was applied to all tasks by simply modifying some high-level parameters such as the time discretization. We showed that our framework can handle multivariate controls in a principled way. The proposed approach can be directly applied to hardware as shown for the cart-pole problem. Across all tasks, we reported an unprecedented speed of learning in the absence of expert knowledge. For example, the cart-pole swing up plus balancing required less than thirty seconds of experience. For balancing a robotic unicycle, a dynamics model and a controller were learned using data from less than a minute's experience. Moreover, to our best knowledge, our learning framework is the first one that can learn the cart-double pendulum problem a) without expert knowledge and b) with a single nonlinear controller for both swing up and balancing.

The RL framework can naturally be applied to episodic learning tasks with continuous states and actions. Therefore, it seems to fit nicely in the context of *iterative learning control*, where "a system that executes the same task multiple times can be improved by learning from previous executions (trials, iterations, passes)" (Bristow et al., 2006).

Appendix A

Some Mathematical Tools

A.1 Integration

This section gives exact integral equations for trigonometric functions, which are required to implement the discussed algorithms. The following expressions can be found in the book by Gradshteyn and Ryzhik (2000), where $x \sim \mathcal{N}(\mu, \sigma^2)$ is Gaussian distributed with mean μ and variance σ^2.

$$\begin{aligned}
\mathbb{E}_x[\sin(x)] &= \int \sin(x) p(x)\,\mathrm{d}x = \exp(-\tfrac{\sigma^2}{2})\sin(\mu)\,, \\
\mathbb{E}_x[\cos(x)] &= \int \cos(x) p(x)\,\mathrm{d}x = \exp(-\tfrac{\sigma^2}{2})\cos(\mu)\,, \\
\mathbb{E}_x[\sin(x)^2] &= \int \sin(x)^2 p(x)\,\mathrm{d}x = \tfrac{1}{2}\big(1 - \exp(-2\sigma^2)\cos(2\mu)\big)\,, \\
\mathbb{E}_x[\cos(x)^2] &= \int \cos(x)^2 p(x)\,\mathrm{d}x = \tfrac{1}{2}\big(1 + \exp(-2\sigma^2)\cos(2\mu)\big)\,, \\
\mathbb{E}_x[\sin(x)\cos(x)] &= \int \sin(x)\cos(x) p(x)\,\mathrm{d}x = \int \tfrac{1}{2}\sin(2x) p(x)\,\mathrm{d}x = \tfrac{1}{2}\exp(-2\sigma^2)\sin(2\mu)\,.
\end{aligned}$$

Gradshteyn and Ryzhik (2000) also provide a more general solution to an integral involving squared exponentials, polynomials, and trigonometric functions,

$$\begin{aligned}
&\int x^n \exp\big(-(ax^2 + bx + c)\big)\sin(px + q)\,\mathrm{d}x \\
&\quad = -\left(\frac{-1}{2a}\right)^n \sqrt{\frac{\pi}{a}} \exp\left(\frac{b^2 - p^2}{4a} - c\right) \\
&\qquad \times \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{n!}{(n-2k)!k!} a^k \sum_{j=0}^{n-2k} \binom{n-2k}{j} b^{n-2k-j} p^j \sin\left(\frac{pb}{2a} - q + \frac{\pi}{2} j\right)\,, \quad a > 0\,, \\
&\int x^n \exp\big(-(ax^2 + bx + c)\big)\cos(px + q)\,\mathrm{d}x \\
&\quad = \left(\frac{-1}{2a}\right)^n \sqrt{\frac{\pi}{a}} \exp\left(\frac{b^2 - p^2}{4a} - c\right) \\
&\qquad \times \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{n!}{(n-2k)!k!} a^k \sum_{j=0}^{n-2k} \binom{n-2k}{j} p^j \cos\left(\frac{pb}{2a} - q + \frac{\pi}{2} j\right)\,, \quad a > 0\,.
\end{aligned}$$

A.2 Differentiation Rules

Let $\mathbf{A}, \mathbf{B}, \mathbf{K}$ be matrices of appropriate dimensions and $\boldsymbol{\theta}$ a parameter vector. We re-state results from the book by Petersen and Pedersen (2009) to compute derivatives of products, inverses, determinants,

and traces of matrices with respect to $\boldsymbol{\theta}$.

$$\begin{aligned}
\frac{\partial|\mathbf{K}(\boldsymbol{\theta})|}{\partial\boldsymbol{\theta}} &= |\mathbf{K}|\mathrm{tr}\left(\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\boldsymbol{\theta}}\right)\\
\frac{\partial|\mathbf{K}|}{\partial\mathbf{K}} &= |\mathbf{K}|(\mathbf{K}^{-1})^\top\\
\frac{\partial\mathbf{K}^{-1}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} &= -\mathbf{K}^{-1}\frac{\partial\mathbf{K}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\mathbf{K}^{-1}\\
\frac{\partial\boldsymbol{\theta}^\top\mathbf{K}\boldsymbol{\theta}}{\partial\boldsymbol{\theta}} &= (\mathbf{K}+\mathbf{K}^\top)\boldsymbol{\theta}\\
\frac{\partial\mathrm{tr}(\mathbf{AKB})}{\partial\mathbf{K}} &= \mathbf{A}^\top\mathbf{B}^\top
\end{aligned}$$

A.3 Properties of Gaussians

Assume $\mathbf{x}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$ is Gaussian distributed with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Then,

$$p(\mathbf{x}) \coloneqq (2\pi)^{-\frac{D}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}}\exp\left((\mathbf{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

where $\mathbf{x}\in\mathbb{R}^D$. The *marginals* of the joint

$$p(\mathbf{x}_1,\mathbf{x}_2) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix},\begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12}\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\right)$$

are given as the "sliced-out" distributions

$$\begin{aligned}
p(\mathbf{x}_1) &= \mathcal{N}(\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_{11})\\
p(\mathbf{x}_2) &= \mathcal{N}(\boldsymbol{\mu}_2,\boldsymbol{\Sigma}_{22}),
\end{aligned}$$

respectively. The *conditional* distribution of $\mathbf{x}_1$ given $\mathbf{x}_2$ is

$$p(\mathbf{x}_1|\mathbf{x}_2) = \mathcal{N}(\boldsymbol{\mu}_1+\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2-\boldsymbol{\mu}_2),\boldsymbol{\Sigma}_{11}-\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}).$$

The *product* of two Gaussians $\mathcal{N}(\mathbf{a},\mathbf{A})\mathcal{N}(\mathbf{b},\mathbf{B})$ is an unnormalized Gaussian $c\mathcal{N}(\mathbf{c},\mathbf{C})$ with

$$\begin{aligned}
\mathbf{C} &= (\mathbf{A}^{-1}+\mathbf{B}^{-1})^{-1}\\
\mathbf{c} &= \mathbf{C}(\mathbf{A}^{-1}\mathbf{a}+\mathbf{B}^{-1}\mathbf{b})\\
c &= (2\pi)^{-\frac{D}{2}}|\mathbf{A}+\mathbf{B}|^{-\frac{1}{2}}\exp\left(-\tfrac{1}{2}(\mathbf{a}-\mathbf{b})^\top(\mathbf{A}+\mathbf{B})^{-1}(\mathbf{a}-\mathbf{b})\right).
\end{aligned}$$

Note that the normalizing constant c itself is a Gaussian either in $\mathbf{a}$ or in $\mathbf{b}$ with an "inflated" covariance matrix $\mathbf{A}+\mathbf{B}$.

A Gaussian distribution in $\mathbf{Ax}$ can be transformed into an unnormalized Gaussian distribution in $\mathbf{x}$ by re-arranging the means according to

$$\begin{aligned}
\mathcal{N}(\mathbf{Ax}\,|\,\boldsymbol{\mu},\boldsymbol{\Sigma}) &= c_1\mathcal{N}(\mathbf{x}\,|\,\mathbf{A}^{-1}\boldsymbol{\mu},(\mathbf{A}^\top\boldsymbol{\Sigma}^{-1}\mathbf{A})^{-1}), \qquad (A.1)\\
c_1 &= \frac{\sqrt{|2\pi(\mathbf{A}^\top\boldsymbol{\Sigma}^{-1}\mathbf{A})^{-1}|}}{\sqrt{|2\pi\boldsymbol{\Sigma}|}}.
\end{aligned}$$

A.4 Matrix Inversion

To avoid explicit inversion of a possibly singular matrix, we often employ the following two identities:

$$(\mathbf{A}^{-1}+\mathbf{B}^{-1})^{-1} = \mathbf{A}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{B} = \mathbf{B}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{A} \qquad (A.2)$$

$$(\mathbf{Z}+\mathbf{UWV}^\top)^{-1} = \mathbf{Z}^{-1}-\mathbf{Z}^{-1}\mathbf{U}(\mathbf{W}^{-1}+\mathbf{V}^\top\mathbf{Z}^{-1}\mathbf{U})^{-1}\mathbf{V}^\top\mathbf{Z}^{-1} \qquad (A.3)$$

$$(\mathbf{A}+\mathbf{BC})^{-1} = \mathbf{A}^{-1}-\mathbf{A}^{-1}\mathbf{B}(\mathbf{I}+\mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1}. \qquad (A.4)$$

The Searle identity in equation (A.2) is useful if the individual inverses of $\mathbf{A}$ and $\mathbf{B}$ do not exist or if they are ill conditioned. The Woodbury identity in equation (A.3) can be used to reduce the computational burden: If $\mathbf{Z} \in \mathbb{R}^{p\times p}$ is diagonal, the inverse $\mathbf{Z}^{-1}$ can be computed in $\mathcal{O}(p)$. Consider the case where $\mathbf{U} \in \mathbb{R}^{p\times q}$, $\mathbf{W} \in \mathbb{R}^{q\times q}$, and $\mathbf{V}^\top \in \mathbb{R}^{q\times p}$ with $p \gg q$. The inverse $(\mathbf{Z} + \mathbf{UWV}^\top)^{-1}$ would require $\mathcal{O}(p^3)$ computations. Using equation (A.3), the computational burden reduces to $\mathcal{O}(p)$ for the inverse of the diagonal $\mathbf{Z}$ plus $\mathcal{O}(q^3)$ for the inverse of $\mathbf{W}$ and the inverse of $(\mathbf{W}^{-1} + \mathbf{V}^\top\mathbf{Z}^{-1}\mathbf{U})$. Note that this bracket is a $q\times q$ matrix and that $q \ll p$. Therefore, the inversion of a $p\times p$ matrix can be reduced to the inversion of $q \times q$ matrixes and some matrix multiplications. The Kailath inverse in equation (A.4) is helpful if $\mathbf{A}$ has very large entries. Consider the following example: Let $\mathbf{A}$ be the (diagonal) matrix of length-scales in the SE kernel from equation (2.2). In an almost linear model, some of the length-scales can become very big, which causes the matrix $\mathbf{\Lambda} + \mathbf{BC}$ to be ill conditioned. The Kailath inverse alleviates this problem.

Appendix B

Equations of Motion

B.1 Cart Pole (Inverted Pendulum)

The inverted pendulum shown in Figure B.1 consists of a cart with mass m_1 and an attached pedulum with mass m_2 and length l, which swings freely in the plane. The pendulum angle θ_2 is measured anti-clockwise from hanging down. The cart can move horizontally with an applied external force u and a parameter b, which describes the friction between cart and ground. Typical values are: $m_1 = 0.5\,\mathrm{kg}$, $m_2 = 0.5\,\mathrm{kg}$, $l = 0.6\,\mathrm{m}$ and $b = 0.1\,\mathrm{N/m/s}$.

The position of the cart along the track is denoted by x_1. The coordinates x_2 and y_2 of the midpoint of the pendulum are

$$\begin{aligned} x_2 &= x_1 + \tfrac{1}{2} l \sin\theta_2\,, \\ y_2 &= -\tfrac{1}{2} l \cos\theta_2, \end{aligned}$$

and the squared velocity of the cart and the midpoint of the pendulum are

$$\begin{aligned} v_1^2 &= \dot{x}_1^2 \\ v_2^2 &= \dot{x}_2^2 + \dot{y}_2^2 = \dot{x}_1^2 + \tfrac{1}{4} l^2 \dot{\theta}_2^2 + l\dot{x}_1\dot{\theta}_2 \cos\theta_2\,, \end{aligned}$$

respectively. We derive the equations of motion via the system Lagrangian L, which is the difference between kinetic energy T and potential energy V and given by

$$L = T - V = \tfrac{1}{2} m_1 v_1^2 + \tfrac{1}{2} m_2 v_2^2 + \tfrac{1}{2} I \dot{\theta}_2^2 - m_2 g y_2\,, \tag{B.1}$$

where $g = 9.82\,\mathrm{m/s^2}$ is the acceleration of gravity and $I = \frac{1}{12} m l^2$ is the moment of inertia of a thin pendulum around the pendulum midpoint. Pluggin this value for I into the system Lagrangian (B.1), we obtain

$$L = \tfrac{1}{2}(m_1 + m_2)\dot{x}_1^2 + \tfrac{1}{6} m_2 l^2 \dot{\theta}_2^2 + \tfrac{1}{2} m_2 l(\dot{x}_1\dot{\theta}_2 + g)\cos\theta_2\,.$$

The equations of motion can generally be derived from a set of equations defined through

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i\,,$$

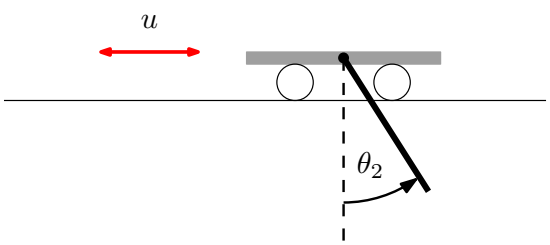


Figure B.1: Cart-pole system (inverted pendulum).

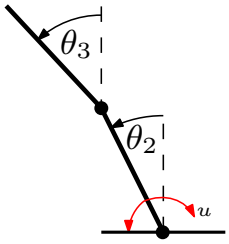


Figure B.2: Pendubot.

where Q_i are the non-conservative forces and q_i and $\dot{q}_i$ are the state variables of the system. In our case,

$$\begin{aligned}
\frac{\partial L}{\partial \dot{x}_1} &= (m_1 + m_2)\dot{x}_1 + \tfrac{1}{2} m_2 l \dot{\theta}_2 \cos\theta_2 \,, \\
\frac{\partial L}{\partial x_1} &= 0 \,, \\
\frac{\partial L}{\partial \dot{\theta}_2} &= \tfrac{1}{3} m_2 l^2 \dot{\theta}_2 + \tfrac{1}{2} m_2 l \dot{x}_1 \cos\theta_2 \,, \\
\frac{\partial L}{\partial \theta_2} &= -\tfrac{1}{2} m_2 l (\dot{x}_1 \dot{\theta}_2 + g),
\end{aligned}$$

lead to the equations of motion

$$\begin{aligned}
(m_1 + m_2)\ddot{x}_1 + \tfrac{1}{2} m_2 l \ddot{\theta}_2 \cos\theta_2 - \tfrac{1}{2} m_2 l \dot{\theta}_2^2 \sin\theta_2 &= u - b\dot{x}_1 \,, \\
2l\ddot{\theta}_2 + 3\ddot{x}_1 \cos\theta_2 + 3g \sin\theta_2 &= 0 \,.
\end{aligned}$$

Collecting the four variables $\mathbf{z} = [x_1, \dot{x}_1, \dot{\theta}_2, \theta_2]^\top$ the equations of motion can be conveniently expressed as four coupled ordinary differential equations

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \begin{bmatrix}
z_2 \\
\dfrac{2m_2 l z_3^2 \sin z_4 + 3m_2 g \sin z_4 \cos z_4 + 4u - 4bz_2}{4(m_1 + m_2) - 3m_2 \cos^2 z_4} \\
\dfrac{-3m_2 l z_3^2 \sin z_4 \cos z_4 - 6(m_1 + m_2) g \sin z_4 - 6(u - bz_2)\cos z_4}{4l(m_1 + m_2) - 3m_2 l \cos^2 z_4} \\
z_3
\end{bmatrix},$$

which can be simulated numerically.

B.2 Pendubot

The Pendubot in Figure B.2 is a two-link (mass m_2 and m_3 and length s l_2 and l_3 respectively), underactuated robot as described by Spong and Block (1995). The first joint exerts torque, but the second joint cannot. The system has four continuous state variables: two joint positions and two joint velocities. The angles of the joints, θ_2 and θ_3, are measured anti-clockwise from upright. An applied external torque u controls the first joint. Typical values are: $m_2 = 0.5$kg, $m_3 = 0.5$kg $l_2 = 0.6$m, $l_3 = 0.6$m.

The Cartesian coordinates x_2, y_2 and x_3, y_3 of the midpoints of the pendulum elements are

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} l_2 \sin\theta_2 \\ \frac{1}{2} l_2 \cos\theta_2 \end{bmatrix}, \quad \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} -l_2 \sin\theta_2 - \frac{1}{2} l_3 \sin\theta_3 \\ l_2 \cos\theta_2 + \frac{1}{2} l_3 \cos\theta_3 \end{bmatrix},$$

and the squared velocities of the pendulum midpoints are

$$v_2^2 = \dot{x}_2^2 + \dot{y}_2^2 = \tfrac{1}{4} l_2^2 \dot{\theta}_2^2 \tag{B.2}$$

$$v_3^2 = \dot{x}_3^2 + \dot{y}_3^2 = l_2^2 \dot{\theta}_2^2 + \tfrac{1}{4} l_3^2 \dot{\theta}_3^2 + l_2 l_3 \dot{\theta}_2 \dot{\theta}_3 \cos(\theta_2 - \theta_3). \tag{B.3}$$

The system Lagrangian is the difference between the kinematic energy T and the potential energy V and given by

$$L = T - V = \tfrac{1}{2}m_2v_2^2 + \tfrac{1}{2}m_3v_3^2 + \tfrac{1}{2}I_2\dot{\theta}_2^2 + \tfrac{1}{2}I_3\dot{\theta}_3^2 - m_2gy_2 - m_3gy_3\,,$$

where the angular moment of inertia around the pendulum midpoint is $I = \frac{1}{12}ml^2$, and $g = 9.82\text{m/s}^2$ is the acceleration of gravity. Using this moment of inertia, we assume that the pendulum is an infinitely thin (but rigid) wire. Plugging in the squared velocities (B.2) and (B.3), we obtain

$$\begin{aligned} L = {} & \tfrac{1}{8}m_2l_2^2\dot{\theta}_2^2 + \tfrac{1}{2}m_3\big(l_2^2\dot{\theta}_2^2 + \tfrac{1}{4}l_3^2\dot{\theta}_3^2 + l_2l_3\dot{\theta}_2\dot{\theta}_3\cos(\theta_2-\theta_3)\big) \\ & + \tfrac{1}{2}I_2\dot{\theta}_2^2 + \tfrac{1}{2}I_3\dot{\theta}_3^2 - \tfrac{1}{2}m_2gl_2\cos\theta_2 - m_3g(l_2\cos\theta_2 + \tfrac{1}{2}l_3\cos\theta_3)\,. \end{aligned}$$

The equations of motion are

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i\,,$$

where Q_i are the non-conservative forces and q_i and $\dot{q}_i$ are the state variables of the system. In our case,

$$\begin{aligned} \frac{\partial L}{\partial \dot{\theta}_2} &= l_2^2\dot{\theta}_2(\tfrac{1}{4}m_2 + m_3) + \tfrac{1}{2}m_3l_2l_3\dot{\theta}_3\cos(\theta_2-\theta_3) + I_2\dot{\theta}_2\,, \\ \frac{\partial L}{\partial \theta_2} &= -\tfrac{1}{2}m_3l_2l_3\dot{\theta}_2\dot{\theta}_3\sin(\theta_2-\theta_3) + (\tfrac{1}{2}m_2 + m_3)gl_2\sin\theta_2\,, \\ \frac{\partial L}{\partial \dot{\theta}_3} &= m_3l_3\big(\tfrac{1}{4}l_3\dot{\theta}_3 + \tfrac{1}{2}l_2\dot{\theta}_2\cos(\theta_2-\theta_3)\big) + I_3\dot{\theta}_3\,, \\ \frac{\partial L}{\partial \theta_3} &= \tfrac{1}{2}m_3l_3\big(l_2\dot{\theta}_2\dot{\theta}_3\sin(\theta_2-\theta_3) + g\sin\theta_3\big) \end{aligned}$$

lead to the equations of motion

$$\begin{aligned} \ddot{\theta}_2\big(l_2^2(\tfrac{1}{4}m_2 + m_3) + I_2\big) + \ddot{\theta}_3\tfrac{1}{2}m_3l_3l_2\cos(\theta_2-\theta_3) \\ + l_2\big(\tfrac{1}{2}m_3l_3\dot{\theta}_3^2\sin(\theta_2-\theta_3) - g\sin\theta_2(\tfrac{1}{2}m_2 + m_3)\big) = u\,, \\ \ddot{\theta}_2\tfrac{1}{2}l_2l_3m_3\cos(\theta_2-\theta_3) + \ddot{\theta}_3(\tfrac{1}{4}m_3l_3^2 + I_3) - \tfrac{1}{2}m_3l_3\big(l_2\dot{\theta}_2^2\sin(\theta_2-\theta_3) + g\sin\theta_3\big) = 0\,. \end{aligned}$$

To simulate the system numerically, we solve the linear equation system

$$\begin{bmatrix} l_2^2(\frac{1}{4}m_2 + m_3) + I_2 & \frac{1}{2}m_3l_3l_2\cos(\theta_2-\theta_3) \\ \frac{1}{2}l_2l_3m_3\cos(\theta_2-\theta_3) & \frac{1}{4}m_3l_3^2 + I_3 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} = \begin{bmatrix} c_2 \\ c_3 \end{bmatrix}$$

for $\ddot{\theta}_2$ and $\ddot{\theta}_3$, where

$$\begin{bmatrix} c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} -l_2\big(\frac{1}{2}m_3l_3\dot{\theta}_3^2\sin(\theta_2-\theta_3) - g\sin\theta_2(\frac{1}{2}m_2 + m_3)\big) + u \\ \frac{1}{2}m_3l_3\big(l_2\dot{\theta}_2^2\sin(\theta_2-\theta_3) + g\sin\theta_3\big) \end{bmatrix}.$$

B.3 Cart-Double Pendulum

The cart-double pendulum dynamic system (see Figure B.3) consists of a cart with mass m_1 and an attached double pendulum with masses m_2 and m_3 and lengths l_2 and l_3 for the two links, respectively. The double pendulum swings freely in the plane. The angles of the pendulum, θ_2 and θ_3, are measured anti-clockwise from upright. The cart can move horizontally, with an applied external force u and the coefficient of friction b. Typical values are: $m_1 = 0.5\,\text{kg}$, $m_2 = 0.5\,\text{kg}$, $m_3 = 0.5\,\text{kg}$ $l_2 = 0.6\,\text{m}$, $l_3 = 0.6\,\text{m}$, and $b = 0.1\,\text{Ns/m}$.

The coordinates, x_2, y_2 and x_3, y_3 of the midpoint of the pendulum elements are

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 - \frac{1}{2}l_2\sin\theta_2 \\ \frac{1}{2}l_2\cos\theta_2 \end{bmatrix}$$

$$\begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_1 - l_2\sin\theta_2 - \frac{1}{2}l_3\sin\theta_3 \\ y_3 = l_2\cos\theta_2 + \frac{1}{2}l_3\cos\theta_3 \end{bmatrix}.$$

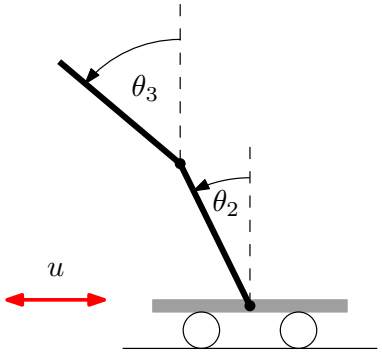


Figure B.3: Cart-double pendulum.

The squared velocities of the cart and the pendulum midpoints are

$$
\begin{aligned}
v_1^2 &= \dot{x}_1^2\,,\\
v_2^2 &= \dot{x}_2^2 + \dot{y}_2^2 = \dot{x}_1^2 - l_2\dot{x}_1\dot{\theta}_2\cos\theta_2 + \tfrac{1}{4}l_2^2\dot{\theta}_2^2\,,\\
v_3^2 &= \dot{x}_3^2 + \dot{y}_3^2 = \dot{x}_1^2 + l_2^2\dot{\theta}_2^2 + \tfrac{1}{4}l_3^2\dot{\theta}_3^2 - 2l_2\dot{x}_1\dot{\theta}_2\cos\theta_2 - l_3\dot{x}_1\dot{\theta}_3\cos\theta_3 + l_2l_3\dot{\theta}_2\dot{\theta}_3\cos(\theta_2-\theta_3)\,.
\end{aligned}
$$

The system Lagrangian is the difference between the kinematic energy T and the potential energy V and given by

$$
\begin{aligned}
L = T - V &= \tfrac{1}{2}m_1v_1^2 + \tfrac{1}{2}m_2v_2^2 + \tfrac{1}{2}m_3v_3^2 + \tfrac{1}{2}I_2\dot{\theta}_2^2 + \tfrac{1}{2}I_3\dot{\theta}_3^2 - m_2gy_2 - m_3gy_3\\
&= \tfrac{1}{2}(m_1+m_2+m_3)\dot{x}_1^2 - \tfrac{1}{2}m_2l_2\dot{x}\dot{\theta}_2\cos(\theta_2) - \tfrac{1}{2}m_3\big(2l_2\dot{x}\dot{\theta}_2\cos(\theta_2) + l_3\dot{x}_1\dot{\theta}_3\cos(\theta_3)\big) + \tfrac{1}{8}m_2l_2^2\dot{\theta}_2^2\\
&\quad + \tfrac{1}{2}I_2\dot{\theta}_2^2 + \tfrac{1}{2}m_3(l_2^2\dot{\theta}_2^2 + \tfrac{1}{4}l_3^2\dot{\theta}_3^2 + l_2l_3\dot{\theta}_2\dot{\theta}_3\cos(\theta_2-\theta_3)) + \tfrac{1}{2}I_3\dot{\theta}_3^2 - \tfrac{1}{2}m_2gl_2\cos(\theta_2)\\
&\quad - m_3g(l_2\cos(\theta_2) + \tfrac{1}{2}l_3\cos(\theta_3))\,.
\end{aligned}
$$

The angular moment of inertia I_j, $j = 2,3$ around the pendulum midpoint is $I_j = \frac{1}{12}ml_j^2$, and $g = 9.82\,\mathrm{m/s^2}$ is the acceleration of gravity. This moment inertia implies the assumption that the pendulums are infinitely thin (but rigid) wires.

The equations of motion are

$$
\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i\,,
$$

where Q_i are the non-conservative forces. We obtain the partial derivatives

$$
\begin{aligned}
\frac{\partial L}{\partial \dot{x}_1} &= (m_1+m_2+m_3)\dot{x}_1 - (\tfrac{1}{2}m_2+m_3)l_2\dot{\theta}_2\cos\theta_2 - \tfrac{1}{2}m_3l_3\dot{\theta}_3\cos\theta_3\,,\\
\frac{\partial L}{\partial x_1} &= 0\,,\\
\frac{\partial L}{\partial \dot{\theta}_2} &= (m_3l_2^2 + \tfrac{1}{4}m_2l_2^2 + I_2)\dot{\theta}_2 - (\tfrac{1}{2}m_2+m_3)l_2\dot{x}_1\cos\theta_2 + \tfrac{1}{2}m_3l_2l_3\dot{\theta}_3\cos(\theta_2-\theta_3)\,,\\
\frac{\partial L}{\partial \theta_2} &= (\tfrac{1}{2}m_2+m_3)l_2(\dot{x}_1\dot{\theta}_2 + g)\sin\theta_2 - \tfrac{1}{2}m_3l_2l_3\dot{\theta}_2\dot{\theta}_3\sin(\theta_2-\theta_3)\,,\\
\frac{\partial L}{\partial \dot{\theta}_3} &= m_3l_3\big[-\tfrac{1}{2}\dot{x}_1\cos\theta_3 + \tfrac{1}{2}l_2\dot{\theta}_2\cos(\theta_2-\theta_3) + \tfrac{1}{4}l_3\dot{\theta}_3\big] + I_3\dot{\theta}_3\,,\\
\frac{\partial L}{\partial \theta_3} &= \tfrac{1}{2}m_3l_3\big[(\dot{x}_1\dot{\theta}_3 + g)\sin\theta_3 + l_2\dot{\theta}_2\dot{\theta}_3\sin(\theta_2-\theta_3)\big]
\end{aligned}
$$

leading to the equations of motion

$$\begin{aligned}(m_1+m_2+m_3)\ddot{x}_1+\tfrac{1}{2}m_2+m_3)l_2(\dot{\theta}_2^2\sin\theta_2-\ddot{\theta}_2\cos\theta_2)+\\ \tfrac{1}{2}m_3l_3(\dot{\theta}_3^2\sin\theta_3-\ddot{\theta}_3\cos\theta_3)=u-b\dot{x}_1\\ (m_3l_2^2+I_2+\tfrac{1}{4}m_2l_2^2)\ddot{\theta}_2-(\tfrac{1}{2}m_2+m_3)l_2(\ddot{x}_1\cos\theta_2+g\sin\theta_2)\\ +\tfrac{1}{2}m_3l_2l_3[\ddot{\theta}_3\cos(\theta_2-\theta_3)+\dot{\theta}_3^2\sin(\theta_2-\theta_3)]=0\\ (\tfrac{1}{4}m_2l_3^2+I_3)\ddot{\theta}_3-\tfrac{1}{2}m_3l_3(\ddot{x}_1\cos\theta_3+g\sin\theta_3)\\ +\tfrac{1}{2}m_3l_2l_3[\ddot{\theta}_2\cos(\theta_2-\theta_3)-\theta_2^2\sin(\theta_2-\theta_3)]=0\end{aligned}$$

These three linear equations in $(\ddot{x}_1,\ddot{\theta}_2,\ddot{\theta}_3)$ can be rewritten as the linear equation system

$$\begin{bmatrix}(m_1+m_2+m_3) & -\frac{1}{2}(m_2+2m_3)l_2\cos\theta_2 & -\frac{1}{2}m_3l_3\cos\theta_3\\ -(\frac{1}{2}m_2+m_3)l_2\cos\theta_2 & m_3l_2^2+I_2+\frac{1}{4}m_2l_2^2 & \frac{1}{2}m_3l_2l_3\cos(\theta_2-\theta_3)\\ -\frac{1}{2}m_3l_3\cos\theta_3 & \frac{1}{2}m_3l_2l_3\cos(\theta_2-\theta_3) & \frac{1}{4}m_2l_3^2+I_3\end{bmatrix}\begin{bmatrix}\ddot{x}_1\\ \ddot{\theta}_2\\ \ddot{\theta}_3\end{bmatrix}=\begin{bmatrix}c_1\\ c_2\\ c_3\end{bmatrix},$$

where

$$\begin{bmatrix}c_1\\ c_2\\ c_3\end{bmatrix}=\begin{bmatrix}u-b\dot{x}_1-\frac{1}{2}(m_2+2m_3)l_2\dot{\theta}_2^2\sin\theta_2-\frac{1}{2}m_3l_3\dot{\theta}_3^2\sin\theta_3\\ (\frac{1}{2}m_2+m_3)l_2g\sin\theta_2-\frac{1}{2}m_3l_2l_3\dot{\theta}_3^2\sin(\theta_2-\theta_3)\\ \frac{1}{2}m_3l_3[g\sin\theta_3+l_2\dot{\theta}_2^2\sin(\theta_2-\theta_3)]\end{bmatrix}.$$

This linear equation system can be solved for $\ddot{x}_1,\ddot{\theta}_2,\ddot{\theta}_3$ and used for numerical simulation.

B.4 Robotic Unicycle

For the equations of motion for the robotic unicycle, we refer to the thesis by Forster (2009).

Appendix C

Parameter Settings

C.1 Cart Pole (Inverted Pendulum)

Table C.1: Simulation parameters: cart pole.

mass of the cart	$M = 0.5\,\mathrm{kg}$
mass of the pendulum	$l = 0.5\,\mathrm{kg}$
pendulum length	$l = 0.6\,\mathrm{m}$
time discretization	$\Delta_t = 0.1\,\mathrm{s}$
variance of cost function	$1/a^2 = \frac{1}{16}\,\mathrm{m}^2$
exploration parameter	$b = -0.2$
initial prediction horizon	$T_{\text{init}} = 2.5\,\mathrm{s}$
maximum prediction horizon	$T_{\max} = 6.3\,\mathrm{s}$
number of policy searches	$PS = 12$
number of basis functions for RBF controller	100
state	$[x\,, \dot{x},\, \dot{\varphi},\, \varphi]^\top$
mean of start state	$\boldsymbol{\mu}_0 = [0,\, 0,\, 0,\, 0]^\top$
target state	$\mathbf{x} = [0,\, *,\, *,\, \pi + 2k\pi]^\top,\, k \in \mathbb{Z}$
force constraint	$u \in [-10, 10]\,\mathrm{N}$
initial state covariance	$\boldsymbol{\Sigma}_0 = 10^{-2}\,\mathbf{I}$

Table C.1 lists the parameters of the cart-pole task.

C.2 Pendubot

Table C.2 lists the parameters of the Pendubot task.

C.3 Cart-Double Pendulum

Table C.3 lists the parameters of the cart-double pendulum task.

Table C.2: Simulation parameters: Pendubot.

pendulum masses	$m_2 = 0.5\,\mathrm{kg} = m_3$
pendulum lengths	$l_2 = 0.6\,\mathrm{m} = l_3$
time discretization	$\Delta_t = 0.075\,\mathrm{s}$
variance of cost function	$1/a^2 = \frac{1}{4}\,\mathrm{m}^2$
exploration parameter	$b = -0.1$
initial prediction horizon	$T_{\mathrm{init}} = 2.55\,\mathrm{s}$
maximum prediction horizon	$T_{\mathrm{max}} = 10.05\,\mathrm{s}$
number of policy searches	$PS = 30$
number of basis functions for RBF controller	150
number of basis functions for sparse $\mathcal{GP}_f$	250
state	$\mathbf{x} = [\dot{\theta}_2, \dot{\theta}_3, \theta_2, \theta_3]^\top$
mean of start state	$\boldsymbol{\mu}_0 = [0,\, 0,\, \pi,\, \pi]^\top$
target state	$\mathbf{x} = [*,\, *,\, 2\,k_2\pi,\, 2\,k_3\pi]^\top,\, k_2,\, k_3 \in \mathbb{Z}$
torque constraint	$u \in [-3.5, 3.5]\,\mathrm{Nm}$
initial state covariance	$\boldsymbol{\Sigma}_0 = 10^{-2}\,\mathbf{I}$

C.4 Robotic Unicycle

Table C.4 lists the parameters used in the simulation of the robotic unicycle. These parameters correspond to the parameters of the hardware realization of the robotic unicycle. Further details are given in the theses by Mellors (2005), Lamb (2005), D'Souza-Mathew (2008), and Forster (2009).

Table C.3: Simulation parameters: cart-double pendulum.

mass of the cart	$m_1 = 0.5\,\mathrm{kg}$
pendulum masses	$m_2 = 0.5\,\mathrm{kg} = m_3$
pendulum lengths	$l_2 = 0.6\,\mathrm{m} = l_3$
time discretization	$\Delta_t = 0.075\,\mathrm{s}$
variance of cost function	$1/a^2 = \frac{1}{4}\,\mathrm{m}^2$
exploration parameter	$b = -0.2$
initial prediction horizon	$T_{\text{init}} = 3\,\mathrm{s}$
maximum prediction horizon	$T_{\max} = 7.425\,\mathrm{s}$
number of policy searches	$PS = 25$
number of basis functions for RBF controller	200
number of basis functions for sparse $\mathcal{GP}_f$	300
state	$\mathbf{x} = [x,\, \dot{x},\, \dot{\theta}_2,\, \dot{\theta}_3,\, \theta_2,\, \theta_3]^\top$
mean of start state	$\boldsymbol{\mu}_0 = [0,\, 0,\, 0,\, 0,\, \pi,\, \pi]^\top$
target state	$\mathbf{x} = [0,\, *,\, *,\, *,\, 2\,k_2\pi,\, 2\,k_3\pi]^\top,\, k_2,\, k_3 \in \mathbb{Z}$
force constraint	$u \in [-20, 20]\,\mathrm{Nm}$
initial state covariance	$\boldsymbol{\Sigma}_0 = 10^{-2}\,\mathbf{I}$

Table C.4: Simulation parameters: robotic unicycle.

mass of the turntable	$m_t = 10\,\mathrm{kg}$
mass of the wheel	$m_w = 1\,\mathrm{kg}$
mass of the frame	$m_f = 23.5\,\mathrm{kg}$
radius of the wheel	$r_w = 0.22\,\mathrm{m}$
length of the frame	$r_f = 0.54\,\mathrm{m}$
time discretization	$\Delta_t = 0.05\,\mathrm{s}$
variance of cost function	$1/a^2 = \frac{1}{100}\,\mathrm{m}^2$
exploration parameter	$b = 0$
initial prediction horizon	$T_{\text{init}} = 1\,\mathrm{s}$
maximum prediction horizon	$T_{\max} = 10\,\mathrm{s}$
number of policy searches	$PS = 11$
state	$\mathbf{x} = [\dot{\theta},\, \dot{\phi},\, \dot{\psi}_w,\, \dot{\psi}_f,\, \dot{\psi}_t,\, \theta,\, \phi,\, \psi_w,\, \psi_f,\, \psi_t]^\top$
mean of start state	$\boldsymbol{\mu}_0 = \mathbf{0}$
target state	$\mathbf{x} = [*,\, *,\, *,\, *,\, *,\, 2k_1\pi,\, *,\, *,\, 2k_2\pi,\, *]^\top,\ k_1,\, k_2 \in \mathbb{Z}$
torque constraints	$u_t \in [-10, 10]\,\mathrm{Nm},\ , u_w \in [-50, 50]\,\mathrm{Nm}$
initial state covariance	$\boldsymbol{\Sigma}_0 = 0.25^2\mathbf{I}$

Bibliography

Abbeel, P. and Ng, A. Y. (2005). Exploration and Apprenticeship Learning in Reinforcement Learning. In *Proceedings of th 22nd International Conference on Machine Learning*, pages 1–8, Bonn, Germay.

Abbeel, P., Quigley, M., and Ng, A. Y. (2006). Using Inaccurate Models in Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 1–8, Pittsburgh, PA, USA.

Alamir, M. and Murilo, A. (2008). Swing-up and Stabilization of a Twin-Pendulum under State and Control Constraints by a Fast NMPC Scheme. *Automatica*, 44(5):1319–1324.

Asmuth, J., Li, L., Littman, M. L., Nouri, A., and Wingate, D. (2009). A Bayesian Sampling Approach to Exploration in Reinforcement Learning. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*.

Åström, K. J. (2006). *Introduction to Stochastic Control Theory*. Dover Publications, Inc., NY, USA.

Atkeson, C. G. and Santamaría, J. C. (1997). A Comparison of Direct and Model-Based Reinforcement Learning. In *Proceedings of the International Conference on Robotics and Automation*.

Atkeson, C. G. and Schaal, S. (1997). Robot Learning from Demonstration. In Fisher Jr., D. H., editor, *Proceedings of the 14th International Conference on Machine Learning*, pages 12–20, Nashville, TN, USA. Morgan Kaufmann.

Attias, H. (2003). Planning by Probabilistic Inference. In Bishop, C. M. and Frey, B. J., editors, *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, Key West, FL, USA.

Bagnell, J. A. and Schneider, J. C. (2001). Autonomous Helicopter Control using Reinforcement Learning Policy Search Methodss. In *In International Conference on Robotics and Automation*, pages 1615–1620. IEEE Press.

Barber, D. (2006). Expectation Correction for Smoothed Inference in Switching Linear Dynamical Systems. *Journal of Machine Learning Research*, 7:2515–2540.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike Elements that Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):835–846.

Baxter, J., Bartlett, P. L., and Weaver, L. (2001). Experiments with Infinite-Horizon, Policy-Gradient Estimation. *Journal of Artificial Intelligence Research*, 15:351–381.

Bays, P. M. and Wolpert, D. M. (2007). Computational Principles of Sensorimotor Control that Minimise Uncertainty and Variability. *Journal of Physiology*, 578(2):387–396.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum Learning. In Bottou, L. and Littman, M., editors, *Proceedings of the 26th International Conference on Machine Learning*, pages 41–48, Montreal, Canada. Omnipress.

Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control*, volume 1 of *Optimization and Computation Series*. Athena Scientific, Belmont, MA, USA, 3rd edition.

Bertsekas, D. P. (2007). *Dynamic Programming and Optimal Control*, volume 2 of *Optimization and Computation Series*. Athena Scientific, Belmont, MA, USA, 3rd edition.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Optimization and Computation. Athena Scientific, Belmont, MA, USA.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. (2009). Natural Actor-Critic Algorithms. Technical Report TR09-10, Department of Computing Science, University of Alberta.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag.

Bloch, A. M., Baillieul, J., Crouch, P., and Marsden, J. E. (2003). *Nonholonomic Mechanis and Control*. Springer-Verlag.

Bogdanov, A. (2004). Optimal Control of a Double Inverted Pendulum on a Cart. Technical Report CSE-04-006, Department of Computer Science and Electrical Engineering, OGI School of Science and Engineering, OHSU.

Brafman, R. I. and Tennenholtz, M. (2002). R-max - A General Polynomial Time Algorithm for Near-optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3:213–231.

Bristow, D. A., Tharayils, M., and Alleyne, A. G. (2006). A Survey of Iterative Learning Control. *IEEE Control Systems Magazine*, 26(3):96–114.

Catanzaro, B., Sundaram, N., and Kreutzer, K. (2008). Fast Support Vector Machine Training and Classification on Graphics Processors. In McCallum, A. and Roweis, S., editors, *Proceedings of the 25th International Conference on Machine Learning*, pages 104–111, Helsinki, Finland. Omnipress.

Chaloner, K. and Verdinelli, I. (1995). Bayesian Experimental Design: A Review. *Statistical Science*, 10:273–304.

Coulom, R. (2002). *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble.

Cressie, N. A. C. (1993). *Statistics for Spatial Data*. Wiley-Interscience.

Csató, L. and Opper, M. (2002). Sparse On-line Gaussian Processes. *Neural Computation*, 14(3):641–668.

Daw, N. D., Niv, Y., and Dayan, P. (2005). Uncertainty-based Competition between Prefrontal and Dorsolateral Striatal Systems for Behavioral Control. *Nature Neuroscience*, 8(12):1704–1711.

Deisenroth, M. P. and Rasmussen, C. E. (2009). Efficient Reinforcement Learning for Motor Control. In *Proceedings of the 10th International PhD Workshop on Systems and Control*, Hluboká nad Vltavou, Czech Republic.

Deisenroth, M. P., Rasmussen, C. E., and Peters, J. (2008). Model-Based Reinforcement Learning with Continuous States and Actions. In *Proceedings of the 16th European Symposium on Artificial Neural Networks (ESANN 2008)*, pages 19–24, Bruges, Belgium.

Deisenroth, M. P., Rasmussen, C. E., and Peters, J. (2009). Gaussian Process Dynamic Programming. *Neurocomputing*, 72(7–9):1508–1524.

Doya, K. (2000). Reinforcement Learning in Continuous Time and Space. *Neural Computation*, 12(1):219–245.

D'Souza-Mathew, N. (2008). Balancing of a Robotic Unicycle. Master's thesis, Department of Engineering, University of Cambridge, UK.

Engel, Y. (2005). *Algorithms and Representations for Reinforcement Learning*. PhD thesis, Hebrew University, Jerusalem, Israel.

Engel, Y., Mannor, S., and Meir, R. (2003). Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*, volume 20, pages 154–161, Washington, DC, USA.

Engel, Y., Mannor, S., and Meir, R. (2005). Reinforcement Learning with Gaussian Processes. In *Proceedings of the 22nd International Conference on Machine Learning (ICML-2005)*, volume 22, pages 201–208, Bonn, Germany.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, 6:503–556.

Forster, D. (2009). Robotic Unicycle. Master's thesis, Department of Engineering, University of Cambridge.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). *Bayesian Data Analysis*. Second. Chapman & Hall/CRC.

Girard, A., Rasmussen, C. E., and Murray-Smith, R. (2002). Gaussian Process Priors with Uncertain Inputs: Multiple-Step Ahead Prediction. Technical Report TR-2002-119, University of Glasgow.

Girard, A., Rasmussen, C. E., Quiñonero Candela, J., and Murray-Smith, R. (2003). Gaussian Process Priors with Uncertain Inputs—Application to Multiple-Step Ahead Time Series Forecasting. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 529–536. The MIT Press, Cambridge, MA, USA.

Gradshteyn, I. S. and Ryzhik, I. M. (2000). *Table of Integrals, Series, and Products*. Academic Press, 6th edition.

Graichen, K., Treuer, M., and Zeitz, M. (2007). Swing-up of the Double Pendulum on a Cart by Feedforward and Feedback Control with Experimental Validation. *Automatica*, 43(1):63–71.

Grancharova, A., Kocijan, J., and Johansen, T. A. (2007). Explicit Stochastic Nonlinear Predictive Control Based on Gaussian Process Models. In *Proceedings of the 9th European Control Conference 2007 (ECC 2007)*, pages 2340–2347, Kos, Greece.

Grancharova, A., Kocijan, J., and Johansen, T. A. (2008). Explicit Stochastic Predictive Control of Combustion Plants based on Gaussian Process Models. *Automatica*, 44(6):1621–1631.

Huang, C. and Fu, L. (2003). Passivity Based Control of the Double Inverted Pendulum Driven by a Linear Induction Motor. In *Proceedings of the 2003 IEEE Conference on Control Applications*, volume 2, pages 797–802.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kakade, S. M. (2002). A Natural Policy Gradient. In Dietterich, T. G., S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 1531–1538. The MIT Press, Cambridge, MA, USA.

Kappeler, F. (2007). Unicycle Robot. Technical report, Automatic Control Laboratory, Ecole Polytechnique Federale de Lausanne.

Kearns, M. and Singh, S. (1998). Near-Optimal Reinforcement Learning in Polynomial Time. In *Machine Learning*, pages 260–268. Morgan Kaufmann.

Khalil, H. K. (2002). *Nonlinear Systems*. Prentice Hall, 3rd (international) edition.

Kimura, H. and Kobayashi, S. (1999). Efficient Non-Linear Control by Combining Q-learning with Local Linear Controllers. In *Proceedings of the 16th International Conference on Machine Learning*, pages 210–219.

Ko, J. and Fox, D. (2008). GP-BayesFilters: Bayesian Filtering using Gaussian Process Prediction and Observation Models. In *Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3471–3476, Nice, France.

Ko, J. and Fox, D. (2009a). GP-BayesFilters: Bayesian Filtering using Gaussian Process Prediction and Observation Models. *Autonomous Robots*, 27(1):75–90.

Ko, J. and Fox, D. (2009b). Learning GP-BayesFilters via Gaussian Process Latent Variable Models. In *Proceedings of Robotics: Science and Systems*, Seattle, USA.

Ko, J., Klein, D. J., Fox, D., and Haehnel, D. (2007). Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 742–747, Rome, Italy.

Kober, J. and Peters, J. (2009). Policy Search for Motor Primitives in Robotics. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 849–856. The MIT Press.

Kocijan, J. and Likar, B. (2008). Gas-Liquid Separator Modelling and Simulation with Gaussian-Process Models. *Simulation Modelling Practice and Theory*, 16(8):910–922.

Kocijan, J., Murray-Smith, R., Rasmussen, C. E., and Girard, A. (2004). Gaussian Process Model Based Predictive Control. In *Proceedings of the 2004 American Control Conference (ACC 2004)*, pages 2214–2219, Boston, MA, USA.

Kocijan, J., Murray-Smith, R., Rasmussen, C. E., and Likar, B. (2003). Predictive Control with Gaussian Process Models. In Zajc, B. and Tkalčič, M., editors, *Proceedings of IEEE Region 8 Eurocon 2003: Computer as a Tool*, pages 352–356, Piscataway, NJ, USA.

Körding, K. P. and Wolpert, D. M. (2004a). Bayesian Integration in Sensorimotor Learning. *Nature*, 427(6971):244–247.

Körding, K. P. and Wolpert, D. M. (2004b). The Loss Function of Sensorimotor Learning. In McClelland, J. L., editor, *Proceedings of the National Academy of Sciences (PNAS)*, volume 101, pages 9839–9842.

Körding, K. P. and Wolpert, D. M. (2006). Bayesian Decision Theory in Sensorimotor Control. *Trends in Cognitive Sciences*, 10(7):319–326.

Kuss, M. (2006). *Gaussian Process Models for Robust Regression, Classification, and Reinforcement Learning*. PhD thesis, Technische Universität Darmstadt, Germany.

Kuss, M. and Rasmussen, C. E. (2006). Assessing Approximations for Gaussian Process Classification. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems 18*, pages 699–706. The MIT Press, Cambridge, MA, USA.

Lamb, A. (2005). Robotic Unicycle: Electronics & Control. Master's thesis, Department of Engineering, University of Cambridge, UK.

Lawrence, N. (2005). Probabilistic Non-linear Principal Component Analysis with Gaussian Process Latent Variable Models. *Journal of Machine Learning Research*, 6:1783–1816.

MacKay, D. J. C. (1992). Information-Based Objective Functions for Active Data Selection. *Neural Computation*, 4:590–604.

MacKay, D. J. C. (1999). Comparison of Approximate Methods for Handling Hyperparameters. *Neural Computation*, 11(5):1035–1068.

MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK.

Matheron, G. (1973). The Intrinsic Random Functions and Their Applications. *Advances in Applied Probability*, 5:439–468.

Mellors, M. (2005). Robotic Unicycle: Mechanics & Control. Master's thesis, Department of Engineering, University of Cambridge, UK.

Miall, R. C. and Wolpert, D. M. (1996). Forward Models for Physiological Motor Control. *Neural Networks*, 9(8):1265–1279.

Michels, J., Saxena, A., and Ng, A. Y. (2005). High Speed Obstacle Avoidance using Monocular Vision and Reinforcement Learning. In *Proceedings of the 22nd International Conference on Machine learning*, pages 593–600, Bonn, Germany. ACM.

Minka, T. P. (2001). *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Murray-Smith, R. and Sbarbaro, D. (2002). Nonlinear Adaptive Control Using Non-Parametric Gaussian Process Prior Models. In *Proceedings of the 15th IFAC World Congress*, volume 15, Barcelona, Spain. Academic Press.

Murray-Smith, R., Sbarbaro, D., Rasmussen, C. E., and Girard, A. (2003). Adaptive, Cautious, Predictive Control with Gaussian Process Priors. In *13th IFAC Symposium on System Identification*, Rotterdam, Netherlands.

Naveh, Y., Bar-Yoseph, P. Z., and Halevi, Y. (1999). Nonlinear Modeling and Control of a Unicycle. *Journal of Dynamics and Control*, 9(4):279–296.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. PhD thesis, Department of Computer Science, University of Toronto.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2004a). Autonomous Inverted Helicopter Flight via Reinforcement Learning. In H. Ang Jr., M. and Khatib, O., editors, *International Symposium on Experimental Robotics*, volume 21 of *Springer Tracts in Advanced Robotics*, pages 363–372. Springer.

Ng, A. Y. and Jordan, M. (2000). PEGASUS: A Policy Search Method for Large MDPs and POMDPs. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 406–415.

Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2004b). Autonomous Helicopter Flight via Reinforcement Learning. In Thrun, S., Saul, L. K., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, USA. The MIT Press.

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2009). Local Gaussian Process Regression for Real Time Online Model Learning. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 1193–1200. The MIT Press, Cambridge, MA, USA.

O'Flaherty, R., Sanfelice, R. G., and Teel, A. R. (2008). Robust Global Swing-Up of the Pendubot Via Hybrid Control. In *Proceedings of the 2008 American Control Conference*, pages 1424–1429.

O'Hagan, A. (1978). Curve Fitting and Optimal Design for Prediction. *Journal of the Royal Statistical Society, Series B*, 40(1):1–42.

Orlov, Y., Aguilar, L., Acho, L., and Ortiz, A. (2008). Robust Orbital Stabilization of Pendubot: Algorithm Synthesis, Experimental Verification, and Application to Swing up and Balancing Control. In *Modern Sliding Mode Control Theory*, volume 375/2008 of *Lecture Notes in Control and Information Sciences*, pages 383–400. Springer.

Peters, J. and Schaal, S. (2006). Policy Gradient Methods for Robotics. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robotics Systems*, pages 2219–2225, Beijing, China.

Peters, J. and Schaal, S. (2008a). Natural Actor-Critic. *Neurocomputing*, 71(7–9):1180–1190.

Peters, J. and Schaal, S. (2008b). Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks*, 21:682–697.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement Learning for Humanoid Robotics. In *Third IEEE-RAS International Conference on Humanoid Robots*, Karlsruhe, Germany.

Petersen, K. B. and Pedersen, M. S. (2009). The Matrix Cookbook. `http://matrixcookbook.com/`.

Poupart, P. and Vlassis, N. (2008). Model-based Bayesian Reinforcement Learning in Partially Observable Domains. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, Fort Lauderdale, FL, USA.

Poupart, P., Vlassis, N., Hoey, J., and Regan, K. (2006). An Analytic Solution to Discrete Bayesian Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 697–704, Pittsburgh, PA, USA. ACM.

Quiñonero-Candela, J., Girard, A., Larsen, J., and Rasmussen, C. E. (2003a). Propagation of Uncertainty in Bayesian Kernel Models—Application to Multiple-Step Ahead Forecasting. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2003)*, volume 2, pages 701–704.

Quiñonero-Candela, J., Girard, A., and Rasmussen, C. E. (2003b). Prediction at an Uncertain Input for Gaussian Processes and Relevance Vector Machines—Application to Multiple-Step Ahead Time-Series Forecasting. Technical Report IMM-2003-18, Technical University of Denmark, 2800 Kongens Lyngby, Denmark.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of Machine Learning Research*, 6(2):1939–1960.

Raiko, T. and Tornio, M. (2005). Learning Nonlinear State-Space Models for Control. In *Proceedings of the International Joint Conference on Neural Networks*, pages 815–820, Montreal, Canada.

Raiko, T. and Tornio, M. (2009). Variational Bayesian Learning of Nonlinear Hidden State-Space Models for Model Predictive Control. *Neurocomputing*, 72(16–18):3702–3712.

Raina, R., Madhavan, A., and Ng, A. Y. (2009). Large-scale Deep Unsupervised Learning using Graphics Processors. In Bouttou, L. and Littman, M. L., editors, *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada. Omnipress.

Rasmussen, C. E. (1996). *Evaluation of Gaussian Processes and other Methods for Non-linear Regression.* PhD thesis, Department of Computer Science, University of Toronto.

Rasmussen, C. E. and Ghahramani, Z. (2001). Occam's Razor. In *Advances in Neural Information Processing Systems 13*, pages 294–300. The MIT Press.

Rasmussen, C. E. and Kuss, M. (2004). Gaussian Processes in Reinforcement Learning. In Thrun, S., Saul, L. K., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*, pages 751–759. The MIT Press, Cambridge, MA, USA.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning.* Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA.

Richter, S. L. and DeCarlo, R. A. (1983). Continuation Methods: Theory and Applications. In *IEEE Transactions on Automatic Control*, volume AC-28, pages 660–665.

Riedmiller, M. (2005). Neural Fitted Q Iteration—First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *Proceedings of the 16th European Conference on Machine Learning (ECML)*, Porto, Portugal.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-Learning Using Connectionist Systems. Technical Report CUED/F-INFENG/TR 166, Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, UK.

Schaal, S. (1997). Learning From Demonstration. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, pages 1040–1046. The MIT Press, Cambridge, MA, USA.

Seeger, M., Williams, C. K. I., and Lawrence, N. D. (2003). Fast Forward Selection to Speed up Sparse Gaussian Process Regression. In Bishop, C. M. and Frey, B. J., editors, *Ninth International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics.

Silverman, B. W. (1985). Some Aspects of the Spline Smoothing Approach to Non-Parametric Regression Curve Fitting. *Journal of the Royal Statistical Society, Series B*, 47(1):1–52.

Smola, A. J. and Bartlett, P. (2001). Sparse Greedy Gaussian Process Regression. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 619—625. The MIT Press, Cambridge, MA, USA.

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian Processes using Pseudo-inputs. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264. The MIT Press, Cambridge, MA, USA.

Snelson, E. L. (2007). *Flexible and Efficient Gaussian Process Models for Machine Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London.

Spong, M. W. and Block, D. J. (1995). The Pendubot: A Mechatronic System for Control Research and Education. In *Proceedings of the Conference on Decision and Control*, pages 555–557.

Stein, M. L. (1999). *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Series in Statistics. Springer Verlag.

Strens, M. J. A. (2000). A Bayesian Framework for Reinforcement Learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 943–950. Morgan Kaufmann Publishers Inc.

Sutton, R. S. (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximate Dynamic Programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 215–224. Morgan Kaufman Publishers.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA.

Titsias, M. K. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*.

Toussaint, M. (2008). Bayesian Inference for Motion Control and Planning. Technical Report TR 2007-22, Technical University Berlin.

Toussaint, M. and Storkey, A. (2006). Probabilistic Inference for Solving Discrete and Continuous State Markov Decision Processes. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 945–952, Pittsburgh, Pennsylvania, PA, USA. ACM.

Valpola, H. and Karhunen, J. (2002). An Unsupervised Ensemble Learning Method for Nonlinear Dynamic State-Space Models. *Neural Computation*, 14(11):2647–2692.

Verdinelli, I. and Kadane, J. B. (1992). Bayesian Designs for Maximizing Information and Outcome. *Journal of the American Statistical Association*, 87(418):510–515.

Wahba, G., Lin, X., Gao, F., Xiang, D., Klein, R., and Klein, B. (1999). The Bias-variance Tradeoff and the Randomized GACV. In *Advances in Neural Information Processing Systems 8*, pages 620–626. The MIT Press, Cambridge, MA, USA.

Walder, C., Kim, K. I., and Schölkopf, B. (2008). Sparse Multiscale Gaussian Process Regression. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1112–1119, Helsinki, Finland. ACM.

Wasserman, L. (2006). *All of Nonparametric Statistics*. Springer Texts in Statistics. Springer Science+Business Media, Inc., New York, NY, USA.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, Cambridge, UK.

Wawrzynski, P. and Pacut, A. (2004). Model-free off-policy Reinforcement Learning in Continuous Environment. In *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, pages 1091–1096.

Williams, C. K. I. (1995). Regression with Gaussian Processes. In Ellacott, S. W., Mason, J. C., and Anderson, I. J., editors, *Mathematics of Neural Networks and Applications*.

Williams, C. K. I. and Rasmussen, C. E. (1996). Gaussian Processes for Regression. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Processing Systems 8*, pages 598–604, Cambridge, MA, USA. The MIT Press.

Williams, R. J. (1992). Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3):229–256.

Zhong, W. and Röck, H. (2001). Energy and Passivity Based Control of the Double Inverted Pendulum on a Cart. In *Proceedings of the 2001 IEEE International Conference on Control Applications*, pages 896–901, Mexico City, Mexico.