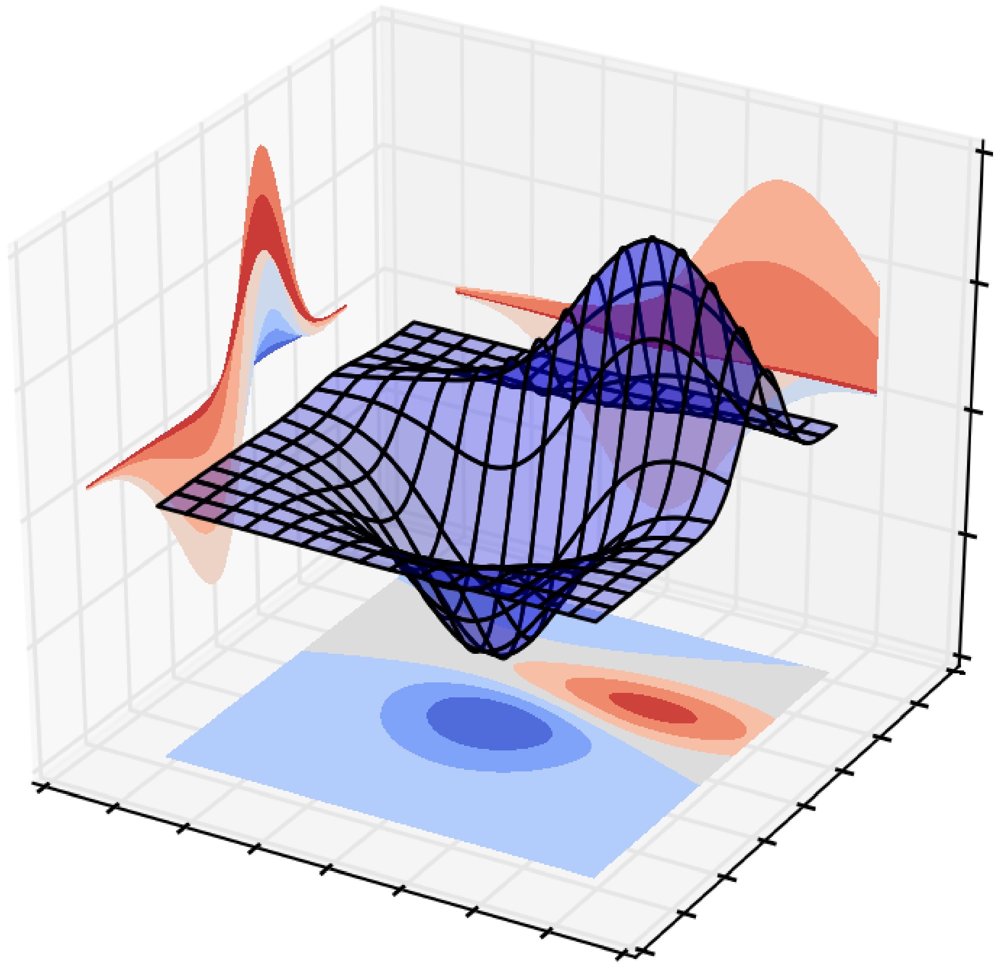
Adaptive Reinforcement Learning

Increasing the applicability for large and time varying systems using parallel Gaussian Process regression and adaptive nonlinear control

K. van Witteveen

Master of Science Thesis



Delft University of Technology

Delft Center for Systems and Control

Adaptive Reinforcement Learning

Increasing the applicability for large and time varying systems using parallel Gaussian Process regression and adaptive nonlinear control

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

K. van Witteveen

January 14, 2014

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by

Abstract

This thesis investigates the applicability of the Probabilistic Inference for Learning COntrol (PILCO) algorithm to large systems and systems with time varying measurement noise. PILCO is a state-of-the-art model-learning Reinforcement Learning (RL) algorithm that uses a Gaussian Process (GP) model to average over uncertainties during learning. Simulated case studies on a second-order system and a cart-pole system show that both the Radial Basis Function (RBF) controller and the GP controller find good solutions when the number of basis functions is chosen correctly. However, when a high number of basis functions is selected, the RBF controller fails completely, while the GP controller is able find a suboptimal solution. In order to reduce the computational time for large systems is the identification of the GP model parallelized. For a four dimensional model the parallelization results in a 20 to 40 percent reduction of the identification computational time. A simulated case study of a cart-pole system shows a strong decrease in performance when increasing the measurement noise variance or kurtosis. The controller is robust for changing skewness of the measurement noise. Furthermore is the variance of the measurement noise an important parameter, because it has to be selected as a fixed parameter of the GP controller prior to learning. Therefore Adaptive-Probabilistic Inference for Learning COntrol (A-PILCO) is proposed. This is a framework that initiates a new learning process when the measurement noise variance exceeds its confidence bounds. By reducing the computational time significantly for large and/or complex systems and by implementing the A-PILCO framework the PILCO algorithm becomes applicable larger set of systems.

Table of Contents

List of Figures

Acknowledgements

This thesis is the result of just over a year of research. I conducted a large part of this research at the University of Newcastle, Australia. My visit to the University of Newcastle, Australia contributed not only to my thesis but also to my personal development. I am grateful for this experience and want to thank Michel Verhaegen and Brett Ninness for giving me this opportunity. Furthermore, I want to thank the people who supported me during this research.

First of all, I want to thank Michel Verhaegen for his supervision and the always positive attitude and enthusiasm. Right from the start, he gave me his confidence and support in order to successfully complete this research. Secondly, I want thank Brett Ninness, for the confidence he showed to have in me, by inviting me to Newcastle. Furthermore, I want to thank Robert Babuška, not only for being a member of my MSc Thesis Committee, but also for being available for a discussion and questions during the project. Last but not least, I want to thank Hans Verstraete. During the the whole research project he was willing to help me. It did not matter if I walked in his office or emailed from Australia I always got an immediate response.

Delft, University of Technology K. van Witteveen
January 14, 2014

Chapter 1

Introduction

Reinforcement Learning (RL) is a control method which learns an (sub)optimal control strategy automatically. This is realized by trying out different control inputs – called actions – and evaluating the performance of these actions. Some actions improve the performance, others deteriorate it. By implementing the actions that lead to an improvement of the performance, the controller learns from its mistakes and successes. This framework is similar to the way organisms learn. Organisms learn by interacting with their environment. Every action leads to a reward. Lewis and Vrabie (2009) have shown that organisms improve their reward by adjusting their actions. For instance, humans are capable of learning complex tasks such as, riding a bike, playing chess, and surfing by trial and error.
Controllers that exploit RL have certain advantages when compared to conventional (hand programmed) controllers. Especially when dynamical systems are made of cheap hardware with inaccurate dynamics to reduce costs, RL controllers become a promising alternative to conventional controllers. Because, individual tuning is not required for different systems that have similar but non-identical dynamics. Furthermore, RL controllers are able to solve complex tasks. Tasks that are very hard or even impossible to solve with hand coded solutions. Robotic walking motion is such a complex tasks. Kohl and Stone (2004) present a RL controller that out performs hand-coded solutions for fast forward motion of a four-legged robot (quadruped trot gait). RL algorithms can be found in many different fields such as playing Backgammon (Tesauro, 1994), driving a bike (Randløv and Alstrøm, 1998), truckload scheduling (Simão et al., 2009), aerobatic maneuvering helicopters (Abbeel et al., 2007), extreme autonomous car driving (Kolter et al., 2010), drug treatment for HIV infected patients (Ernst et al., 2006) or modeling the basal ganglia functioning (Joel et al., 2002).

When system dynamics (model) and reward function are known a priori, the automatic learning process of a controller is often called model-based RL. Where classical techniques of control require restrictive assumptions on the model such as linearity and determinism, the models used in model-based RL methods are generally nonlinear and stochastic. Furthermore, model-based RL methods do not require an analytical model but can use a sample model instead. The advantage of a sample model is that they are often easier to construct than deriving an analytical model, especially when the behavior is stochastic.
The system dynamics and reward function –environment– are often unknown in practice. Model-free RL techniques are developed to directly learn the optimal policy from interacting with the environment. Popular methods are Watkins Q-learning (Watkins, 1989; Watkins and Dayan, 1992), SARSA (Rummery and Niranjan, 1994; Sutton and Barto, 1998)[1] and actor-critic methods. See (Grondman et al., 2012) for an overview of actor-critic RL. Model-free RL typically needs many interactions with the environment. Interacting with a real system can be costly and time consuming. In order to decrease the required interaction time with the real system during learning, which is the measure of data efficiency, algorithms can use models.
Combining model-based RL with unknown environments results in an algorithm that iterates between a system identification and a policy learning step. (1) Starting with no knowledge about the system dynamics, input-output data of an initial trial with the real system is used to find a model. (2) The model is used to find an

[1]Rummery uses the name Modified Q-Learning instead of SARSA

(sub)optimal controller, which is applied to the real system to collect new input-output data.
With this extra information the model can be improved and a possible better controller can be found using this new model. These steps are repeated till the solution converges. This kind of algorithms are called model-learning RL algorithms. Note, that there are three types of learning: model-free RL, model-learning RL and model-based RL. Kuvayev and Sutton (1996) and Atkeson and Santamaria (1997) compared model-free with model-learning methods and found that model-learning RL is more data efficient than model-free RL. Furthermore, the model-learning framework finds better long term plans, policies and handles changing goals more efficiently

1-1 Recent development in model-learning RL

Model-learning methods are scarce, because by iteratively learning a model and a policy, model mismatch arises, called *model-bias*. A totally wrong control policy can be learned when the model does not cover the important system dynamics. This problem is encountered frequently in literature as the key disadvantage of model-learning methods (Schaal, 1997; Atkeson and Schaal, 1997; Atkeson and Santamaria, 1997). Deisenroth and Rasmussen (2011) show that model-bias is especially an issue when no useful prior knowledge and only few data samples are available.
The research of Daw et al. (2005) shows that humans use models to learn a specific task when only a moderate amount of experience is available. Furthermore, Körding and Wolpert (2004, 2006) and Miall and Wolpert (1996) conclude that humans use models for planning by averaging over uncertainties when predicting or making decisions. Furthermore, Schneider (1997) presents that by incorporating uncertainties in models and averaging over these uncertainties model-bias can be taken into account. Inspired from the promising results of Schneider (1997), Deisenroth and Rasmussen (2011) developed the Probabilistic Inference for Learning COntrol (PILCO) algorithm. This model-learning RL algorithm shows that by using a Gaussian Process (GP) model model-bias can be taken into account by averaging over uncertainties and the interaction time is decreased significantly. Due to the nonparametric GP model and learning a single nonlinear controller from scratch, PILCO is a state-of-the-art algorithm has the potential to become a general tool.

1-2 Research objective

Before the PILCO algorithm can become a general tool the applicability of the algorithm has to be enlarged. The choice to use this algorithm is motivated by: (1) the strong decrease of required interaction time during learning, (2) that a GP model can simulate a wide range of nonlinear systems, and (3) that a single nonlinear controller can be learned.

The goal of the thesis is to increase the applicability of the PILCO algorithm for a larger set of systems. The thesis focuses on large systems and systems with time varying measurement noise.

The following sub-objectives have been formulated in order to reach this goal:

1. Identify if the GP controller has learning advantages over the Radial Basis Function (RBF) controller.
2. Reduce the computational time of the algorithm by parallel computing.
3. Identify the influence of different measurement noise characteristics to gain insight in the robustness of the algorithm.
4. Develop an framework that can cope with time varying measurement noise.

1-3 Outline of the thesis

The contents of the chapters of this thesis are:

Chapter 2: Gaussian Process method for regression Before the conducted research is presented some background knowledge is given about regression and the PILCO algorithm. In this chapter regression using a GP is explained, this chapter concludes with a system identification example and motivation why Gaussian Process regression is used in PILCO in Section 2-3.

Chapter 3: The PILCO algorithm In this chapter a detailed explanation on the PILCO algorithm is given. The detailed description of the algorithm ends with an overview of the algorithm in Section 3-4 followed by a discussion on the method.

Chapter 4: Controller choice A performance comparison with respect to the convergence during learning using the GP or RBF controller is presented in this chapter. From this comparison the decision is made which controller to use in the remaining of the thesis.

Chapter 5: Reducing the computational time To make the algorithm usable for a large systems, parallel programming is used to reduce the computational time. This chapter proposes a parallel implementation of the GP training.

Chapter 6: Robustness analyses The robustness of the algorithm is investigated by determining the influence of different characteristics of the measurement distribution on the performance. This chapter gives insight in which moment of the measurement noise distribution is the most important.

Chapter 7: The A-PILCO algorithm This chapter uses the insights of the previous chapter to propose an extension to the algorithm for time varying measurement noise, Adaptive-Probabilistic Inference for Learning COntrol (A-PILCO). This extension holds a variance tracker and decision maker, which initiate a new learning process if the variance exceeds its confidence bounds.

Chapter 8: Conclusions and recommendations The thesis concludes in this chapter with a small summary of the thesis and the conclusions on the research objective. Furthermore, some recommendations for future research are given.

Chapter 2

Gaussian Process regression

The results presented by Deisenroth and Rasmussen (2011) are promising because: (1) the strong decrease of required interaction time during learning, (2) that a Gaussian Process (GP) model can simulate a wide range of nonlinear systems, and (3) that a single nonlinear controller can be learned. Before further research can be presented, the knowledge about the mathematics and methods behind the Probabilistic Inference for Learning COntrol (PILCO) algorithm is required. In the PILCO algorithm Gaussian *regression* is used for the simulation of system dynamics. Regression is often used in financial fields because it concerns the prediction of continue quantities, for instance the price of a commodity as function of the interest rates (Rasmussen and Williams, 2006). How Gaussian random variables can be used to model system dynamics is presented in (Rasmussen and Williams, 2006, Section 2.2) with the *functions-space view*. In this chapter a brief summary of the Gaussian regression for uncertain inputs is given. The chapter concludes with an discussion on GP regression for system identification using a comparison with a parametric gradient based Output Error (OE) system identification method.

2-1 The Gaussian Process model

The goal of GP regression is to find a model that simulates system dynamics as Gaussian distributed random variables. The formal definition of a GP is given in Definition 2-1.1 (Rasmussen and Williams, 2006).

Definition 2-1.1. *A GP is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

A Gaussian random variable is completely described by its mean and covariance. Identically, a collection of Gaussian random variables is completely specified by its mean vector and joint covariance matrix. Consider the function $y = f(x)$, the function values of this function can be described by Gaussian variables. The collection of these variables result in a Gaussian random vector. Hence, the goal of Gaussian regression is to find a GP model that fits the Gaussian random vector. The mean vector and covariance matrix of this GP model can be described by the *mean function* $m(x)$ and *covariance function* $k(x, x')$. Hence, a GP is described by

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')), \tag{2-1a}$$

$$m(x) = \mathbb{E}[f(x)], \tag{2-1b}$$

$$k(x, x') = \mathbb{E}[(f(x) - \mu(x))(f(x') - \mu(x'))]. \tag{2-1c}$$

By fitting the mean and covariance function on the observed data a GP model, Eq. (2-1), is obtained. When data is observed from a dynamic system, the GP model will represent these dynamics. The marginalization property of a GP implies that an examination of a larger set of data does not change the distribution of the smaller set, i.e. if $(y_1, y_2) \sim \mathcal{N}(\mu, \Sigma)$ than $(y_1) \sim \mathcal{N}(\mu_1, \Sigma_{11})$, in which Σ_{11} is the relevant submatrix of Σ (Rasmussen and Williams, 2006). The marginalization property is also known as the consistency requirement. This property makes it possible to add new data to the identification process without interfering with the already found system dynamics. Often the mean function is chosen to be zero. A suitable covariance function is the *Squared Exponential* (SE) covariance function:

$$\operatorname{cov}\left[f(x_p), f(x_q)\right] = k(x_p, x_q) = \exp\left(-\frac{1}{2}|x_p - x_q|^2\right). \tag{2-2}$$

Note, that the covariance of the function *values* $f(x_p), f(x_q)$ depends on the function *inputs* x_p, x_q, see Eq. (2-1c). For inputs close to each other this function will reach unity, while for inputs further apart the function will approach zero. A Bayesian linear regression model with infinite basis functions or a linear combination of infinite number of Gaussian-shaped basis functions corresponds to the Squared Exponential (SE) covariance function (Rasmussen and Williams, 2006).
The informal definition of the *characteristic length-scale*, ℓ, is roughly the distance to move in input space to significantly change the output. The characteristic length-scale and overall variance of the latent function σ_f^2, can be adjusted by positive factors in the following manner:

$$\operatorname{cov}\left[f(x_p), f(x_q)\right] = k(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2}\frac{|x_p - x_q|^2}{\ell^2}\right). \tag{2-3}$$

To evaluate the covariance function test inputs are needed. The distinction between *training inputs* and *test inputs* is made. Training inputs are used to train the model, i.e. fit the most suitable hyper-parameters (σ_f^2, ℓ in this case), see Section 2-2-3. Test inputs are used to test the found model Normally, the test inputs are different from the training inputs. With test inputs $X_* = \begin{bmatrix} x_1^* & , \cdots & x_n^* \end{bmatrix}^T$, the covariance matrix can be found as

$$K(X_*, X_*) = \begin{bmatrix} k(x_1^*, x_1^*) & \cdots & k(x_1^*, x_n^*) \\ \vdots & \ddots & \vdots \\ k(x_n^*, x_1^*) & \cdots & k(x_n^*, x_n^*) \end{bmatrix}. \tag{2-4}$$

2-1-1 Sampling from the prior distribution

By using an example the concept of regression and the mathematics will be explained. The goal of the example is to find a model that accurately models the function of the unknown dynamics, $f(x)$. Choosing the prior mean function equal to zero $m(x) = 0$, $\forall x$ and the prior covariance function as Eq. (2-2) a prior distribution can be found. This prior distribution represents the first guess of the latent function. In order to evaluate the prior distribution a number of test inputs, X_*, must be chosen. By computing the mean vector as $m(x^*) = 0$ $\forall x^*$ and covariance matrix at the test inputs as $K(X_*, X_*)$, (Eq. (2-4)) the *prior distribution* can be found:

$$\mathbf{f}_* \sim \mathcal{N}(0, K(X_*, X_*)). \tag{2-5}$$

Realizations of a Gaussian Process

The realizations of a GP is a Gaussian random vector with all entries sampled from the corresponding Gaussian distribution $\mathbf{y} \sim \mathcal{N}(\mu, \Sigma)$. For an arbitrary mean μ and covariance matrix Σ the Gaussian random vector can be found as

$$\mathbf{y} = \mu + L\mathbf{u}. \tag{2-6}$$

The covariance matrix, which is symmetric and positive definite, holds $\Sigma = LL^T$. Using the Cholesky decomposition the lower triangular matrix L can be found. The entries of the input column vector $\mathbf{u}$ are also

Gaussian distributed, $u_i = \mathcal{N}(0, I)$. Because of the independence of the input $\mathbf{u}$, the realization has the desired distribution with mean μ and covariance matrix Σ. Hence, $\text{var}\,[\mathbf{y}] = L\mathbb{E}[\mathbf{u}\mathbf{u}^T]L^T = LL^T = \Sigma$. By adding a small fraction, ϵI, to the covariance matrix, the Cholesky decomposition can be numerically stabilized. The effect of this stabilization can be considered as adding independent noise with variance ϵ. However, ϵ can be chosen in such way that the effect is insignificant (Rasmussen and Williams, 2006). By plotting the generated values as function of the test inputs, possible functions – realizations – within the prior distribution are found, see Figure 2-1a.

2-1-2 Predictions using noise free observations

Observations from the dynamic system can be used to decrease the uncertainty of the GP model (Rasmussen and Williams, 2006). With X as the vector of training inputs and $\mathbf{f}$ the corresponding noise free observations, $\{(x_i, f_i) | i = 1, ..., n\}$, the prior joint distribution of the training outputs, $\mathbf{f}$, and the test outputs $\mathbf{f}_*$ are

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{2-7}$$

Where the covariance matrices $K(X,X)$, $K(X,X_*)$ and $K(X_*,X)$ are found by following Eq. (2-4). The *posterior distribution* can be imagined as all the possible functions from the prior distribution while rejecting the ones that are in conflict with the observations. This can mathematically be done by conditioning the joint Gaussian prior distribution on the observations. Let $\mathbf{x}$ and $\mathbf{y}$ by the joint Gaussian distribution

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix}\right), \tag{2-8}$$

the *conditional distribution* of $\mathbf{y}$ given $\mathbf{x}$ becomes

$$\mathbf{y}|\mathbf{x} \sim \mathcal{N}\left(\mu_y + C^T A^{-1}(\mathbf{x} - \mu_x), B - C^T A^{-1} C\right). \tag{2-9}$$

Hence, in order to find the function values corresponding to the test inputs, $\mathbf{f}_*$, the joint distribution has to be conditioned on the the observations $\mathbf{f}$. Following Eq. (2-9) the posterior distribution Eq. (2-10), is found. Note, that the prior means are zero, $\mu_y = \mu_x = 0$:

$$\mathbf{f}_*|X_*, X, \mathbf{f} \sim \mathcal{N}\left(\bar{\mathbf{f}}_*, \text{cov}\,[\mathbf{f}_*]\right), \tag{2-10a}$$
$$\bar{\mathbf{f}}_* = K(X_*,X)K(X,X)^{-1}\mathbf{f}, \tag{2-10b}$$
$$\text{cov}\,[\mathbf{f}_*] = K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*). \tag{2-10c}$$

Similar as the prior distribution, samples can be taken from the posterior distribution and plotted as function of the test input, X_*, see Figure 2-1b. These found functions are approximations of the unknown dynamics $f(x)$.

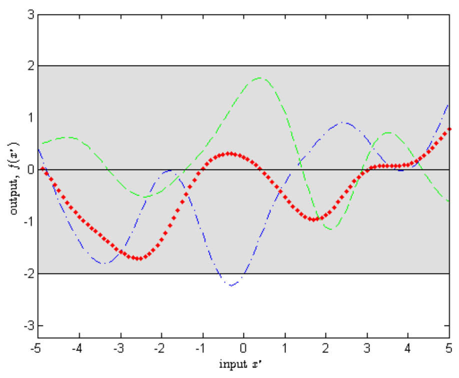
2-1-3 Predictions using noisy observations

When the observations are contaminated with independent and identically distributed (iid) Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$,

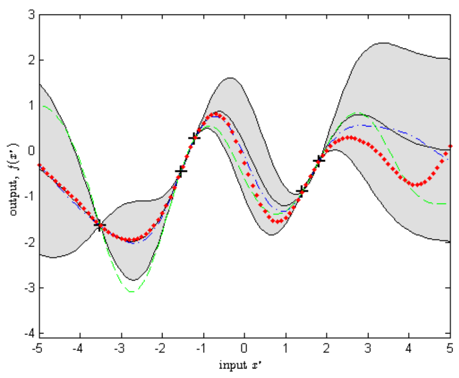
$$y = f(x) + \epsilon, \tag{2-11}$$

the prior covariance function becomes

$$\text{cov}\,[y_p, y_q] = k(x_p, x_q) + \sigma_\epsilon^2 \delta_{pq} \quad \text{or} \quad \text{cov}\,[y] = K(X,X) + \sigma_\epsilon^2 I. \tag{2-12}$$



(a) Sampling from the prior distribution Eq. (2-5).



(b) Sampling from the posterior distribution Eq. (2-10).

Figure 2-1: GP modeling: sampling from prior and posterior distribution using noise free observations. The black solid line is the mean of the distribution and the shaded area is the 95% confidence interval ($\pm 2\sigma$). The red dotted lines are generated points. The other lines are drawn by interpolating between evaluated points. Figure 2-1a shows three functions drawn from the GP prior distribution. Figure 2-1b gives three functions drawn from the posterior distribution. The posterior distribution is obtained by conditioning the prior distribution on the five noise free observations (black crosses). Note, that the uncertainty increases further away from the observation, however the maximum uncertainty is equal to the prior uncertainty.

The δ_{pq} is the Kronecker delta, which is one iff $p = q$ and zero otherwise. The addition of the diagonal noise matrix to the results of the noise free case is followed by the assumption of independent noise. Using the results from Eq. (2-7), the prior joint distribution is

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) + \sigma_\epsilon^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{2-13}$$

Computing the posterior distribution is similar to Eq. (2-10):

$$\mathbf{f}_* | X_*, X, \mathbf{y} \sim \mathcal{N}\left(\bar{\mathbf{f}}_*, \operatorname{cov}[\mathbf{f}_*]\right), \tag{2-14a}$$
$$\bar{\mathbf{f}}_* = K(X_*,X)[K(X,X) + \sigma_\epsilon^2 I]^{-1}\mathbf{y}, \tag{2-14b}$$
$$\operatorname{cov}[\mathbf{f}_*] = K(X_*,X_*) - K(X_*,X)[K(X,X) + \sigma_\epsilon^2 I]^{-1} K(X,X_*). \tag{2-14c}$$

Note, that the predictive mean function Eq. (2-14b) is a linear combination of the observations $\mathbf{y}$. This is often referred to as a *linear predictor*. A function of two arguments that maps a input pair, $x, x' \in \mathcal{X}$ into $\mathbb{R}$ is called a *kernel function*. The kernel is symmetric $k(x', x) = k(x, x')$, note that symmetry is a property of a covariance matrix. For n training inputs $X \in \mathbb{R}^n$ and m test inputs $X_* \in \mathbb{R}^m$, the covariance matrix of the training inputs can be defined as $K = K(X,X)$, the covariances matrix of the test input and the training inputs as $K_* = K(X,X_*)$, and the covariance matrix of the test inputs as $K_{**} = K(X_*,X_*)$. Hence, Eq. (2-14b) and (2-14c) can be written more compact:

$$\bar{\mathbf{f}}_* = K_*(K + \sigma_\epsilon^2 I)^{-1}\mathbf{y}, \tag{2-15}$$
$$\operatorname{cov}[\mathbf{f}_*] = K_{**} - K_*^T (K + \sigma_\epsilon^2 I)^{-1} K_*. \tag{2-16}$$

Rasmussen and Williams (2006) present that the predictive mean function can also be written as a weighted linear combination of n kernel functions, centered on n training inputs:

$$\bar{f}(x_*) = \sum_{i=1}^{n} \beta_i k(x_i, x_*), \tag{2-17}$$

where $\beta = (K + \sigma_\epsilon^2 I)^{-1}\mathbf{y}$.

2-2 Finding the (sub)optimal hyper-parameters

The covariance function is defined by its hyper-parameters, e.g. $\{\ell_1, ..., \ell_D\}, \sigma_f^2$ and σ_ϵ^2 for the SE covariance function with noise. The selection of these hyper-parameters determines if the model is a good representation of the latent function. The selection of the hyper-parameters is often called *training* of a GP model. Consider the SE covariance function with iid Gaussian noise

$$k(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2}(x_p - x_q)^T \Lambda^{-1}(x_p - x_q)\right) + \sigma_\epsilon^2 \delta_{pq}, \tag{2-18}$$

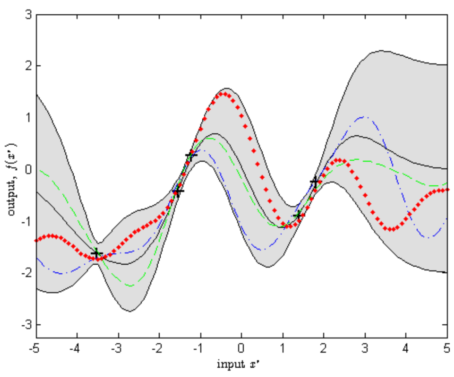
with $x \in \mathbb{R}^D$, $y \in \mathbb{R}^E$ and where $\Lambda = \text{diag}\left(\{\ell_1^2, ..., \ell_D^2\}\right)$. Other choices for the matrix Λ are possible, see (Rasmussen and Williams, 2006, Section 5.1). The vector $\theta = [\{\ell_1, ..., \ell_D\}, \sigma_f^2, \sigma_\epsilon^2]^T$ is the hyper-parameter vector, where ℓ_i denotes the *characteristic length-scales*. In simplified terms, the characteristic length-scales give a measure of the distance one need to move along a particular axis in input space, before the function values become uncorrelated. By inverting these values a relevant weighting matrix is obtained (selecting inputs that are uncorrelated). This property is called *automatic relevance determination*.

2-2-1 Change in the hyper-parameters

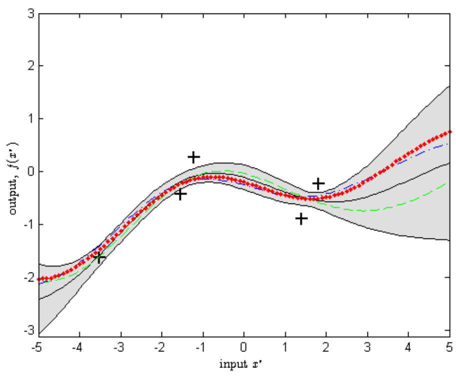
In order to illustrate the influence of change in the hyper-parameters vector θ, the characteristic length-scale is varied for an one-dimensional example. The covariance function as given in (2-18). Note, that by taking the noise into account the covariance function is for the noisy targets y and not for the latent function $f(x)$. Selecting the variance of latent function as $\sigma_f^2 = 1$ and variance of the noise as $\sigma_\epsilon^2 = 0.1$. The hyper-parameter vector is $\theta = [\ell\ 1\ 0.1]^T$, where ℓ is varied.
For $\ell = 1$ we find Figure 2-2a. In this figure the uncertainty increases (to a maximum equal to the prior uncertainty) when the test inputs are 'further away' from the training data. In Figure 2-2b, where the characteristic length-scale is increased to $\ell = 3$, the flexibility of the posterior distribution is decreases. In contrast to Figure 2-2c, where the characteristic length-scale is decreased to $\ell = 0.3$, resulting in a highly flexible function. Because of this flexibility, the mean and covariance functions return rapidly to their prior values, $m(x) = 0$ and $\sigma_f^2 = 1$.
In this section the variance of the latent function and the additive noise are remained constant. By using evidence maximization (Section 2-2-3) the most suitable variances and characteristic length-scales can be found. Typically, the noise variance will decrease for a smaller characteristic length-scale, and visa versa, (Rasmussen and Williams, 2006). This implies that a stiff model – the characteristic length-scale is large – considered the training data as noisy and averages between them by selecting a higher noise variance. For flexible models the noise level will decrease and training data will considered as the true function values. Note, the risk of over fitting exist in the latter case. In this one-dimensional example it is rather easy to select the suitable parameters by hand. However, in multi-dimensional cases it is not that trivial.

2-2-2 Bayesian inference

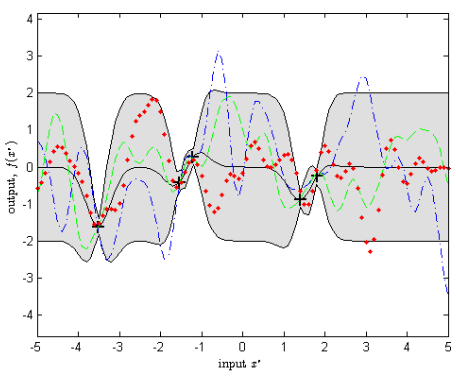
To find the most suitable hyper-parameters (θ) values, *Bayesian inference* techniques are used. With this method the posterior distribution of the hyper-parameters can be found. Two levels of inference are needed. In the first level the posterior distribution of the latent function is found. With the second level the posterior distribution of the hyper-parameters is obtained (Deisenroth, 2010).



(a) $\ell = 1$.



(b) $\ell = 3$



(c) $\ell = 0.3$

Figure 2-2: GP modeling: sampling from posterior distributions using the same training data and hyper-parameter vector $\theta = [\ell\ 1\ 0.1]^T$, but different characteristic length-scales ℓ. The black solid line is the mean of the posterior distribution and the gray shade is the 95% confidence interval ($\pm 2\sigma$). The black crosses are the training data. The red dotted lines are generated points. The other lines are drawn by interpolating between evaluated points. Figure 2-2a shows the posterior distribution using a GP model with $\ell = 1$. Figure 2-2b shows the posterior distribution using $\ell = 3$, and Figure 2-2c shows the posterior distribution using $\ell = 0.3$.

Level-1 inference

The posterior distribution of the function is found as

$$p(f|X, \mathbf{y}, \theta) = \frac{p(\mathbf{y}|X, f, \theta)p(f|\theta)}{p(\mathbf{y}|X, \theta)}, \tag{2-19}$$

where $p(\mathbf{y}|X, \theta)$ is the prior distribution of the function $f(x)$ and $p(\mathbf{y}|X, f, \theta)$ is the *likelihood* of the function $f(x)$. Assuming that the observations y_i are conditionally independent given the inputs X, the likelihood can be written as

$$p(\mathbf{y}|X, f, \theta) = \prod_{i=1}^{n} p(y_i|f(x_i), \theta) = \prod_{i=1}^{n} \mathcal{N}(y_i|f(x_i), \sigma_\epsilon^2) = \mathcal{N}(\mathbf{y}|f(X), \sigma_\epsilon^2 I). \tag{2-20}$$

The normalizing constant of Eq. (2-19) also called *marginal likelihood* or *evidence* is

$$p(\mathbf{y}|X,\theta) = \int p(\mathbf{y}|X,f,\theta)p(f|\theta)df. \tag{2-21}$$

This evidence is the likelihood of the hyper-parameters given the data.

Level-2 inference

The posterior on the hyper-parameters is

$$p(\theta|X,\mathbf{y}) = \frac{p(\mathbf{y}|X,\theta)p(\theta)}{p(\mathbf{y}|X)}, \tag{2-22}$$

where the prior distribution of the hyper-parameters $p(\theta)$ is independent of the training inputs X, hence $p(\theta|X) = p(\theta)$. The distribution of the training targets given the training inputs is

$$p(\mathbf{y}|X) = \iint p(\mathbf{y}|X,f,\theta)p(f|\theta)p(\theta)dfd\theta = \int p(\mathbf{y}|X,\theta)p(\theta)d\theta. \tag{2-23}$$

Computing the distribution $p(y|X)$ is analytical intractable in most interesting cases, because $p(\mathbf{y}|X,\theta)$ is a nasty function of θ

$$\log p(y|X,\theta) = -\frac{1}{2}y^T K_\theta^{-1}\mathbf{y} - \frac{1}{2}\log|K_\theta| - \frac{D}{2}\log 2\pi. \tag{2-24}$$

D is the dimension of the input space and K_θ is the $(n \times n)$ covariance matrix of the training inputs *and* noise, $K_{\theta,ij} = k(X_i,X_j) + \delta_{ij}\sigma_\epsilon^2$, where δ is the Kronecker delta. Markov chain Monte Carlo methods exists to find these posterior distributions without deriving the integrals analytically (Ninness and Henriksen, 2010). These methods are computationally demanding and therefore not used in the PILCO algorithm (Deisenroth, 2010). Instead, the marginal likelihood of Eq. (2-21) is used to find a point estimate of the hyper-parameters, $\hat{\theta}$, using evidence maximization.

2-2-3 Evidence Maximization

To find a point estimate of the hyper-parameter vector $\hat{\theta}$ a 'flat' hyper-prior $p(\theta)$ is selected. In this way none of the possible parameters are excluded. Furthermore, an additional computational advantage is obtained. The marginal likelihood Eq. (2-21) becomes proportional to the posterior distribution of the hyper-parameter Eq. (2-22), so $p(\theta|X,\mathbf{y}) \propto p(\mathbf{y}|X,\theta)$. This implies that the hyper-parameters which maximize the marginal likelihood, will also maximize the posterior distribution of the hyper-parameters. Hence, finding the posterior distribution of the hyper-parameters – which is computational demanding – is done with a less demanding maximization of the marginal likelihood Eq. (2-21). Rasmussen and Williams (2006) show that be taking the log-marginal likelihood the hyper-parameters can be found:

$$\log p(\mathbf{y}|X,\theta) = -\frac{1}{2}\mathbf{y}^T K_\theta^{-1}\mathbf{y} - \frac{1}{2}\log|K_\theta| - \frac{D}{2}\log 2\pi, \tag{2-25}$$

$$\hat{\theta} \in \arg\max_\theta \log p(\mathbf{y}|X,\theta). \tag{2-26}$$

Finding the hyper-parameters by maximizing the log marginal likelihood is called *evidence maximization*. To be more precise the *type II maximum likelihood (ML-II)* estimates of the hyper-parameters is used.[1] To find the maximum the following partial derivative is needed:

[1] To find the point estimate of the hyper-parameters using ML-II estimate gpml-software is used, which is publicly available at http://www.gaussianprocess.org.

$$\begin{aligned}\frac{\partial}{\partial\theta_j}\log(\mathbf{y}|X,\theta) &= \frac{1}{2}\mathbf{y}^T K_\theta^{-1}\frac{\partial K_\theta}{\partial\theta_j}K_\theta^{-1}\mathbf{y} - \frac{1}{2}\operatorname{tr}\left(K_\theta^{-1}\frac{\partial K_\theta}{\partial\theta_j}\right)\\ &= \frac{1}{2}\operatorname{tr}\left(\left(K_\theta^{-1}\mathbf{y}\mathbf{y}^T K_\theta^{-1} - K_\theta^{-1}\right)\frac{\partial K_\theta}{\partial\theta_j}\right).\end{aligned} \tag{2-27}$$

Where $\operatorname{tr}(\cdot)$ is the trace of the argument. The computational load of the partial derivative is dominated by the inversion of matrix K_θ, $\mathcal{O}(n^3)$ for an square positive definite symmetric $n\times n$ matrix. Once the inverse is known, the time to calculate the derivatives is $\mathcal{O}(n^2)$ per hyper-parameter (Rasmussen and Williams, 2006). This is obtained by first computing the vectors times matrices. For the trace only the diagonal entries are computed. This holds for all targets $\mathbf{y}\in\mathbb{R}^E$. Hence, the computational load for K_θ is $\mathcal{O}(En^3)$. The computational load for the remaining computations of the derivative are $\mathcal{O}(En^2)$ per hyper-parameter.

2-3 Simulations using a Gaussian Process model

To illustrate the principle of the presented theory an example will be given. The 'real' dynamics are given by the first order system:

$$\dot{x}_t = -x_t + u_t, \tag{2-28}$$

$$y_t = x_t + \epsilon_t, \tag{2-29}$$

where ϵ_t is iid Gaussian measurement noise $\epsilon_t \sim \mathcal{N}(0, 0.1)$. With this system training data $(\tilde{X}, \mathbf{y})$ is generated with a random input $u = \mathcal{N}(0, 4)$. For $t = 0 : n$ the training inputs[2] $\tilde{X}$ is the vector of the input tuples $\tilde{x}_t = (x_t, u_t)$. The training targets are the corresponding outputs y_t collected in the vector $\mathbf{y}$:

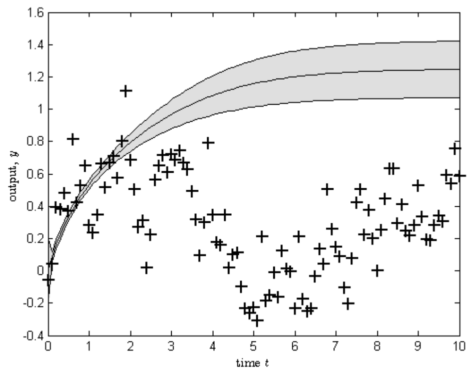
$$\tilde{X} = \begin{bmatrix} x_0 & u_0 \\ \vdots & \vdots \\ x_{n-1} & u_{n-1}\end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n\end{bmatrix}. \tag{2-30}$$

In order to find a GP model that maps the current state into the next, $x_t \to x_{t+1}$, the zero mean function and the SE covariance function are selected. The (sub)optimal hyper-parameters can be found using Evidence Maximization Section 2-2-3, which determines the GP model. In the following two sections this Gaussian Process model will be used to simulate the 'real' dynamics. A step response will be used to compare the GP with the 'real' dynamics. In the firs part the simulation will assume deterministic input states, i.e. the uncertainty of the GP will not be passed on to the next state. Note, that even when using a deterministic input an uncertain output will be obtained. Subsequently, the uncertainty will be taken into account in the second section.

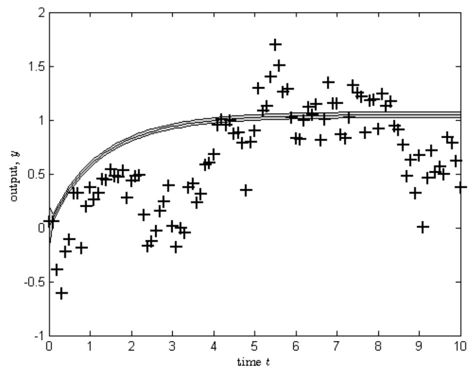
2-3-1 Simulating using deterministic inputs

The system dynamics can be simulated by cascading one-step predictions of the GP model, starting from initial state $x_0 \sim \mathcal{N}(0, 0.1^2)$. The control input in this experiment is set to one ($u = 1$, $\forall x$), in order to compare the results with the 'real' step response. Hence, the *test* input pair becomes: $\tilde{x}_t = [x_t \ 1]$. By ignoring the uncertainty, the model mismatch is ignored. Model mismatch – model bias – is typically caused by poor training data and/or noise. The one-step predictions of the GP model, using *deterministic test inputs*, can be found using

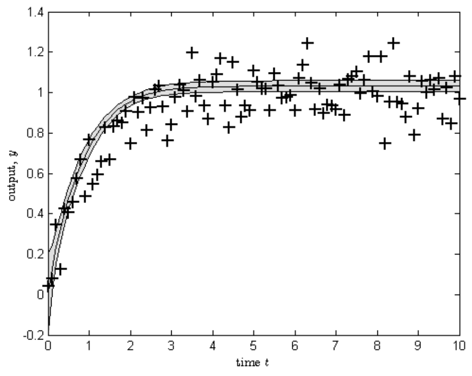
[2] All variables written with a tilde $\tilde{\cdot}$, refer to state-action tuple variables.



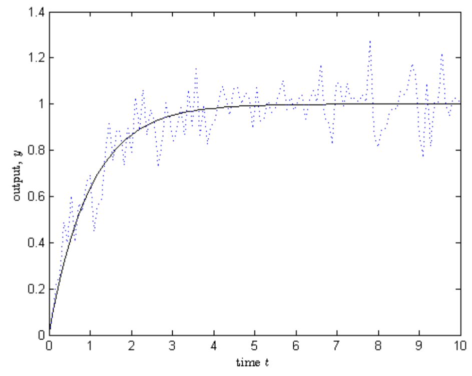
(a) GP predictions when the simulation space is far from training data.



(b) GP predictions when the simulation space is close to training data.



(c) GP predictions when using the same input for training and simulation.



(d) Step response of the 'real' system dynamics.

Figure 2-3: GP modeling: step response simulations of the first-order system $G = \frac{1}{(1+s)}$, using models trained with different training data. The training data given by the black crosses is disturbed with iid measurement noise $\epsilon_t \sim \mathcal{N}(0, 0.1)$. The black solid line is the mean prediction of the state and the gray area is the 95% confidence interval $(\pm 2\sigma)$. In Figure 2-3a the simulation space is 'far away' from the training data. In Figure 2-3b the improvement can be seen when the simulation space is closer to the training targets. In Figure 2-3c the same control input $(u_t = 1 \forall t)$ is used for the training data and prediction. Note, the near perfect step response. Figure 2-3d shows the step response of the 'real' system dynamics. The black solid line is the ouput of the 'real' sytem without mearurement noise and the blue dotted line is with added measurement noise.

$$p(x_{t+1}|\tilde{x}_t) \sim \mathcal{N}(\mu_{t+1}, \Sigma_{t+1}), \tag{2-31a}$$
$$\mu_{t+1} = \mathbb{E}[x_{t+1}|\tilde{x}_t], \tag{2-31b}$$
$$\Sigma_{t+1} = \text{var}\left[x_{t+1}|\tilde{x}_t\right]. \tag{2-31c}$$

Note, that the estimation of the next state is the mean of the approximated Gaussian output of the GP model, $\mathbb{E}[x_{t+1}]$, see Eq. (2-15) and Eq. (2-17). Hence, the state one-step update rule, dependent of the current state-action pair $\tilde{x}_t$ becomes

$$\mu_{t+1} = \mathbb{E}[x_{t+1}|\tilde{x}_t] = K(\tilde{X}, \tilde{x}_t)(K + \sigma_\epsilon^2 I)^{-1}\mathbf{y} = K(\tilde{X}, \tilde{x}_t)\beta = \sum_{i=1}^{n} \beta_i k(\tilde{x}_i, \tilde{x}_*), \tag{2-32}$$

where $K(\tilde{X}, \tilde{x}_t)$ is a vector containing the covariance between the training inputs and the *single* test input. The test input is the tuple of the current state and control input $\tilde{x}_t = [x_t \; u_t]$. Hence,

$$K(\tilde{X}, \tilde{x}_t) = \begin{bmatrix} k([x_0, u_0], \tilde{x}_t) & \dots & k([x_n, u_n], \tilde{x}_t) \end{bmatrix}, \tag{2-33}$$

and K is the covariance matrix of the training inputs,

$$K = \begin{bmatrix} k([x_0, u_0], [x_0, u_0]) & \cdots & k([x_0, u_0], [x_n, u_n]) \\ \vdots & \ddots & \vdots \\ k([x_n, u_n], [x_0, u_0]) & \cdots & k([x_n, u_n], [x_n, u_n]) \end{bmatrix}. \tag{2-34}$$

The uncertainty of the posterior distribution is represented as the 95% conficence interval ($\mu_{t+1} \pm 2\sqrt{\sigma_{t+1}}$) in Figure 2-3. In order to find the uncertainty the variance σ_{t+1} of the next state (in this example a scaler, but generaly represented with a covariance matrix) is required. Using Eq. (2-16) the variance one-step update rule, dependent of the current state-action pair $\tilde{x}_t$ is

$$\Sigma_{t+1} = \text{var}\left[x_{t+1}|\tilde{x}_t\right] = K(\tilde{x}_t, \tilde{x}_t) - K(\tilde{X}, \tilde{x}_t)^T (K + \sigma_\epsilon^2 I)^{-1} K(\tilde{X}, \tilde{x}_t). \tag{2-35}$$

Because only one test input is considered, the covariance 'matrix' $K(\tilde{x}_t, \tilde{x}_t)$ is the scaler $k(\tilde{x}_t, \tilde{x}_t)$. By forming the next test input as $\tilde{x}_{t+1} = [x_{t+1} \; u_{t+1}]$ the one-step predictions can be cascaded and used as a simulation of the 'real' dynamics.
In Figure 2-3 the results of the step response simulations of the first-order system $G(s) = \frac{1}{1+s}$ are given for models trained with different training data. The training data is contaminated with iid normal distributed measurement noise, $\epsilon_t \sim \mathcal{N}(0, 0.1)$. The true and noisy output of the 'real' system is shown in Figure 2-3d. The accuracy of the GP model depends on the training data. When the training data is 'further away' from the simulation domain, the important dynamics are not captured by the training data. Hence, the accuracy of the estimates decreases, see Figure 2-3a. When the workspace of the simulation is close to the training targets, the model approximates the output more accurate, see Figure 2-3b. Using the same control input for obtaining the training data and simulation a near perfect response is found, see Figure 2-3c. This emphasizes the importance of the chosen control input of the training data generating trials. Preferably, an input is chosen in order to reach the entire relevant workspace.

2-3-2 Simulating using uncertain inputs

Even when the input is deterministic the output of the GP is an approximation of the next state given by a Gaussian distribution, $x_{t+1} \sim \mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$. By inserting this state uncertainty into the next prediction the uncertainty of the model is taken into account. In order to implement the uncertainty of the current state in the computations for the next, the mean vector and covariance matrix of the input pair have to be determined. The control input signal u_t is generally dependent on the states, $u_t = \pi(x_t)$. If so, the cross-covariance of the state and the input has to be computed, see Section 3-3-1. However in this example, the 'controller' is independent of the state and time index t, $(u = 1, \forall x, t)$, resulting in the following mean vector and covariance matrix:

$$\tilde{\mu}_t = \begin{bmatrix} \mu_t \\ \bar{u}_t \end{bmatrix}, \qquad \tilde{\Sigma}_t = \begin{bmatrix} \Sigma_t & 0 \\ 0 & 0 \end{bmatrix}. \tag{2-36}$$

The exact predictive distribution for an Gaussian distributed input, $\tilde{x}_t = \mathcal{N}(\tilde{\mu}_t, \tilde{\Sigma}_t)$, mapped through a GP model is

$$p(x_{t+1}|\tilde{\mu}_t, \tilde{\Sigma}_t) = \int p(x_{t+1}|\tilde{x}_t) p(\tilde{x}_t) d\tilde{x}_t. \tag{2-37}$$

In the exact predictive distribution, Eq. (2-37), the dependence on the training data, $\tilde{X}, \mathbf{y}$ and the posterior hyper-parameters $\hat{\theta}$ is omitted for easy reading. Generally this predictive distribution is not Gaussian and unimodal, because mapping a Gaussian distribution through a nonlinear function of a GP leads to an non-Gaussian predictive distribution. Therefore the predictive distribution can not be computed analytical, and is approximated as a Gaussian distribution using Moment Matching (Deisenroth, 2010). This implies that the mean and covariance of the exact prediction distribution is computed and used to approximate the prediction distribution as a Gaussian, see Figure 2-6 for a visualization. In order to compute the mean the next state, the uncertainty is incorporated into Eq. (2-32) resulting in the following update rule:

$$\begin{aligned} \mu_{t+1} = \mathbb{E}[x_{t+1}|\tilde{\mu}_t, \tilde{\Sigma}_t] &= \sigma_f^2 |\tilde{\Sigma}_t \Lambda^{-1} + I|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\tilde{X} - \tilde{\mu}_t)^T (\tilde{\Sigma}_t + \Lambda)^{-1} (\tilde{X} - \tilde{\mu}_t)\right) (\tilde{X} + \sigma_\epsilon^2 I)^{-1} \mathbf{y} \\ &= \beta^T \mathbf{q}. \end{aligned} \tag{2-38}$$

Note, that Eq. (2-38) is related to Eq. (2-32) and that $\mathbf{q}$ is $K(X, \tilde{x}_t)$ infiltrated with the covariance of $\tilde{x}_t$. When setting $\tilde{\Sigma}_t$ to zero the expression of Eq. (2-32) is found again. Note, that for *deterministic* test inputs the mean update Eq. (2-32) and covariance update Eq. (2-35) are written dependent of the *state-action* pair, $\cdot|\tilde{x}_t$. When an uncertain Gaussian distributed test input is used the expressions are written dependent of the *mean* and *covariance* of the state-action pair, $\cdot|\tilde{\mu}_t, \tilde{\Sigma}_t$. The variance of the prediction distribution for an uncertain input is found as

$$\begin{aligned} \Sigma_{t+1} = \text{var}\left[x_{t+1}|\tilde{\mu}_t, \tilde{\Sigma}_t\right] &= \mathbb{E}[\underbrace{\text{var}\left[x_{t+1}|\tilde{x}_t\right]}_{Eq.\ (2-35)} |\tilde{\mu}_t, \tilde{\Sigma}_t] + \text{var}[\underbrace{\mathbb{E}[x_{t+1}|\tilde{x}_t]}_{Eq.\ (2-32)} |\tilde{\mu}_t, \tilde{\Sigma}_t] \\ &= \mathbb{E}[\text{var}\left[x_{t+1}|\tilde{x}_t\right] |\tilde{\mu}_t, \tilde{\Sigma}_t] + \left(\mathbb{E}[\mathbb{E}[x_{t+1}|\tilde{x}_t]^2 |\tilde{\mu}_t, \tilde{\Sigma}_t] - \mathbb{E}[\mathbb{E}[x_{t+1}|\tilde{x}_t]|\tilde{\mu}_t, \tilde{\Sigma}_t]^2\right), \end{aligned} \tag{2-39}$$

where $\mathbb{E}[x_{t+1}|\tilde{x}_t]$ and $\text{var}\left[x_{t+1}|\tilde{x}_t\right]$ are the deterministic mean and variance updates, Eq. (2-32) and Eq. (2-35) respectively. Hence, Eq. (2-39) can be written as

$$\begin{aligned} \Sigma_{t+1} = &\int K(\tilde{x}_t, \tilde{x}_t) - K(\tilde{x}_t, \tilde{X})(K + \sigma_\epsilon^2 I)^{-1} K(\tilde{X}, \tilde{x}_t) p(\tilde{x}_t) d\tilde{x}_t \\ &+ \int K(\tilde{x}_t, \tilde{X}) \beta \beta^T K(\tilde{X}, \tilde{x}_t) p(\tilde{x}_t) d\tilde{x}_t - (\beta^T \mathbf{q})^2 \\ &= \sigma_f^2 - \text{tr}\left((K + \sigma_\epsilon^2 I)^{-1} \int K(\tilde{X}, \tilde{x}_t) K(\tilde{x}_t, \tilde{X}) p(\tilde{x}_t) d\tilde{x}_t\right) \\ &+ \beta^T \underbrace{\int K(\tilde{X}, \tilde{x}_t) K(\tilde{x}_t, \tilde{X}) p(\tilde{x}_t) d\tilde{x}_t}_{:=\hat{Q}} \beta - (\beta^T q)^2. \end{aligned} \tag{2-40}$$

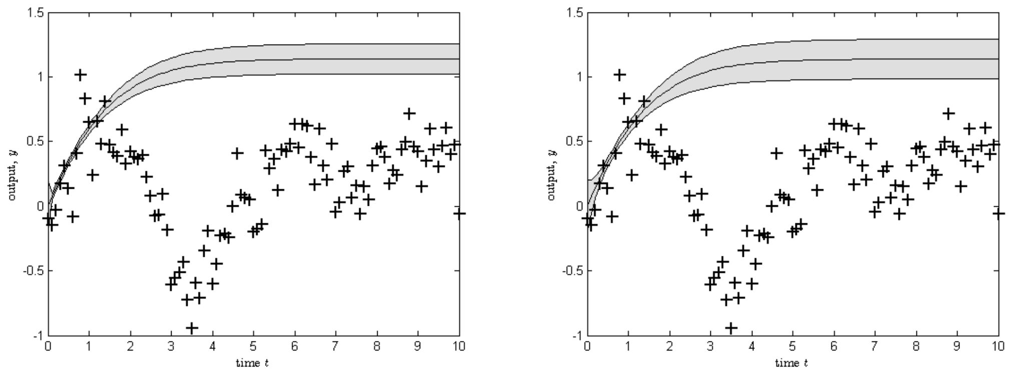
Which can be rewritten compactly as

$$\Sigma_{t+1} = \underbrace{\sigma_f^2 - \mathrm{tr}\left((K+\sigma_\epsilon^2 I)^{-1}\hat{Q}\right)}_{\mathbb{E}\left[\mathrm{var}\left[x_{t+1}|\tilde{x}_t\right]|\mu_t,\Sigma_t\right]} + \underbrace{\beta^T\hat{Q}\beta - \mu_{t+1}^2}_{\mathrm{var}\left[\mathbb{E}[x_{t+1}|\tilde{x}_t]|\mu_t,\Sigma_t\right]}, \tag{2-41}$$

where the entries of $\hat{Q}$ are[3]

$$\hat{Q}_{ij} = \frac{K(\tilde{X}_i,\tilde{\mu}_t)K(\tilde{X}_j,\tilde{\mu}_t)}{\sqrt{|2\tilde{\Sigma}_t\Lambda^{-1}+I|}}\exp\left((z_{ij}-\tilde{\mu}_t)^T(\tilde{\Sigma}_t+\frac{1}{2}\Lambda)^{-1}\tilde{\Sigma}_t\Lambda^{-1}(z_{ij}-\tilde{\mu}_t)\right), \tag{2-42}$$

with $z_{ij} = \frac{1}{2}(\tilde{X}_i+\tilde{X}_j)$. Using a GP model all possible models that fit the training data, prior mean function and covariance function are taken into consideration. By cascading the uncertainty of the model from the current state to the next, the model is more uncertain than by using the predicted mean as a deterministic state, see Figure 2-4. Incorporating the uncertainties in the long term planning is key to the PILCO algorithm, because it takes model bias into account. Model bias is the main disadvantage for using model-learning algorithms.



(a) GP model predictions for deterministic input pairs.

(b) GP model predictions for uncertain input pairs.

Figure 2-4: GP modeling: comparison of the simulations of the step response of the first-order system $G = \frac{1}{(1+s)}$. The training data (black crosses) are disturbed with iid Gaussian measurement noise $\epsilon_t \sim \mathcal{N}(0, 0.1)$. The black solid line is the mean prediction of the state and the gray area is the 95% confidence interval ($\pm 2\sigma$). Figure 2-4a presents the simulated response for deterministic input pairs and Figure 2-4b for uncertain input pairs $p(\tilde{x}_t) = \mathcal{N}(\tilde{\mu}_t, \tilde{\Sigma}_t)$.

2-4 Discussion and conclusion

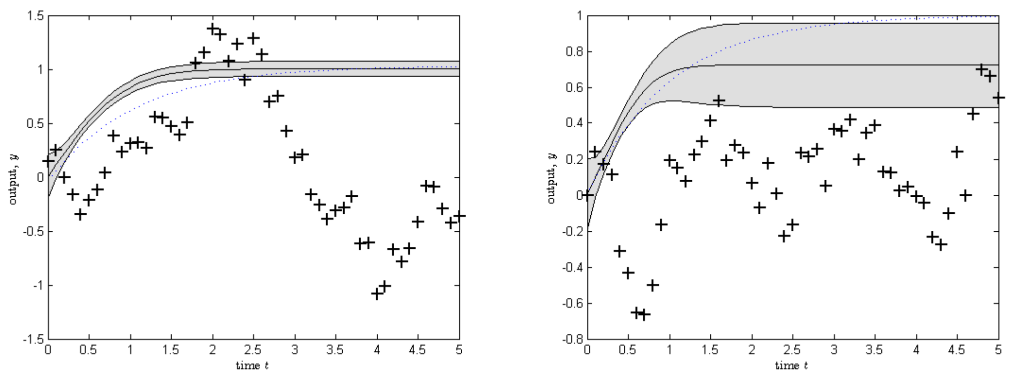
To evaluate and discuss the performance of the GP model, the model will be compared with a parametric OE model. The GP model is found as described in Section 2-3. The parametric OE model is found using the gradient based identification algorithm of the MATLAB toolbox `UNIT` (Ninness et al., 2013). The parametric model is found by selecting the following OE structure and polynomials:

$$y(t) = \frac{B(q)}{A(q)}x_t + \epsilon_t, \qquad A(q) = 1 + a_1 q^{-1}, \qquad B(q) = b_0 + b_1 q^{-1}. \tag{2-43}$$

The gradient based search method is used to find the parameters, (a_1, b_0, b_1). A performance comparison of the two models is shown in Figure 2-5. Both models are found using the same input-output data obtained

[3]$\hat{Q}$ is written with a 'hat' because for the predictions of state vectors the expression of $\hat{Q}$ changes, see Eq. (3-14). In order to indicate this $\hat{Q}$ and Q are used.

using a single trial with a random control input. As presented in Figure 2-5a is the performance of the two methods similar. However, in Figure 2-5b the prediction of the GP model is inaccurate. This shows again the importance of the training data for the GP model to come up with accurate approximations. Although the poor approximations in Figure 2-5b, the GP system identification methods is not weak. However, one has to be carefully when selecting the control input for training. Furthermore, the GP model is capable of modeling high non-linear systems including their uncertainties.
The model found by the parametric system identification claims the output with full confidence, however model mismatch is common. This unrecognized model-bias can lead to big errors in the policy during learning of the controller (Atkeson and Santamaria, 1997). This problem is known as the model-bias problem. The model-bias problem can be reduced by averaging over the model uncertainties (Schneider, 1997). Because: (1) GP modeling is based on well understood approximation principles, and (2) is capable of modeling a wide spread of nonlinear systems including their uncertainties, this method is used in the PILCO algorithm.



(a) Comparison of the GP model predictions and parametric OE model predictions, when training data is rich enough.

(b) Comparison of the GP model predictions and parametric OE model predictions, when training data is poor.

Figure 2-5: GP modeling: comparison of the GP model (the black solid line is the mean prediction with the shaded 95% confidence interval) and the parametric OE model predictions (blue dotted line), the black crosses are training data, disturbed with iid Gaussian measurement noise $\epsilon_t \sim \mathcal{N}(0, 0.1)$. In Figure 2-5a the training data is rich enough to find a accurate GP model. While in Figure 2-5b the training data is insufficient to find an accurate GP model. In both situations the gradient based parametric OE model is capable of finding an accurate prediction.

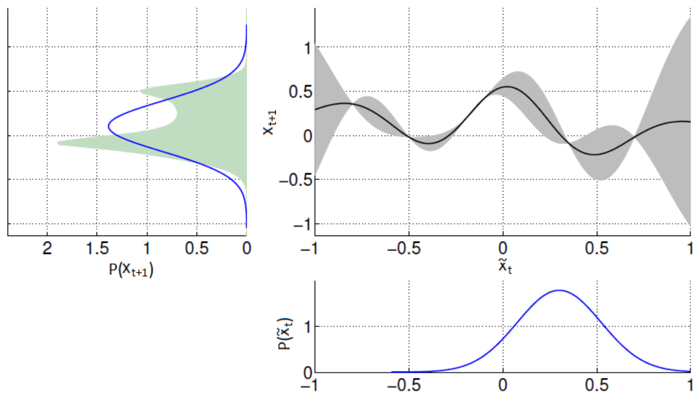


Figure 2-6: The visualization of the approximation of the next state given an uncertain input using Moment Matching, edited from (Deisenroth and Rasmussen, 2009). The input (blue distribution bottom right figure) is fed through the GP model (top right figure). The resulting exact predictive distribution $p(x_{t+1})$ is given as the green shaded area in the top left figure. The mean and covariance can be computed with Eq. (2-38) and Eq. (2-41), which are used to approximate the predictive distribution $p(x_{t+1})$ with a Gaussian, the blue distribution in top left figure.

Chapter 3

The PILCO algorithm

Gaussian Process (GP) models as presented in the previous chapter are used in the Probabilistic Inference for Learning COntrol (PILCO) algorithm because: (1) GP models are capable of modeling a wide spread of nonlinear systems including their uncertainties, and (2) GP modeling is based on well understood approximation principles. The PILCO algorithm consists of two main parts: GP system identification and model-based Reinforcement Learning (RL). The second part of the algorithm learns a single nonlinear controller using a gradient based policy search. This chapter gives insight in the PILCO algorithm, as presented in (Deisenroth and Rasmussen, 2009; Deisenroth, 2010; Deisenroth and Rasmussen, 2011). In this chapter first the essence of the algorithm is presented. Subsequently, the PILCO algorithm is explained in detail using the devision of the two parts and the pseudo code of the complete algorithm is given. The chapter concludes with a discussion on the PILCO algorithm.

3-1 The core of the PILCO algorithm

Although, modeling of the GP model, is described in detail in Chapter 2, a difference should be emphasized. The *training targets* used in the PILCO algorithm are the *increments* of the state updates instead of the state updates itself. As will be explained in Section 3-2 this implies a different one-step update rule for simulation of the model. The unknown function that describes the dynamics of the real set-up can be written as:

$$x_t = f(x_{t-1}, u_{t-1}), \tag{3-1}$$
$$y_t = x_t + \epsilon_t, \tag{3-2}$$

where x defines the state and u the input. The goal of the first part is to find a GP model that approximates this function $f(x)$ accurately. The goal of the second part is to find (learn) a deterministic control strategy (policy), that successfully achieves a certain task using model based RL. This is done by minimizing the expected return

$$J^{\pi}(x_t, \psi) = \sum_{t=0}^{T} \mathbb{E}[c(x_t)], \quad x_0 \sim \mathcal{N}(\mu_0, \Sigma_0), \tag{3-3}$$

where $c(x)$ is a cost function as defined in Section 3-3-3. After convergence the (sub)optimal policy is used for a trial on the real system, and the in- and output data is collected. The received data from all the trials (the just received new data and the old data are combined) is used to find a new GP model. This new model is used for a model-based policy search. This process repeats itself till the task is learned.

3-2 Part 1: GP System Identification

During the learning process the algorithm uses *all* recorded data at that moment. So assuming trials of 2.5 seconds the first model will be found using the initial trial data of the first trial. After the second trial – this is the first trial with an optimized controller – both data sets are used. This implies 5 seconds of in-output data. After the third trial, 7.5 seconds and so on. Although the algorithm learns a GP model using Evidence Maximization as in Chapter 2-2-3, there is an important difference. The PILCO algorithm uses the *increments* of the state update as *training targets*,

$$\Delta_t = x_t - x_{t-1} + \epsilon_t, \tag{3-4}$$

where ϵ_t is the assumed independent and identically distributed (iid) measurement noise with the distribution $\epsilon_t \sim \mathcal{N}(0, \Sigma_\epsilon)$, $\Sigma_\epsilon = \operatorname{diag}(\sigma_{\epsilon_1}, ..., \sigma_{\epsilon_D})$. In Section 3-5-1 a motivation for using state increments as training targets is given. The GP model requires state-action tuples, $\tilde{x} = (x, u)$, as inputs with $x \in \mathbb{R}^D$ and $u \in \mathbb{R}^F$. Define the training inputs as the vector of tuples $\tilde{X}_i = (x_i, u_i) \in \mathbb{R}^{D+F}$, $\tilde{X} = [\tilde{X}_1, ..., \tilde{X}_n]^T$. The corresponding training targets are the *increments* of the state updates $\mathbf{y} = [\Delta_i, ..., \Delta_n]^T \in R^{n\times E}$ as presented in Eq. (3-4). Note that the dimension of the training inputs and targets do not need to be the same, see Section 4-2-2. In the PILCO algorithm the prior mean function is chosen to be zero and the prior covariance function is the Squared Exponential (SE) covariance function:

$$m(\tilde{x}) \equiv 0, \tag{3-5}$$

$$k(\tilde{x}, \tilde{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}[\tilde{x} - \tilde{x}']^T \Lambda^{-1} [\tilde{x} - \tilde{x}']\right) + \delta_{pq}\sigma_\epsilon^2. \tag{3-6}$$

The δ_{pq} is the Kronecker delta for time indices p and q, which is unity when $p = q$ and zero otherwise. This follows from the assumption that the measurements noise is independent. The variance of the latent function f is given as σ_f^2. The matrix $\Lambda = \operatorname{diag}\left(\ell_1^2, ..., \ell_{D+F}^2\right)$, which is dependent of the different characteristic length-scales ℓ_i, is a weighting matrix. In order to fit the model on the data the most suitable hyper-parameters, $[\ell_1, ..., \ell_{D+F}\ \sigma_f\ \sigma_\epsilon]^T$, for every target dimension (E) should be found. The found (sub)optimal hyper-parameters are combined in the hyper-parameter vector $\theta \in \mathbb{R}^{E*(D+F+2)\times 1}$. The hyper-parameters can be found in the same manner as in Section 2-2-3, using Evidence Maximization. Because, the GP model predict the increment of the state update, instead of the state update, the one-step update expressions of Eq. (2-31) will change. The mean of the next state is found by adding the prediction to the current state. The same holds for the variance of the prediction. However, note the following property,

$$\operatorname{var}[X + Y] = \operatorname{var}[X] + \operatorname{var}[Y] + \operatorname{cov}[X, Y] + \operatorname{cov}[Y, X]. \tag{3-7}$$

For deterministic inputs the $\operatorname{cov}[\Delta_t, x_t] = 0$. Hence, the GP model *one-step update equations* for a *deterministic state-action pair* (test input) $\tilde{x}_{t+1} = (x_t, u_t)$, becomes

$$p(x_{t+1}|\tilde{x}_t) \sim \mathcal{N}(\mu_{t+1}, \Sigma_{t+1}), \tag{3-8a}$$
$$\mu_{t+1} = x_t + \mathbb{E}[\Delta_{t+1}], \tag{3-8b}$$
$$\Sigma_{t+1} = \operatorname{cov}[\Delta_{t+1}]. \tag{3-8c}$$

The model is completely determined by its mean μ_{t+1} and covariance Σ_{t+1}. This implies that the model is determined by the hyper-parameter vector θ. Note, that similar to Section 2-3-2 the output of the GP model is Gaussian distributed, even for deterministic inputs. In order to use these uncertainties, the GPmodel must be able to pass these uncertainties through the model. The *one-step update equations* for an *uncertain state-action pair* (test input) $\tilde{x}_t \sim \mathcal{N}(\tilde{\mu}_t, \tilde{\Sigma}_t)$, becomes

$$p(x_{t+1}|\tilde{\mu}_t, \tilde{\Sigma}_t) \sim \mathcal{N}(\mu_{t+1}, \Sigma_{t+1}), \tag{3-9a}$$
$$\mu_{t+1} = x_t + \mathbb{E}[\Delta_{t+1}], \tag{3-9b}$$
$$\Sigma_{t+1} = \Sigma_t + \operatorname{cov}[\Delta_{t+1}] + \operatorname{cov}[\Delta_{t+1}, x_t] + \operatorname{cov}[x_t, \Delta_{t+1}]. \tag{3-9c}$$

The update rule for the covariance matrix, Eq. (3-9c), can be found using Eq. (3-7). In the next section the computations will be presented which are necessary to find μ_{t+1} and Σ_{t+1} from $\tilde{\mu}_t$ and $\tilde{\Sigma}_t$.

3-2-1 Multivariate predictions of the state increments

By propagating the uncertain state-action (test) input $p(x_t, u_t)$ through the GP model the predictive distribution $p(\Delta_{t+1}) = \mathcal{N}(\mathbb{E}[\Delta_{t+1}], \mathrm{cov}\,[\Delta_{t+1}])$ can be found, see (Deisenroth, 2010). Assume for now that the distribution of the state-action pair is known, see Section 3-3 how to find this distribution. The predictive distribution is approximated using moment matching, see Figure 2-6. The state dimension is generally larger than one, $D > 1$, which implies multivariate predictions.

Mean prediction $\mathbb{E}[\Delta_{t+1}]$

The predictive mean vector consists of E independent predictive means of Eq. (2-38):

$$\mathbb{E}[\Delta_{t+1}] = \left[\beta_1^T q_1, ..., \beta_E^T q_E\right]^T . \tag{3-10}$$

For every dimension $a = 1 : E$ holds:

$$\beta_a = (K_a + \sigma_{\epsilon,a}^2 I)^{-1}\mathbf{y}, \tag{3-11}$$
$$q_a = [q_a^1, ..., q_a^n]^T, \tag{3-12}$$
$$q_a^i = \sigma_{f,a}^2 |\tilde{\Sigma}_t \Lambda_a^{-1} + I|^{\frac{1}{2}} \exp\left(-\frac{1}{2}(\tilde{X}_i - \tilde{\mu}_t)^T(\tilde{\Sigma}_t + \Lambda_a)^{-1}(\tilde{X}_i - \tilde{\mu}_t)\right). \tag{3-13}$$

Covariance matrix of the prediction $\mathrm{cov}\,[\Delta_{t+1}]$

The covariance matrix of the predictive distribution is given by:

$$\begin{aligned}\mathrm{cov}\,[\Delta_{t+1}]_{ab} &= \mathrm{cov}\left[\Delta_{t+1}^a, \Delta_{t+1}^b\right] \\ &= \begin{cases} \beta_a^T Q \beta_b - \mathbb{E}[\Delta_{t+1}^a]\mathbb{E}[\Delta_{t+1}^b], & a \neq b \\ \underbrace{\beta_a^T Q \beta_a - \mathbb{E}[\Delta_{t+1}^a]^2}_{\mathrm{cov}\left[\mathbb{E}[\Delta_{t+1}^a, \Delta_{t+1}^b|\tilde{x}_t]|\tilde{\mu}_t, \tilde{\Sigma}_t\right]} + \underbrace{\sigma_{f,a}^2 - \mathrm{tr}\left((K_a + \sigma_{\epsilon,a}^2 I)^{-1}Q\right)}_{\mathbb{E}\left[\mathrm{cov}\left[\Delta_{t+1}^a, \Delta_{t+1}^b|\tilde{x}_t\right]|\tilde{\mu}_t, \tilde{\Sigma}_t\right]}, & a = b \end{cases},\end{aligned} \tag{3-14}$$

where the Q entries for multivariate predictions are

$$Q_{ij} = \frac{K_a(\tilde{X}_i, \tilde{\mu}_t)K_b(\tilde{X}_j, \tilde{\mu}_t)}{\sqrt{|R|}} \exp\left(\frac{1}{2} z_{ij}^T R^{-1} \tilde{\Sigma}_t z_{ij}\right). \tag{3-15}$$

Which can be rewritten for a numerical stable implementation as

$$Q_{ij} = \frac{\exp\left(n_{ij}^2\right)}{\sqrt{|R|}}, \tag{3-16}$$

$$n_{ij}^2 = 2(\log(\sigma_{f,a}) + \log(\sigma_{f,b})) - \frac{\zeta_i^T \Lambda_a^{-1} \zeta_i + \zeta_j^T \Lambda_b^{-1} \zeta_j - z_{ij}^T R^{-1} z_{ij}}{2}, \tag{3-17}$$

where the following is defined:

$$R = \tilde{\Sigma}_t(\Lambda_a^{-1} + \Lambda_b^{-1}) + I, \qquad \zeta_i = (\tilde{X}_i - \tilde{\mu}_t), \qquad z_{ij} = \Lambda_a^{-1}\zeta_i + \Lambda_b^{-1}\zeta_j. \tag{3-18}$$

Note, that for equal target dimensions, ($a = b$), the expression Q of Eq. (3-14) is equal to $\hat{Q}$ of Eq. (2-42). However for unequal target dimension ($a \neq b$), $\mathbb{E}\left[\text{cov}\left[\Delta_{t+1}^a, \Delta_{t+1}^b|\tilde{x}_t\right]|\tilde{\mu}_t, \tilde{\Sigma}_t\right]$ is zero, because of the assumption that the target dimensions are conditionally independent given the input.

Covariance of the current state and target $\text{cov}\left[x_t, \Delta_{t+1}\right]$

The covariance matrix of the input $\tilde{x}_t$ and target Δ_{t+1} is given in (Deisenroth, 2010, Sec. 2.3.3). However, for the computations of the next state in Eq. (3-9c) the covariance of the *current state*, x_t, and the target, $\text{cov}\left[x_t, \Delta_{t+1}\right]$, is required. This covariance matrix can be found by 'slicing out' the correct dimensions of the covariance matrix of the input and target $\text{cov}\left[\tilde{x}_t, \Delta_{t+1}\right]$. The expression for the covariance between the input and target is

$$\text{cov}\left[\tilde{x}_t, \Delta_{t+1}\right] = \mathbb{E}[\tilde{x}_t\Delta_{t+1}^T] - \mathbb{E}[\tilde{x}_t]\mathbb{E}[\Delta_{t+1}]^T. \tag{3-19}$$

The left part of the right hand side of Eq. (3-19) can be computed for every dimension of the target space, $a = 1, .., E$, as

$$\mathbb{E}[\tilde{x}_t\Delta_{t+1}^a|\tilde{\mu}_t\tilde{\Sigma}_t] = \mathbb{E}[\tilde{x}_t \underbrace{\mathbb{E}[\Delta_{t+1}^a|\tilde{x}_t]}_{=Eq.\ (2-32)}|\tilde{\mu}_t\tilde{\Sigma}_t] = \int \tilde{x}_t\left(\sum_{i=1}^n \beta_{a,i}k_a(\tilde{x}_t, \tilde{X}_i)\right)p(\tilde{x}_t)d\tilde{x}_t \tag{3-20}$$

$$= \sum_{i=1}^n \beta_{a,i}\int \tilde{x}_t \underbrace{c_1\mathcal{N}(\tilde{X}_i, \Lambda_a)}_{k_a(\tilde{x}_t, \tilde{X}_i)}\underbrace{\mathcal{N}(\tilde{\mu}_t, \tilde{\Sigma}_t)}_{p(\tilde{x}_t)}d\tilde{x}_t, \tag{3-21}$$

where the normalizing constant for $k_a(\tilde{x}_t, \tilde{X}_i)$ is $c_1^{-1} = \sigma_f^{-2}(2\pi)^{-\frac{D+F}{2}}|\Lambda_a|^{-\frac{1}{2}}$. The product of the two Gaussian distributions in Eq. (3-21) can be found as the Gaussian $c_2^{-1}\mathcal{N}(\omega, \Omega)$:

$$c_2^{-1} = (2\pi)^{-\frac{D+F}{2}}\exp\left(-\frac{1}{2}(\tilde{X}_i - \tilde{\mu}_t)^T(\Lambda_a + \tilde{\Sigma}_t)^{-1}(\tilde{X}_i - \tilde{\mu}_t)\right), \tag{3-22}$$

$$\Omega = (\Lambda_a^{-1} + \tilde{\Sigma}_t^{-1})^{-1}, \tag{3-23}$$

$$\omega_i = \Omega(\Lambda_a^{-1}\tilde{X}_i + \tilde{\Sigma}_t^{-1}\tilde{\mu}_t). \tag{3-24}$$

Which results in:

$$\mathbb{E}[\tilde{x}_t\Delta_{t+1}^a|\tilde{\mu}_t\tilde{\Sigma}_t] = \sum_{i=1}^n c_1c_2^{-1}\beta_{a,i}\omega_i, \qquad a = 1, ..., E. \tag{3-25}$$

Note, that $c_1c_2^{-1} = q_a^i$, see Eq. (3-13), ω_i can be rewritten as $\omega_i = \tilde{\Sigma}_t(\tilde{\Sigma}_t + \Lambda_a)^{-1}\tilde{X}_i + \Lambda(\Lambda + \tilde{\Sigma}_t)^{-1}\tilde{\mu}_t$ and $\mathbb{E}[\Delta_{t+1}^a] = \beta_a^T\mathbf{q}_a$. Hence, the covariance matrix can be written as

$$\begin{aligned}\text{cov}\left[\tilde{x}_t, \Delta_{t+1}\right] &= \sum_{i=1}^n \beta_{a,i}q_{a,i}(\tilde{\Sigma}_t(\tilde{\Sigma}_t + \Lambda_a)^{-1}\tilde{X}_i + (\Lambda_a(\tilde{\Sigma}_t + \Lambda_a)^{-1} - I)\tilde{\mu}_t) \\ &= \sum_{i=1}^n \beta_{a,i}q_{a,i}(\tilde{\Sigma}_t(\tilde{\Sigma}_t + \Lambda_a)^{-1}(\tilde{X}_i - \tilde{\mu}_t) + (\Lambda_a(\tilde{\Sigma}_t + \Lambda_a)^{-1} + \tilde{\Sigma}_t(\tilde{\Sigma}_t + \Lambda_a)^{-1} - I)\tilde{\mu}_t).\end{aligned} \tag{3-26}$$

Using.

$$(\Lambda_a(\tilde{\Sigma}_t + \Lambda_a)^{-1} + \tilde{\Sigma}_t(\tilde{\Sigma}_t + \Lambda_a)^{-1} - I) = (\Lambda_a + \tilde{\Sigma}_t)(\Lambda_a + \tilde{\Sigma}_t)^{-1} - I = 0, \tag{3-27}$$

the compact notation for the covariance becomes

$$\text{cov}\left[\tilde{x}_t, \Delta_{t+1}\right] = \sum_{i=1}^{n} \beta_{a,i} q_{a,i} (\tilde{\Sigma}_t(\tilde{\Sigma}_t + \Lambda_a)^{-1}(\tilde{X}_i - \tilde{\mu}_t)). \tag{3-28}$$

By slicing out the correct dimensions the covariance between the current state and the prediction can be found:

$$\text{cov}\left[x_t, \Delta_{t+1}\right] \subset \text{cov}\left[\tilde{x}_t, \Delta_{t+1}\right]. \tag{3-29}$$

With the mean $\mathbb{E}[\Delta_{t+1}]$ Eq. (3-10) and covariance $\text{cov}\left[\Delta_{t+1}\right]$ Eq. (3-14) of the predictive distribution $p(\Delta_{t+1})$, and the covariation between the current state and predicted target $\text{cov}\left[x_t, \Delta_{t+1}\right]$ Eq. (3-29), the GP model is defined. Using Eq. (3-9), the distribution of the next state $p(x_{t+1}) = \mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$ can be approximated.

3-3 Part 2: Model-based Reinforcement Learning

The GP model is used to find the policy that performs the task successfully. Hence, the (sub)optimal controller parameters ψ that minimize Eq. (3-3) need to be found. During the optimization the model is used to simulate the dynamics of the real system by cascading one-step predictions. Note, that the even with starting from a deterministic initial condition the successive state will be Gaussian distributed. This implies that the GP model should map Gaussian distributed states to targets:

$$\underbrace{p(x_t) \to p(\pi_t)}_{\text{Section 3-3-1}} \overbrace{\to p(u_t) \to}^{\text{Section 3-3-2}} \underbrace{p(x_t, u_t) \to p(\Delta_{t+1}) \to p(x_{t+1})}_{\text{GP one-step prediction Eq. (3-9)}}. \tag{3-30}$$

In Section 3-3-2 the control input is saturated using a saturation function, $u_t = S(\pi_t)$. Hence, the required joint distribution of the state-action pair is

$$p(x_t, u_t) = \mathcal{N}\left(\begin{bmatrix}\mu_{x_t} \\ \mu_{u_t}\end{bmatrix}, \begin{bmatrix}\Sigma_{x_t} & \Sigma_{x_t,u_t} \\ \Sigma_{x_t,u_t}^T & \Sigma_{u_t}\end{bmatrix}\right), \tag{3-31}$$

where $p(x_t)$ is known at the current time step and by using Eq. (3-9b) and (3-9c) the next state can be approximated as a Gaussian $x_{t+1} \sim \mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$. However, there is an unknown distribution $p(u_t)$, the saturated control input. The next section presents how to feed the uncertain state ($x_t \sim \mathcal{N}(\mu_t, \Sigma_t)$) through the controller $\pi_t = \pi(x_t)$ to find the unsaturated control input distribution $p(\pi_t)$.

3-3-1 Controllers

Different controllers can be used in the algorithm. By pushing the uncertainties of the state through the controller, the joint distribution of the states and the controller can be found:

$$p(x_t, \pi_t) = \mathcal{N}\left(\begin{bmatrix}\mu_{x_t} \\ \mu_{\pi_t}\end{bmatrix}, \begin{bmatrix}\Sigma_{x_t} & \Sigma_{x_t,\pi_t} \\ \Sigma_{x_t,\pi_t}^T & \Sigma_{\pi_t}\end{bmatrix}\right). \tag{3-32}$$

In absence of a saturation function the joint distribution of the state and unsaturated control input is used in the GP one step prediction, $p(x_t, \pi_t) = p(x_t, u_t)$. The joint distribution of the state and the control input is depended on the kind of controller, and will be explained in detail. The *linear controller*, *Radial Basis Function (RBF) controller* and *GP controller* will be discussed.

Linear controller

The linear controller, or to be mathematically correct affine controller is given by

$$\pi_t = \pi(x_t, \psi) = wx_t + b. \tag{3-33}$$

Where w is a matrix of weights, b an vector of offsets and $\psi = [w\ b] \in \mathbb{R}^{D+1}$ the parameter vector. So for every control dimensions the control input is a linear combination of the states plus an offset. In this example is the control dimension one, so w is a row vector and b a scalar. The controller input has to be determined for an uncertain input $x_t = \mathcal{N}(\mu_t, \Sigma_t)$. This can be done by propagating the uncertain state trough the controller:

$$\mathbb{E}[\pi_t|\mu_t, \Sigma_t] = \mathbb{E}[\pi(x_t, \psi)|\mu_t, \Sigma_t] = \mathbb{E}[wx_t + b|\mu_t, \Sigma_t] = w\mu_t + b. \tag{3-34}$$

Because of the uncertainty in the state, the control input has a variance:

$$\begin{aligned} \operatorname{var}[\pi_t|\mu_t, \Sigma_t] &= \mathbb{E}[(\pi(x_t, \psi) - \mathbb{E}[\pi(x_t, \psi)|\mu_t, \Sigma_t])^2\,|\mu_t, \Sigma_t] && (3\text{-}35)\\ &= w\mathbb{E}[(x_t - \mu)(x_t - \mu)^T|\mu_t, \Sigma_t]w^T = w\Sigma_t w^T. && (3\text{-}36) \end{aligned}$$

As preparation to approximate the joint distribution of the state-action pair which is needed as input for the GP model, see Eq. (3-30), the covariance between state and control input can be computed as

$$\begin{aligned} \operatorname{cov}[x_t, \pi_t|\mu_t, \Sigma_t] &= \mathbb{E}[x_t\pi_t^T|\mu_t, \Sigma_t] - \mathbb{E}[x_t|\mu_t, \Sigma_t]\mathbb{E}[\pi_t|\mu_t, \Sigma_t]^T && (3\text{-}37)\\ &= \int x_t\pi(x_t, \psi)p(x_t)dx_t - \mathbb{E}[x_t|\mu_t, \Sigma_t]\mathbb{E}[\pi_t|\mu_t, \Sigma_t]^T && (3\text{-}38)\\ &= \int x_t(wx_t + b)^T p(x_t)dx_t - \mu_t(w\mu_t + b)^T && (3\text{-}39)\\ &= \mathbb{E}[(x_t x_t^T - \mu_t\mu_t^T)w^T + (x_t - \mu_t)b^T|\mu_t, \Sigma_t] && (3\text{-}40)\\ &= \mathbb{E}[(x_t x_t^T - \mu_t\mu_t^T)|\mu_t, \Sigma_t]w^T = \Sigma_t w^T. && (3\text{-}41) \end{aligned}$$

This controller is suitable for stabilizing a system in an equilibrium point. For more advanced tasks different controllers should be chosen.

Radial basis functions controller

The RBF controller is more flexible than the linear controller, which implies that harder task can be performed. This controller is a summation of Gaussian functions and is very similar to the GP model:

$$\begin{aligned} \pi_t = \pi(x_t, \psi) &= \sum_{i=1}^{n} w_i\phi_i(x_t) = \mathbf{w}^T\phi, && (3\text{-}42)\\ \phi_i(x_t) &= \exp\left(-\frac{1}{2}(x_{\pi,i} - x_t)^T\Lambda^{-1}(x_{\pi,i} - x_t)\right). && (3\text{-}43) \end{aligned}$$

The centers of the basis functions are given as $x_{\pi,i}$, the weighting vector as w and Λ is a matrix containing the length-scales. The dimension of the hyper-parameters for n basis functions and $x_t \in \mathbb{R}^D$ results in $w \in \mathbb{R}^n$, $x_\pi \in \mathbb{R}^{D*n}$ and $\Lambda \in \mathbb{R}^D$. Where $\Lambda = \operatorname{diag}\left(\ell_1^2, ..., \ell_D^2\right)$. So the hyper-parameter vector ψ of the controller has dimension $\psi = \{w,\ \ell_1, ..., \ell_D,\ x_\pi\} \in \mathbb{R}^{D+n(D+1)}$. By varying the centers $x_{\pi,i}$, characteristic length-scales ℓ_i and the weighting factor w_i for $i = 1 : D$ the controller that minimizes the expected return, Eq. (3-3), can be found. When feeding an uncertain state, $x_t = \mathcal{N}(\mu_t, \Sigma_t)$, through the controller the uncertain output of the controller can be found as a Gaussian distribution with the mean

$$\begin{aligned}\mathbb{E}[\pi_t|\mu_t,\Sigma_t] &= \mathbb{E}[\pi_t|\mu_t,\Sigma_t] = \mathbb{E}[\pi(x_t)|\mu_t,\Sigma_t] = \mathbb{E}[\mathbf{w}^T\phi|\mu_t,\Sigma_t] \\ &= \sum_{i=1}^{n} w_i|\Sigma_t\Lambda^{-1}+I|^{\frac{1}{2}} \exp\Big(-\frac{1}{2}(x_{\pi,i}-\mu_t)^T(\Sigma_t+\Lambda)^{-1}(x_{\pi,i}-\mu_t)\Big).\end{aligned} \tag{3-44}$$

Due to uncertainties in the state the controller has a variance. By following Eq. (2-39) the variance can be found as

$$\begin{aligned}\text{var}\,[\pi_t|\mu_t,\Sigma_t] &= \text{var}\,[\pi_t|\mu_t,\Sigma_t] = \mathbb{E}[\underbrace{\text{var}\,[\pi_t|x_t]}_{=0}|\mu_t,\Sigma_t] + \text{var}\,[\mathbb{E}[\pi_t|x_t]|\mu_t,\Sigma_t] \\ &= \mathbf{w}^TQ\mathbf{w} - \mathbb{E}[\pi_t|\mu_t,\Sigma_t]^2.\end{aligned} \tag{3-45}$$

Q is as defined in Eq. (3-16). Because the RBF controller is a deterministic controller, the variance for a deterministic input is zero var $[\pi_t|x_t]$. Similar to the input-output covariance matrix of the GP model prediction of Eq. (3-28), the covariance between the state and action is found as

$$\text{cov}\,[x_t,\pi_t|\mu_t,\Sigma_t] = \sum_{i=1}^{n} w_i|\Sigma_t\Lambda^{-1}+I|^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(x_{\pi,i}-\mu_t)^T(\Sigma_t+\Lambda)^{-1}(x_{\pi,i}-\mu_t)\Big)\,\Sigma_t(\Sigma_t+\Lambda)^{-1}(x_{\pi,i}-\mu_t). \tag{3-46}$$

Gaussian Process controller

The mean prediction of the GP model and the RBF controller are rather similar. The $\mathbf{w}$ is the β in Eq. (2-38) and remaining part of the equation is equal to $\mathbf{q}$ in Eq. (2-38) with $\sigma_f^2 = 1$ and instead of the centers of the RBF's the training inputs X:

$$\pi_t = \pi(x_t,\psi) = \sum_{i=1}^{n}\beta_{\pi,i}k(x_\pi,x_t) = \beta_\pi^T K(X_\pi,x_t), \tag{3-47}$$

$$k(x_\pi,x_t) = \sigma_{f,\pi}^2 \exp\Big(-\frac{1}{2}(x_{\pi,i}-x_t)^T\Lambda^{-1}(x_{\pi,i}-x_t)\Big), \tag{3-48}$$

$$\beta_\pi = (K_\pi(X_\pi,X_\pi)+\sigma_{\epsilon,\pi}^2 I)^{-1}\mathbf{y}_\pi, \tag{3-49}$$

where $\sigma_{f,\pi}^2$ is fixed to one and $\sigma_{\epsilon,\pi}^2$ is the measurement noise variance. The hyper-parameters for this controller are the training inputs[1] $X_\pi \in \mathbb{R}^{n*D}$, training targets $\mathbf{y}_\pi \in \mathbb{R}^n$ and the length scales of $\Lambda \in \mathbb{R}^D$, $\Lambda = \text{diag}\left(\ell_1^2,...,\ell_D^2\right)$. This gives the dimension of the hyper-parameters, $\psi = \{\mathbf{y}_\pi,\ \ell_1,...,\ell_D,\ X_\pi\ \sigma_{\epsilon,\pi}^2\} \in \mathbb{R}^{D+1+n(D+1)}$. The matrix $K_\pi(X_\pi,X_\pi)+\sigma_{\epsilon,\pi}^2 I$ has full rank, if $\sigma_{\epsilon,\pi}^2 \neq 0$. Hence, The prediction of the GP model is functionally equivalent as the RBF network if $\sigma_f^2 = 1$ and $\sigma_{\epsilon,\pi}^2 \neq 0$.
The implementation of the noise variance $\sigma_{\epsilon,\pi}^2$ in the indirect description could lead to smoothing of the cost function of the gradient based policy search (Section 3-3-3). In Chapter 4 the convergence of the RBF and GP controllers are compared in order to investigate the possible advantage of the smoothing. The mean, variance and the covariance between the state and action of the GP controller due to the uncertainty of the input can be found similar to Eq. (2-38), (3-45) and (3-46) as:

$$\mathbb{E}[\pi_t|\mu_t,\Sigma_t] = \beta_\pi^T q_\pi, \tag{3-50}$$

$$\text{var}\,[\pi_t|\mu_t,\Sigma_t] = \beta_\pi^T Q\beta_\pi - (\beta_\pi^T q_\pi)^2, \tag{3-51}$$

$$\text{cov}\,[x_t,\pi_t|\mu_t,\Sigma_t] = \sum_{i=1}^{n}\beta_{\pi,i}q_{\pi,i}\Sigma_t(\Sigma_t+\Lambda)^{-1}(x_{\pi,i}-\mu_t). \tag{3-52}$$

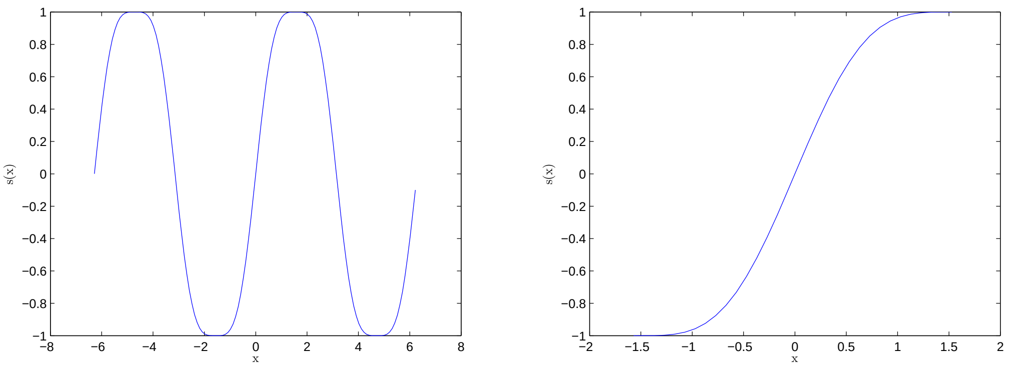
[1]Note, that the training inputs for the GP controller do *not* include the inputs u_t. Therefore, X_π is written without a tilde.

3-3-2 Saturation of the controller

When designing a controller for a real set-up, the physical boundaries have to be taken into account. For these reasons the control input has to be limited on a certain interval, $[-u_{max}, u_{max}]$. The saturation function, that limits the control input, has to be on a finite interval, such that a maximum and minimum are obtained for finite function inputs. Furthermore, is a monotonically increasing function required. The derivative and second derivative of the saturation function have to be zero at the boundary points to require stationary points at this boundaries. Consider the third-order Fourier series expansion of a trapezoidal wave (Deisenroth et al., 2013b), see Figure 3-1. Given the boundary conditions and by normalizing the function to the maximum control input interval $[-u_{max}, u_{max}]$ the *saturation function* is found as

$$S(x) = u_{max}\frac{(9\sin(x) + \sin(3x))}{8}. \tag{3-53}$$

If the function is considered on the domain, $[-\frac{\pi}{2}, \frac{\pi}{2}]$ the function is monotonically increasing, see Figure 3-1b.



(a) Normalized third-order Fourier series expansion of a trapezoidal function.

(b) Normalized third-order Fourier series expansion of a trapezoidal function on the domain $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

Figure 3-1: Visualization of the saturation function of the control input. In Figure 3-1a is the third-order Fourier series expansion is given. In Figure 3-1b the third-order Fourier series expansion is given for the monotonically increasing domain $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

For the uncertain state an uncertain control input is found, see Section 3-3-1,

$$p(\pi(x_t, \psi) = \mathcal{N}(\mu_t^\pi, \Sigma_t^\pi) = \mathcal{N}(\mathbb{E}[\pi(x_t, \psi)|\mu_t, \Sigma_t], \mathrm{var}\,[\pi(x_t, \psi)|\mu_t, \Sigma_t]). \tag{3-54}$$

This control input has to be squeezed by the saturation function to find the squeezed control input which is applied to the system, $u_t = S(p(\pi(x, \psi))$. This implies that the uncertainty has to pass through the saturation function. Consider the scaler control input $\pi_t \sim \mathcal{N}(\mu_t^\pi, \Sigma_t^\pi)$ that has to be squeezed by the saturation function. A helpful joint distribution for the upcoming equations can be formed,

$$p(\pi_t, 3\pi_t) = \mathcal{N}\left(\begin{bmatrix}\mu_t^\pi \\ 3\mu_t^\pi\end{bmatrix}, \begin{bmatrix}\Sigma_t^\pi & 3\Sigma_t^\pi \\ 3\Sigma_t^\pi & 9\Sigma_t^\pi\end{bmatrix}\right). \tag{3-55}$$

Using this joint distribution the mean of the saturation function can be found as

$$\mathbb{E}[S(\pi_t)|\mu_t^\pi,\Sigma_t^\pi] = u_{max}\frac{\left(9\,\overbrace{\mathbb{E}[\sin(\pi_t)|\mu_t^\pi,\Sigma_t^\pi]}^{Eq.\ (A-4)}+\overbrace{\mathbb{E}[\sin(3\pi_t)|3\mu_t^\pi,9\Sigma_t^\pi]}^{Eq.\ (A-4)}\right)}{8} \tag{3-56}$$

$$= u_{max}\frac{\left(9\exp\left(-\frac{\Sigma_t^\pi}{2}\right)\sin(\mu_t^\pi)+\exp\left(-\frac{9\Sigma_t^\pi}{2}\right)\sin(3\mu_t^\pi)\right)}{8}. \tag{3-57}$$

The variance is found using the property, Eq. (3-7), the variance of the saturation function becomes

$$\text{var}\left[S(\pi_t)|\mu_t^\pi,\Sigma_t^\pi\right] = \text{var}\left[u_{max}\frac{(9\sin(\pi_t)+\sin(3\pi_t))}{8}|\mu_t^\pi,\Sigma_t^\pi,3\mu_t^\pi,9\Sigma_t^\pi\right] \tag{3-58}$$

$$= \text{var}\left[u_{max}\frac{9\sin(\pi_t)}{8}|\mu_t^\pi,\Sigma_t^\pi\right]+\text{var}\left[u_{max}\frac{\sin(3\pi_t)}{8}|3\mu_t^\pi,9\Sigma_t^\pi\right] \tag{3-59}$$

$$+\,\text{cov}\left[u_{max}\frac{9\sin(\pi_t)}{8},u_{max}\frac{\sin(3\pi_t)}{8}|\mu_t^\pi,\Sigma_t^\pi,3\mu_t^\pi,9\Sigma_t^\pi\right] \tag{3-60}$$

$$+\,\text{cov}\left[u_{max}\frac{\sin(3\pi_t)}{8},u_{max}\frac{9\sin(\pi_t)}{8}|3\mu_t^\pi,9\Sigma_t^\pi,\mu_t^\pi,\Sigma_t^\pi\right]. \tag{3-61}$$

Where the variances in Eq. (3-59) can be solved by using Eq. (A-6). The covariances of Eq. (3-60) and (3-61) can be computed with

$$\text{cov}\left[\sin(\pi_t),\sin(3\pi_t)|\mu_t,\Sigma_t,3\mu_t,9\Sigma_t\right] = \mathbb{E}[\sin(\pi_t)\sin(\pi_t)|\mu_t,\Sigma_t,3\mu_t,9\Sigma_t]-\mathbb{E}[\sin(\pi_t)|\mu_t,\Sigma_t]\mathbb{E}[\sin(3\pi_t)|3\mu_t,9\Sigma_t]. \tag{3-62}$$

This can be solved by using

$$\sin(\theta)\sin(\phi)=\frac{\cos(\theta-\phi)-\cos(\theta+\phi)}{2} \qquad \text{and} \qquad 5\Sigma_t^\pi=\frac{\Sigma^\pi+9\Sigma_t^\pi}{2}. \tag{3-63}$$

The left hand side of the covariance of Eq. (3-62) can be computed with

$$\mathbb{E}[\sin(x)\sin(x)|\mu_t,\Sigma_t,3\mu_t,9\Sigma_t] = u_{max}^2\frac{9}{128}\left(\exp\left(-(5\Sigma_t^\pi-3\Sigma_t^\pi)\right)\cos(\mu_t^\pi-3\mu_t^\pi)-\exp\left(-(5\Sigma_t^\pi+3\Sigma_t^\pi)\right)\cos(\mu_t^\pi+3\mu_t^\pi)\right). \tag{3-64}$$

With the covariance found, the distribution of the squeezed control input $p(u_t) = p(S(\pi_t))$ is completely determined. However to be able to use this control input for simulation, the joint distribution of the control input and the states has to be computed. Known is the joint distribution of the unsqueezed controller and the states, $p(x_t,\pi_t)$, see Eq. (3-32). The required joint distribution is the distribution of the states and the squeezed control input, $p(x_t,u_t)$, with $u_t = S(\pi_t)$, which is given in Eq. (3-31). The missing link is the covariance of the state vector and the squeezed control input u_t. Using the known distribution of $p(x_t,\pi_t)$ and $p(x_t)=\mathcal{N}(\mu_t,\Sigma_t)$ the covariance becomes:

$$\text{cov}\left[x_t,u_t|\mu_t,\Sigma_t\right] = \underbrace{\text{cov}\left[x_t,\pi_t|\mu_t,\Sigma_t\right]}_{\text{see Section 3-3-1}}\text{cov}\left[\pi_t,u_t|\mu_t,\Sigma_t\right], \tag{3-65}$$

where,

$$\text{cov}\left[\pi_t,u_t|\mu_t,\Sigma_t\right] = u_{max}\frac{9\mathbb{E}[\cos(\pi)|\mu_t^\pi,\Sigma_t^\pi]}{8}+3u_{max}\frac{\mathbb{E}[\cos(3\pi)|3\mu_t^\pi,9\Sigma_t^\pi]}{8}. \tag{3-66}$$

Which defines the joint distribution of the states and the control input in presence of the saturation function.

3-3-3 Gradient-based policy search

The goal of the second part of the algorithm is to find the policy $\pi(x_t)$ that minimizes the expected return $J^\pi(x_t, \psi)$ Eq. (3-3). This can be done by adjusting the hyper-parameters of the policy ψ such that it minimizes the total expected costs:

$$\psi* = \arg\min J^\pi(x_t, \psi) = \sum_{t=0}^{T} \mathbb{E}[c(x_t)], \quad x_0 \sim \mathcal{N}(\mu_0, \Sigma_0). \tag{3-67}$$

For every evaluation of the current policy the GP model is used to simulate the dynamics of the system. From this virtual trial the expected total cost and the derivatives of the expected total cost with respect to the hyper-parameters of the controller is computed. Using the gradient the search direction is selected. The search is typically non-convex, hence non-convex search methods are required. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm of the `gpml` toolbox (Rasmussen and Williams, 2006) is the used gradient based search method. But also the Conjugate Gradient (CG) method can be used (Deisenroth, 2010; Deisenroth and Rasmussen, 2011). Both methods require the gradient for determining the search direction.

Cost function

The cost function gives a measure for how good it is to be in a certain state and is related to the Euclidean distance between the current state and a so called target state, i.e. goal. The immediate cost is given as

$$c(x_t) = 1 - \exp\left(-\frac{d(x_t - x_{target})^2}{2\sigma_c^2}\right) \in [0, 1], \tag{3-68}$$

where $d(\cdot)$ is the Euclidean distance of its argument and σ_c the width of the valley. As an example the target state for the cart-pole problem would be the inverted position of the pendulum in the center of the track. With x as the difference between the cart position and the center of the track $x = x_{cart} - x_{target}$ and φ the difference between the angle with the target angle $\varphi = \varphi_{pendulum} - \varphi_{target}$ and l the length of the pendulum the Euclidean distance becomes,

$$d(x_t, x_{target})^2 = x^2 + 2x^2 l \sin(\varphi) + 2l^2 + 2l^2 \cos(\varphi). \tag{3-69}$$

The state vector for the cart-pole problem is typically $\mathbf{x}_t = [x_t, \dot{x}_t, \dot{\varphi}_t, \varphi_t]^T$ Measuring the angle of the pendulum anti-clockwise from hanging down, the target state (inverted position of the pendulum in the middle of the track) can be described by the vector $\mathbf{x}_{target} = [0, *, *, \pi]^T$.[2] To ease the implementation of the cost function the state is augmented with the complex values of the angle, $\mathbf{z}_t = [x_t, \dot{x}_t, \dot{\varphi}, \varphi_t, \sin(\varphi), \cos(\varphi)]^T$ and the corresponding target $\mathbf{z}_{target} = [0, *, *, *, 0, -1]^T$ is used. Note that for this new, *full state*, the uncertainties has to be pushed through the $\sin(\cdot)$ and $\cos(\cdot)$ functions. This is presented in Appendix A-1. With the full state and target defined the cost function is given as

$$c(\mathbf{z}_t) = 1 - \exp\left(-\frac{1}{2}(\mathbf{z}_t - \mathbf{z}_{target})^T T^{-1} (\mathbf{z}_t - \mathbf{z}_{target})\right) \in [0,\ 1]. \tag{3-70}$$

With T^{-1}, such that it can be used for vector matrix multiplication to obtain the Euclidean distance of Eq. (3-69) divided by the width of the cost function,

$$T^{-1} = \sigma_c^{-2} \begin{bmatrix} 1 & 0 & 0 & 0 & l & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ l & 0 & 0 & 0 & l^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & l^2 \end{bmatrix}. \tag{3-71}$$

[2]Velocities are not taking into account when computing the cost.

For the uncertain state $p(\mathbf{z}_t) = \mathcal{N}(\mu_t^z, \Sigma_t^z)$ the immediate cost function becomes

$$\mathbb{E}\left[c(\mathbf{z}_t)\right] = \int c(\mathbf{z}_t)p(\mathbf{z}_t)d\mathbf{z}_t \tag{3-72}$$

$$= 1 - |I + \Sigma_t^z T^{-1}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mu_t^z - \mathbf{z}_{target})^T T^{-1}(I + \Sigma_t^z T^{-1})^{-1}(\mu_t^z - \mathbf{z}_{target})\right). \tag{3-73}$$

Gradient of the cost function

To find the parameter vector ψ^* that minimizes the expected return, $J^\pi(x_t, \psi)$, the search direction, in form of the gradient, has to be determined. An expression of the gradient can be found with the chain rule

$$\frac{d\mathbb{E}\left[c(x_t)\right]}{d\psi} = \frac{d\mathbb{E}\left[c(x_t)\right]}{dp(x_t)}\frac{dp(x_t)}{d\psi} = \frac{\partial\mathbb{E}\left[c(x_t)\right]}{\partial\mu_t}\frac{d\mu_t}{d\psi} + \frac{\partial\mathbb{E}\left[c(x_t)\right]}{\partial\Sigma_t}\frac{d\Sigma_t}{d\psi}. \tag{3-74}$$

The first are:

$$\frac{\partial\mathbb{E}[c(x_t)]}{\mu_t} = -2\mathbb{E}[c(x_t)](\mu_t - x_{target})^T T^{-1}(I + \Sigma_t T^{-1})^{-1}, \tag{3-75}$$

$$\frac{\partial\mathbb{E}[c(x_t)]}{\Sigma_t} = \mathbb{E}[c(x_t)](T^{-1}(I + \Sigma_t T^{-1})^{-1}(\mu_t - x_{target})(\mu_t - x_{target})^T - I)T^{-1}(I + \Sigma_t T^{-1})^{-1}. \tag{3-76}$$

The second derivatives are:

$$\frac{d\mu_t}{d\psi} = \frac{\partial\mu_t}{\partial\mu_{t-1}}\frac{d\mu_{t-1}}{d\psi} + \frac{\partial\mu_t}{\partial\Sigma_{t-1}}\frac{d\Sigma_{t-1}}{d\psi} + \frac{\partial\mu_t}{\partial\psi}, \tag{3-77}$$

$$\frac{d\Sigma_t}{d\psi} = \frac{\partial\Sigma_t}{\partial\mu_{t-1}}\frac{d\mu_{t-1}}{d\psi} + \frac{\partial\Sigma_t}{\partial\Sigma_{t-1}}\frac{d\Sigma_{t-1}}{d\psi} + \frac{\partial\Sigma_t}{\partial\psi}. \tag{3-78}$$

The derivatives $\frac{\partial\mu_{t-1}}{\partial\psi}$ and $\frac{\partial\Sigma_{t-1}}{\partial\psi}$ are both computed in the previous time step. Note, that $\mu_t = \mu_{t-1} + \mathbb{E}[\Delta_t]$ and $\Sigma_t = \Sigma_{t-1} + \text{cov}\,[\Delta_t] + \text{cov}\,[\Delta_t, x_{t-1}] + \text{cov}\,[x_{t-1}, \Delta_t]$. Also notice that $\mathbb{E}[\Delta_t]$, $\text{cov}\,[\Delta_t]$, $\text{cov}\,[\Delta_t, x_{t-1}]$ and $\text{cov}\,[x_{t-1}, \Delta_t]$ are dependent of the control input. So this implies that the derivatives vary with the choice of controller. For a more detailed description of the gradient see (Deisenroth, 2010, Sec. 3.6.2).

3-4 Pseudo code of the Probabilistic Inference for Learning COntrol algorithm

In Algorithm (3.1) an overview of the algorithm is given. Note, that the GP system identification and model-based RL steps are alternated till the (sub)optimal solution is found. Note, that with every iteration the observed input-output data from trials is added to the old data. Hence, these data matrices are growing with every iteration. In this way the GP model gathers more information about the system with every iteration, which result in a more accurate model.

Algorithm 3.1: PILCO

```
  For a system with unknown dynamics $x_{t+1} = f(x_t, u_t)$, with $x \in \mathbb{R}^D$ and $u \in \mathbb{R}^F$.
  Define for $N$ input-output data samples, the training inputs as $\tilde{X} = [\tilde{X}_0 ... \tilde{X}_{N-1}] \in \mathbb{R}^{(D+F)\times N}$, with
  $\tilde{X}_i = \begin{bmatrix} x_i & u_i \end{bmatrix}^T$ and the corresponding training targets as $\mathbf{y} = [\Delta_1 ... \Delta_N]^T \in \mathbb{R}^{N\times E}$, with
  $\Delta_i = x_i - x_{i-1} \in \mathbb{R}^E$, target dimension $E$ might be different than state dimension $D$.
  **Initialize**: Controller parameters $\psi$, for $N_c$ Gaussian basis functions: $X_\pi \in \mathbb{R}^{N_c\times D} \sim \mathcal{N}(\mu_0, \Sigma_0)$,
      $\mathbf{y}_\pi \in \mathbb{R}^{N_c\times F} \sim \mathcal{N}(0, I)$ and $\ell = \mathbb{1}_{D,N_c}$.
1 Apply control input $u_t \sim \mathcal{U}(-u_{max}, u_{max})$, save training inputs and targets $(\tilde{X}, \mathbf{y})$ from data
  **repeat**
2     Learn probabilistic (GP) dynamic model (find the hyper-parameters $\theta^* \in \mathbb{R}^{D+F+2\times E}$) using $\tilde{X}$, $\mathbf{y}$.
      **Initialize**: GP hyper-parameters $\forall\ \mathbb{R}^E$: $\ell = \text{var}\left[\tilde{X}\right]$, $\sigma_f = \text{var}\,[\mathbf{y}]$, $\sigma_\epsilon = \text{var}\,[\mathbf{y}/10]$.
          $\Lambda = \text{diag}\left(\{\ell_1^2, ..., \ell_D^2\}\right)$    $K_{ij} = \sigma_f^2 \exp\left(-\frac{1}{2}(\tilde{X}_i - \tilde{X}_j)^T \Lambda^{-1} (\tilde{X}_i - \tilde{X}_j)\right) + \sigma_\epsilon^2 \delta_{ij}$
      **For** $e = 1 : E$ **:**
          $\theta^* = \arg\max_\theta \log p(\mathbf{y}_e|\tilde{X}, \theta)$ using BFGS or CG method

          $\log p(\mathbf{y}_e|\tilde{X}, \theta) = -\frac{1}{2}\mathbf{y}_e^T K^{-1} \mathbf{y}_e - \frac{1}{2}\log|K| - \frac{D}{2}\log 2\pi$
          $\frac{\partial}{\partial\theta_j}\log(\mathbf{y}_e|\tilde{X}, \theta) = \frac{1}{2}\,\text{tr}\left(\left(K^{-1}\mathbf{y}_e\mathbf{y}_e^T K^{-1} - K^{-1}\right)\frac{\partial K}{\partial\theta_j}\right)$    for $j = 1 : D + F + 2$
      **end**
3     Gradient-based policy search, find $\psi^* = \arg\min_\psi J^\pi(x_t, \psi)$, using BFGS or CG.
      **For** *t=0:T* **:**
          **Step 1** Compute $p(\tilde{x}_t) = p(x_t, u_t), \frac{\partial\mu_t}{\partial\psi}$ and $\frac{\partial\Sigma_t}{\partial\psi}$ with $p(u_t) = p(S(\pi^*(x_t)))$, where $S(\cdot)$ is a
          saturation function, see Section 3-3-2, and $\pi^*(x_t)$ the current optimal policy.
          **Step 2** Compute the next state, using Eq. (3-9):

              $p(x_{t+1}|\tilde{\mu}_t, \tilde{\Sigma}_t) \sim \mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$
              $\mu_{t+1} = x_t + \mathbb{E}[\Delta_{t+1}]$
              $\Sigma_{t+1} = \Sigma_t + \text{cov}\,[\Delta_{t+1}] + \text{cov}\,[\Delta_{t+1}, x_t] + \text{cov}\,[x_t, \Delta_{t+1}]$

          With $\mathbb{E}[\Delta_{t+1}] = \left[\beta_1^T q_1, ..., \beta_E^T q_E\right]^T$ as found in Eq. (3-10)–(3-13), and
          $\text{cov}\,[x_t, \Delta_{t+1}] \subset \text{cov}\,[\tilde{x}_t, \Delta_{t+1}]$ using Eq. (3-28):
          **For** $e = 1 : E$ **:**

              $\beta_e = (K_e + \sigma_{\epsilon,e}^2 I)^{-1}\mathbf{y}$
              $q_e = [q_e^1, ..., q_e^{N_c}]^T$
              $q_e^i = \sigma_{f,e}^2 |\tilde{\Sigma}_t \Lambda_e^{-1} + I|^{\frac{1}{2}} \exp\left(-\frac{1}{2}(\tilde{X}_i - \tilde{\mu}_t)^T (\tilde{\Sigma}_t + \Lambda_e)^{-1} (\tilde{X}_i - \tilde{\mu}_t)\right)$
              $\text{cov}\,[\tilde{x}_t, \Delta_{t+1}^e] = \sum_{i=1}^{n} \beta_{e,i} q_{e,i} (\tilde{\Sigma}_t (\tilde{\Sigma}_t + \Lambda_e)^{-1} (\tilde{X}_i - \tilde{\mu}_t))$
              find $\frac{\partial\mathbb{E}[\Delta_{t+1}^i]}{\partial\Sigma_t}$    and    **For** $d = 1 : D$ **:** $\frac{\partial\,\text{cov}\left[x_t^i, \Delta_{t+1}\right]}{\partial\Sigma_t}$ **end**

          **end**

              Find $\frac{\partial\mathbb{E}[\Delta_{t+1}]}{\partial\mu_t}$    and    $\frac{\partial\,\text{cov}\,[x_t, \Delta_{t+1}]}{\partial\mu_t}$

          And with $\text{cov}\,[\Delta_{t+1}]$ as follows, Eq. (3-14):
          **For** $e = 1 : E$ **:**
              **For** $b = 1 : e$ **:**

                  $Q_{ij} = \frac{\exp\left(n_{ij}^2\right)}{\sqrt{|R|}}$    with $n$ and $R$ as defined in Eq. (3-17) and (3-18)
                  $\text{cov}\,[\Delta_{t+1}]_{eb} = \begin{cases} \beta_e^T Q \beta_b - \mathbb{E}[\Delta_{t+1}^e]\mathbb{E}[\Delta_{t+1}^b], & e \neq b \\ \beta_e^T Q \beta_e - \mathbb{E}[\Delta_{t+1}^e]^2 + \sigma_{f,e}^2 - \text{tr}\left((K_e + \sigma_{\epsilon,e}^2 I)^{-1} Q\right), & e = b \end{cases}$
                  $\text{cov}\,[\Delta_{t+1}]_{eb} = \text{cov}\,[\Delta_{t+1}]_{be}$

              **end**

              find $\frac{\partial\,\text{cov}\left[\Delta_{t+1}^i\right]}{\partial\mu_t}$, using $\frac{\partial\,\text{cov}\,[x_t, \Delta_{t+1}]}{\partial\mu_t}$    and    **For** $d = 1 : D$ **:** $\frac{\partial\,\text{cov}\left[\Delta_{t+1}^i\right]}{\partial\Sigma_t}$ **end**, using $\frac{\partial\,\text{cov}\,[x_t, \Delta_{t+1}]}{\partial\Sigma_t}$

          **end**
          **Step 3** Compute the gradient $dJ^\pi(x_t, \psi)/d\psi$ using Eq. (3-74) – (3-78).
      **end**
4     Update controller parameters: set $\pi^* \leftarrow \pi(\psi^*)$
      Apply $u_t = S(\pi^*(x_t))$ to the system (single trial) and record data $\tilde{X}_{new}$, $\mathbf{y}_{new}$
      Combine old and new data: $\tilde{X} = \begin{bmatrix} \tilde{X} & \tilde{X}_{new} \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} \mathbf{y} & \mathbf{y}_{new} \end{bmatrix}$
  **until** *Task learned*
```

3-5 Discussion on the PILCO algorithm

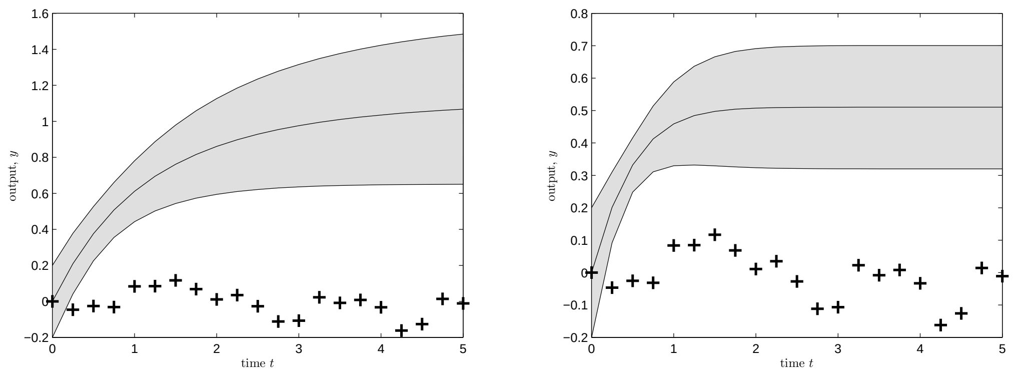
This discussion on the algorithm is divided in three section. The motivation of using the increments as targets will be given before the computational load – a major disadvantage – and its current solution are discussed.

3-5-1 Motivation for using state increments as targets

By learning the difference rather than the state updates itself has a few advantages. Because differences vary less than the absolute values, the underlaying function that describes these differences varies less. Hence learning is easier, which implies that less data is needed to find an accurate model (Deisenroth, 2010).
In practice robotic dynamics are typical relative to the current state and do not depend on the absolute values (Deisenroth et al., 2013b). Predicting outside the training space can lead to poor results. However, by learning difference dynamics the model generalizes better across different parts of the state space, see for an example Figure 3-2. In this example of the first order system $G = \frac{1}{(1+s)}$ the black crosses are the input-output data disturbed with iid Gaussian noise $\epsilon_t \sim \mathcal{N}(0, 0.1)$. In order to test the model outside the training data a constant input of one is used. The expected response is a steady state output of one, the step response. A better approximation of the step response is found by using the state increments as targets. Note, that the model that learns in-output dynamics directly, has a steady state of 0.5. Mathematically learning the differences means that the prior mean function is given as

$$m(\tilde{x}_t) = x_t. \tag{3-79}$$

This can be derived from Eq. (3-9b), the prior mean function of the prediction, Δ_{t+1}, is $m(\tilde{x}_t) \equiv 0$. So $\mathbb{E}[\Delta_{t+1}] = 0$ when test inputs are leaving the training set. Hence, the prior mean function for the increments Eq. (3-79) is obtained. This implies that when the predictions leave the trainings set the prediction will not fall back to zero but remain constant.



(a) Prediction when using increment targets for training.

(b) Prediction when using state updates for training.

Figure 3-2: GP modeling: comparison of the simulations of the step-response of the first-order system $G = \frac{1}{(1+s)}$. The black solid line is the mean prediction of the state and the gray area the 95% confidence interval ($\pm 2\sigma$). In both cases the same input-output data (black crosses) disturbed with iid Gaussian noise $\epsilon_t \sim \mathcal{N}(0, 0.1)$ of the 'real' system is used. However, Figure 3-2a uses the increments of the state updates for training and Figure 3-2b state updates itself.

3-5-2 Computational Complexity

Because the algorithm uses a GP model to learn the (sub)optimal policy, only a moderate amount of data is required. Hence, the algorithm is data efficient. The cost of the data efficiency is a high *computational* *load*, which is a major disadvantage of this algorithm. The computational load is depending on the number of training data, which is growing after every iteration. So for every iteration new data is found and the computational load increases. The dominating computational factors are:

- The computational load of training a GP using gradient-based evidence maximization is dominated by the inversion of the matrix K_θ, see Section 2-2-3. This requires $\mathcal{O}(En^3)$ operations for a n size training set and a target dimension of E. The remaining part of the derivative computations require $\mathcal{O}(En^2)$ per hyper-parameter.
- The most demanding task for predicting the posterior distribution from uncertain inputs is computing the $(n \times n)$ Q matrix Eq. (3-16) for the predictive covariance matrix. This requires a $D+F$ dimensional scalar product per entry, $D+F$ is the dimension of the training inputs, see Section 3-2. Furthermore, Q has to be computed for each entry of the $(E \times E)$ predictive covariance matrix Eq. (3-14). Resulting in the dominating factor for making predictions of $\mathcal{O}((D+F)E^2n^2)$ operations. The predictive mean is less demanding with $\mathcal{O}(En)$.

Because $n \gg D \geq E$, the depending variable of the computational load is the length n of the training data. Note, that the training data is growing after each trial. Sparse approximations are implemented in (Deisenroth, 2010; Deisenroth et al., 2013b) to fix the number of data pairs.

3-5-3 Sparse approximations

The sparse approximation is used to limit the computation load of the predictions and derivatives during policy learning, by limiting the number of training data to 300 data pairs. This can be done by selecting a subset of the training data, called *pseudo training data*. In order to prevent over- or underfitting of the sparse GP model, a *full* GP is trained first. The obtained hyper-parameters of the full GP are used as the hyper-parameters of the pseudo training data of the sparse GP. In order to omit problem of sequential data – a lot of similar data samples of the begin and end states give more weight to these areas – the locations of the pseudo training data are optimized after every trail. The least important data pair of the current sparse GP is substituted with a data pair of the full GP. If the performance of the sparse GP improves the data pair is included in the sparse GP, otherwise the original sparse GP is used. Every training data pair of the full GP is evaluated in this manner, see (Deisenroth, 2010, Section 3.9.1). The log marginal likelihood of the sparse implementation is used as performance measure, see for a detailed description of the evaluation (Deisenroth and Rasmussen, 2009, Section 2.4).

3-6 Conclusion

A major disadvantage of model learning RL is the risk of identifying inaccurate models. Which could lead to learning wrong (sub)optimal policies or even to complete failure. By modeling uncertainties of the model and averaging over them, model-bias can be taken into account (Schneider, 1997). Which reduces the risk of catastrophic failure during learning. Therefore, is in the PILCO algorithm a *probabilistic dynamic model* used, this implies that a distribution over all plausible models that fit the observed data is found. These uncertainties are implemented in the long term predictions and decision making, see Section 3-2-1. The averaging is done according to the posterior distribution, hence over all plausible dynamic models. A GP model is used because: (1) GP models are capable of modeling a wide spread of nonlinear systems including their uncertainties, and (2) GP modeling is based on well understood approximation principles.
The aim of this thesis is to investigates the applicability of the algorithm for large systems and systems with time varying measurement noise. The first step in reaching that goal is investigating the convergence of the RBF controller and GP controller, which will be the discussed of the next chapter.

Chapter 4

Controller choice

The Probabilistic Inference for Learning COntrol (PILCO) algorithm, as presented in the previous chapter, is mainly used with the Radial Basis Function (RBF) controllers or Gaussian Process (GP) controller (Deisenroth and Rasmussen, 2011; Deisenroth et al., 2013a,b). Although the two controllers mathematically describe the same function, no performance comparison it terms of convergence can be found in literature. The GP controller is proposed in (Deisenroth, 2010), because it could benefit from smoothing effect, which are not present in the RBF controller. Furthermore, is the implementations relatively easy because a GP is already used for prediction. This chapter compares the convergence performance of the GP controller and the RBF controller. This chapter concludes with a motivation – based on the results – to use one of the two controllers.

4-1 RBF controller and GP controller

After fixing $\sigma^2_{f,\pi} = 1$ the GP controller is over-parametrized with one parameter compared with the RBF controller. The dependence on the inverse of kernel matrix $K_\pi + I\sigma^2_{\epsilon,\pi}$ could lead to numerical instabilities. However, by selecting $\sigma^2_{\epsilon,\pi} = \sigma^2_\epsilon = 0.01^2$, the GP is as algebraically expressive RBF controller because, $K + I\sigma^2_\epsilon$ has full rank. Remember, $X_\pi = [x_{\pi,i}, ..., x_{\pi,i}]$. For easy comparison an overview of the two controllers is presented below:

RBF controller

$$\pi(x_t, \psi) = \sum_{i=1}^{n} w_i \phi_i(x_t)$$

$$\phi_i(x_t) = \exp\left(-\frac{1}{2}(x_{\pi,i} - x_t)^T \Lambda^{-1} (x_{\pi,i} - x_t)\right)$$

GP controller

$$\pi(x_t, \psi) = \sum_{i=1}^{n} \beta_{\pi,i} k(x_\pi, x_t)$$

$$k(x_\pi, x_t) = \sigma^2_{f,\pi} \exp\left(-\frac{1}{2}(x_{\pi,i} - x_t)^T \Lambda^{-1} (x_{\pi,i} - x_t)\right)$$

$$\beta_\pi = (K_\pi(X_\pi, X_\pi) + \sigma^2_{\epsilon,\pi} I)^{-1} \mathbf{y}_\pi.$$

The initial controller parameters are selected as

$$w = \mathcal{N}(0, 0.1^2), \qquad X_\pi = \mathcal{N}(\mu_0, \Sigma_0), \qquad \text{and} \qquad \mathbf{y}_\pi = \mathcal{N}(0, 0.1^2). \tag{4-1}$$

The hyper-parameter vector of the RBF controller is $\psi = \{w,\ \ell_1, ..., \ell_D,\ X_\pi\} \in \mathbb{R}^{D+n(D+1)}$, for the GP controller the hyper-parameter vector is $\psi = \{\mathbf{y}_\pi,\ \ell_1, ..., \ell_D,\ X_\pi\ \sigma^2_{\epsilon,\pi}\} \in \mathbb{R}^{D+1+n(D+1)}$. For characteristic length-scales that correspond with original states are chosen to be 1. For the characteristic length-scales that correspond with the augmented states (see Appendix A-1) are set to 0.7. The random control input for the initial trial $u_t \sim \mathcal{U}(-u_{max}, u_{max})$. System states and control input can be controlled at $10Hz$.

4-2 Simulations set-up

To evaluate the algorithm two scenario's are selected: (1) a second order system and (2) the cart-pole system. For both systems the width of the cost function is set to $\sigma_c = \frac{1}{4}$.

4-2-1 1: The second order system

The 'real' dynamics of the second order system are chosen as

$$G(s) = \frac{1}{s^2 + s + 1}. \tag{4-2}$$

For the state vector $\begin{bmatrix} x \\ \dot{x} \end{bmatrix}$, the state space description becomes

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \tag{4-3}$$

where the initial state is drawn from the normal distribution

$$\begin{bmatrix} x_0 \\ \dot{x}_0 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1^2 & 0 \\ 0 & 0.1^2 \end{bmatrix} \right). \tag{4-4}$$

The states for GP training are defined as follows:

$$\mathbf{x}_{training\ input} = \begin{bmatrix} x \\ \dot{x} \\ u \end{bmatrix}, \qquad \text{and} \qquad \mathbf{y}_{training\ targets} = \begin{bmatrix} \Delta_x \\ \Delta_{\dot{x}} \end{bmatrix}. \tag{4-5}$$

For easy understanding of the results the goal is chosen as a steady state solution at one. Hence, the goal state is defined as

$$\mathbf{x}_{target} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{4-6}$$

4-2-2 2: The cart-pole system

The task of the second case is to learn the swing-up and stabilization of a pendulum in the cart-pole set-up, a benchmark problem in Reinforcement Learning (RL). This cart-pole system is used throughout the thesis. The unknown dynamics are described by the following equations of motion:

$$(m_1 + m_2)\ddot{x}_1 + \frac{1}{2} m_2 l \ddot{\varphi} \cos\varphi - \frac{1}{2} m_2 l \dot{\varphi}^2 \sin\varphi = u - b\dot{x}_1, \tag{4-7}$$

$$2l\ddot{\varphi} + 3\ddot{x}_1 \cos\varphi + 3g \sin\varphi = 0, \tag{4-8}$$

where x is the position on the cart and φ the angle of the pendulum measured anti-clockwise from hanging down. The gravitation acceleration is given as $g = 9.82\ m/s^2$, the constants $m_1 = 0.5\ kg$ and $m_2 = 0.5\ kg$ are the masses of the cart and the pendulum respectively, $l = 0.5\ m$ is the length of the pendulum and $b = 0.1$ $N/m/s$ the friction between the cart and the rails, see Figure 4-1. Note, that when using the PILCO algorithm the dynamics are assumed to be unknown.

The state used to simulate the dynamics of the 'real' system is

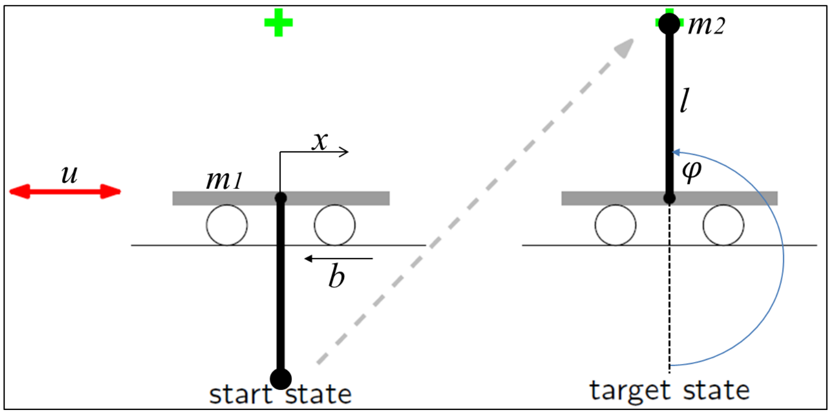


Figure 4-1: Schematic overview of the cart-pole system, edited from (Deisenroth and Rasmussen, 2009). In the left figure the initial state is given, where the green cross is the target state. In the right figure the goal is presented, stabilizing the pendulum upright in the center of the track, after swing-up.

$$\mathbf{x} = \begin{bmatrix} x \\ \dot{x} \\ \dot{\varphi} \\ \varphi \end{bmatrix}. \tag{4-9}$$

Because, there is a angle in the state description this angle is given by its complex parts, $\sin(\varphi)$ and $\cos(\varphi)$. This ease the GP training problem because of some linearizations. The computations for the augmented state given in Appendix A-1. The augmented and full states are:

$$\mathbf{x}_{augm} = \begin{bmatrix} \sin(\varphi) \\ \cos(\varphi) \end{bmatrix}, \qquad \text{and} \qquad \mathbf{x}_{full} = \begin{bmatrix} x \\ \dot{x} \\ \dot{\varphi} \\ \varphi \\ \sin(\varphi) \\ \cos(\varphi) \end{bmatrix}. \tag{4-10}$$

However, instead of using the full state to learn the dynamics, the angle is omitted from the full state for the training input. The corresponding training targets, which are the increments, are represent with 'original' state which is also used for simulating the dynamics. Augmenting the training inputs with the control input results in the training data states:

$$\mathbf{x}_{training\ input} = \begin{bmatrix} x \\ \dot{x} \\ \dot{\varphi} \\ \sin(\varphi) \\ \cos(\varphi) \\ u \end{bmatrix}, \qquad \text{and} \qquad \mathbf{y}_{training\ targets} = \begin{bmatrix} \Delta_x \\ \Delta_{\dot{x}} \\ \Delta_{\dot{\varphi}} \\ \Delta_{\varphi} \end{bmatrix}. \tag{4-11}$$

Note, that the input state for the one-step update predictions has to be the same as the training input state. The cost function uses the full state representation. The target state, the inverted position of the pendulum in the middle of the track, is given as

$$\mathbf{x}_{target} = \begin{bmatrix} 0 \\ * \\ * \\ * \\ 0 \\ -1 \end{bmatrix}. \tag{4-12}$$

Note, that this target state correspond to $\mathbf{x}_{target} = \begin{bmatrix} 0 & * & * & \pi + 2k\pi \end{bmatrix}^T$ in the original state representation. See Section 3-3-3 for detailed explanation of the cost function.

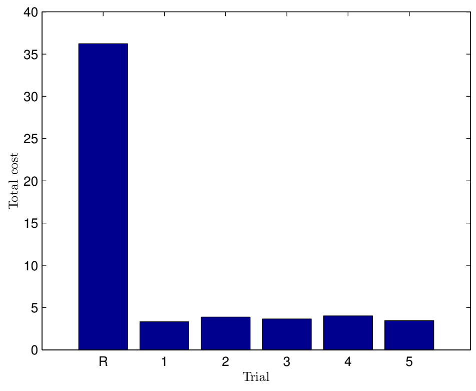
4-3 Simulation results

For the second order system both controllers are initialized with five basis functions. The trial length is four seconds and one initial trial is made with a random input $u_t \sim \mathcal{U}(-u_{max}, u_{max})$. Both controller give similar results, which are presented in Figure 4-2. The left side is the RBF controller and the right the GP controller. The controller corresponding to the trial with the lowest total cost is considered as the best solution. This is 3.43 and 3.28 for the RBF controller and GP controller respectively. The GP controller is more aggressive as it has large differences in two consecutive control inputs. Furthermore, a larger range of the control input domain ($[-10\ 10]$) is used. From the results the smoother RBF controller is the more feasible solution, and therefor better to implement in a real system. On the other hand the flexibility and aggressive behavior of the GP controller can result in better policies for complex tasks.
For the simulation of the cart-pole system the trial lengths are set to 2.5 seconds an using two trials with a random control input to initialize the model $u_t \sim \mathcal{U}(-u_{max}, u_{max})$. Ten basis functions are used for both controllers. In Figure 4-3 is presented that both controller are able to solve the task. The trajectories of the states are shown, the position (solid black), cart velocity (blue dotted) and angular velocity of the pendulum (green dot dashed) should all go to zero. The angle of the pendulum (red striped) should go to $\pi + 2k\pi$, for k as an integer. The best trial of the GP controller is trial eight and has a cost of 6.65, which is similar to the best trial of the RBF controller 6.52.
To increase the convergence difficulty of the learning process the basis functions of the RBF and GP controller are increased to fifty, see Figure 4-4. Although, the RBF controller is not able to find a solution, the GP controller is able to stabilize the pendulum. However, this suboptimal solution swings the pendulum one complete circle before stabilization, see Figure 4-4d. Hence, the algorithm got stuck in a local minimum.
Although, sparse implementations exists to limit the computational complexity, the computational load is high. Chapter 5 will address this issue. A key assumption of the PILCO algorithm is that the uncertainties are Gaussian distributed. However, measurement noise could have many different distributions. An interesting question is to test the generality of the algorithm by testing the performance of the algorithm subjected to different measurement noise distributions. This study is presented in Chapter 6. The RBF controller is nonlinear flexible controller which is able to learn complex tasks. The suggestion to use a GP controller is mainly driven by the ease of implementation. However, the performance of the GP controller is not verified or compared with the RBF controller. The next chapter presents a case study on the performance of the GP controller.

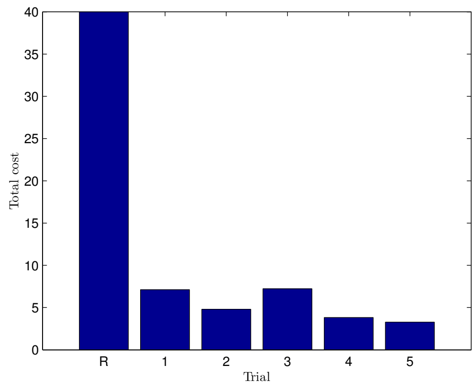
4-4 Conclusion

The GP controller has one parameter more the initialize, $\sigma_{\epsilon,\pi}$, which is set to the measurement noise standard deviation. This parameter should be chosen unequal to zero. Firstly, to prevent that the matrix $K + I\sigma_{\epsilon,\pi}$ becomes singular. Secondly, it causes the matrix to be full rank which makes it as algebraically expressive as the RBF controller.
The algorithm is able to find an (sub)optimal controller automatically. However, the choice of controller and controller structure are of great importance. The number of basis functions in the RBF and GP controllers can be a deciding factor for failure or success. Both controllers find good solutions for the tasks provided, if the number of basis function is chosen correctly. Too few, could result in a controller which is not able to solve the problem. Too many, could cause the optimization problem to end up in a local minimum.

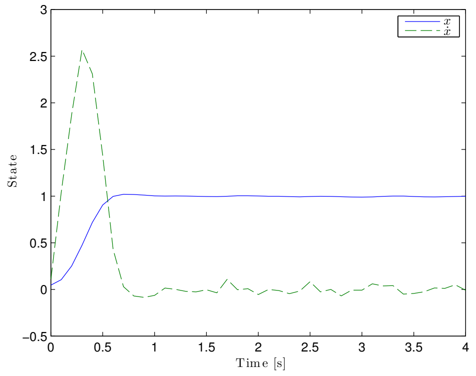
By increasing the number of basis functions of the controller, the applicability of the controller for larger and more complex systems and/or tasks is investigated. A convergence difference between the two controllers is found when the number of basis functions is increased to fifty. The algorithm seems to profit from the indirect chosen weighting factors of the GP controller. The small uncertainty factor $\sigma_{\epsilon,\pi}$ smooths the function, which ease the optimization. Although, both controllers were not able to find the global optimal policy, the algorithm with the GP controller converges slightly easier than with the RBF controller. Because of the higher convergence rate, the GP controller is better suited for the algorithm and is used in the remainder of this thesis.



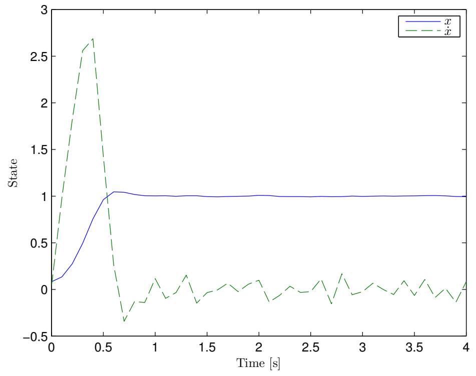
(a) Total cost per trial, RBF controller



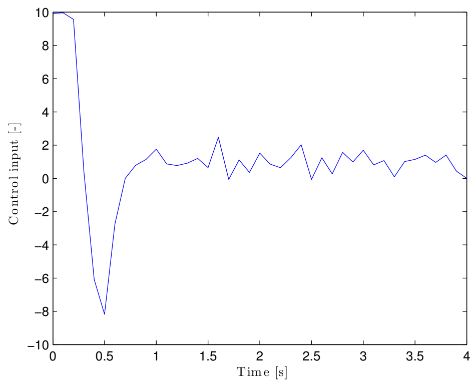
(b) Total cost per trial, GP controller



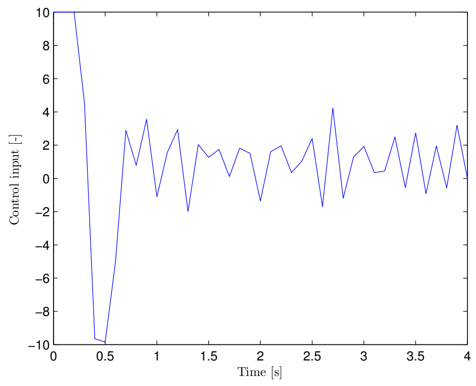
(c) State trajectories of the best trial, RBF controller



(d) State trajectories of the best trial, GP controller

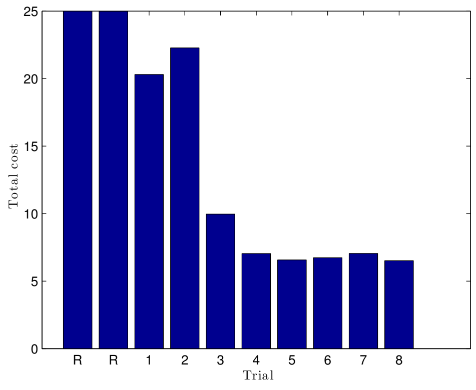


(e) Control input of the best trial, RBF controller

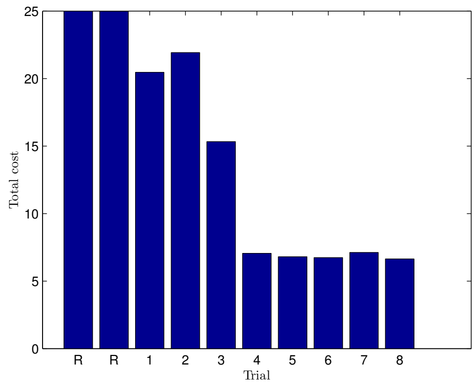


(f) Control input of the best trial, GP controller

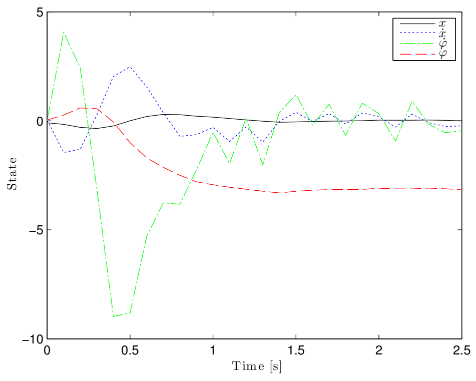
Figure 4-2: Controller choice 2nd order system: comparison of the GP and RBF controller controller using 5 basis functions. The left figures are the results of the RBF controller and the right figures of the GP controller. The top figures give the total cost per trial. Note, that the first bar is the initial trial with the random controller. The middle two figures present the state trajectories of the best trials, i.e total lowest cost, which is trial one for the RBF controller and trial five for the GP controller. The blue solid line is the state x and the striped green line the state $\dot{x}$. The corresponding control input of the best trial is given in the bottom two figures.



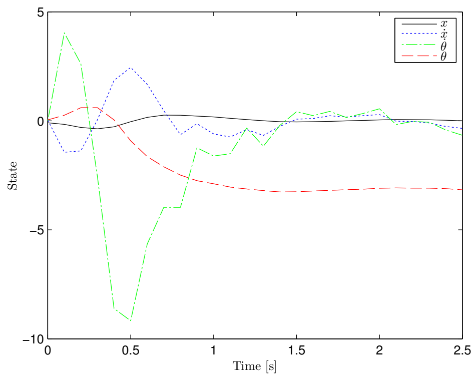
(a) Total cost per trial, RBF controller



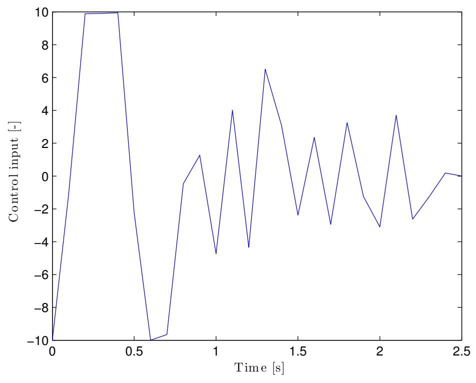
(b) Total cost per trial, GP controller



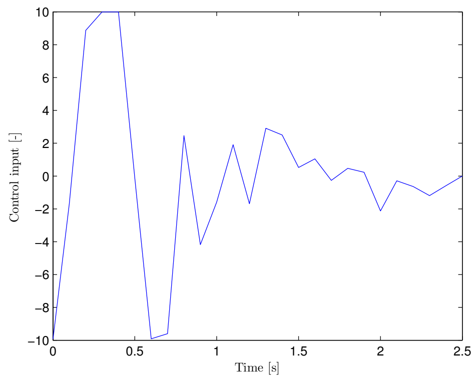
(c) State trajectories of the best trial, RBF controller



(d) State trajectories of the best trial, GP controller

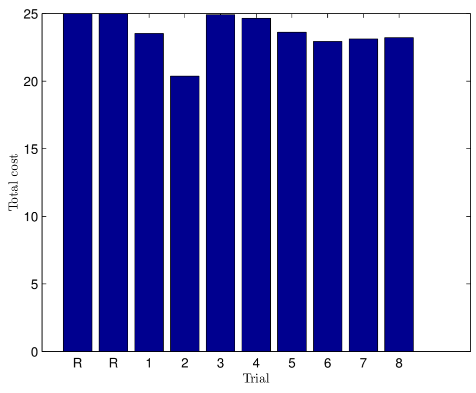


(e) Control input of the best trial, RBF controller

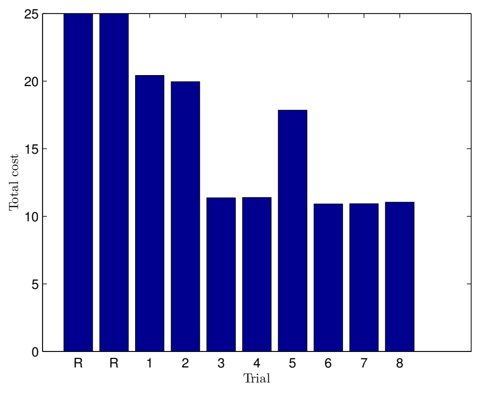


(f) Control input of the best trial, GP controller

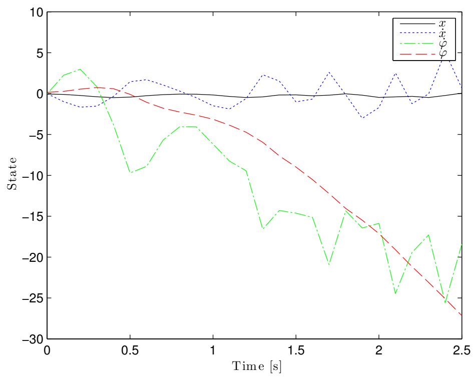
Figure 4-3: Controller choice cart-pole system: comparison of the GP and RBF controller using 10 basis functions. The left figures are the results of the RBF controller and the right figures of the GP controller. The top figures give the total cost per trial. Note, that the first bar is the initial trial with the random controller. The middle two figures present the state trajectories of the best trials, i.e. total lowest cost, which is trial eight for both controllers. The black solid line is the position of the cart (x). The blue dotted line is the carts velocity $(\dot{x})$, the dot-stripped red line is the angle of the pendulum (φ) and the green striped line is the angular velocity of the pendulum $(\dot{\varphi})$. The bottom two figures give the control input corresponding to the best trial of the middle figures.



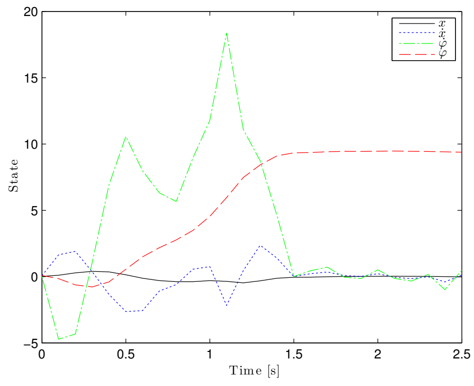
(a) Total cost per trial, RBF controller



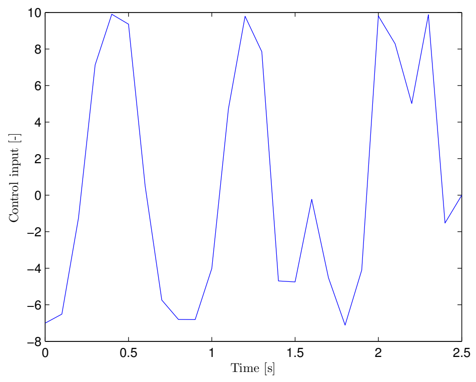
(b) Total cost per trial, GP controller



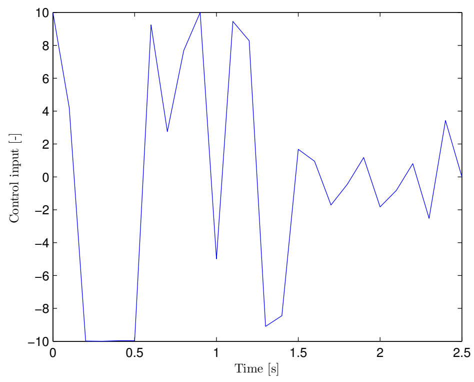
(c) State trajectories of the best trial, RBF controller



(d) State trajectories of the best trial, GP controller



(e) Control input of the best trial, RBF controller



(f) Control input of the best trial, GP controller

Figure 4-4: Controller choice cart-pole system: comparison of the GP and RBF controller using 50 basis functions. The left figures are the results of the RBF controller and the right figures of the GP controller. The top figures give the total cost per trial. Note, that the first bar is the initial trial with the random controller. The middle two figures present the state trajectories of the best trials, i.e. total lowest cost, which are trial two for the RBF controller and six for the GP controller. The black solid line is the position of the cart (x). The blue dotted line is the carts velocity ($\dot{x}$), the dot-stripped red line is the angle of the pendulum (φ) and the green striped line is the angular velocity of the pendulum ($\dot{\varphi}$). The bottom two figures give the control input corresponding to the best trial of the middle figures.

Chapter 5

Reducing the computational time

The Probabilistic Inference for Learning COntrol (PILCO) algorithm using the Gaussian Process (GP) controller – as found in the previous chapter – is a promising algorithm because: (1) the GP model is capable of modeling a wide spread of nonlinear systems including their uncertainties, and (2) the GP controller shows good convergence properties. These good convergence properties are necessary for implementation of the algorithm on large systems. Although, a fast converging controller is found the computational load of the algorithm is high. Between the trials the algorithm requires approximately 10 minutes on an average computer to find the (sub)optimal controller for the cart-pole task, (Deisenroth and Rasmussen, 2009). This computational load is a major disadvantage, especially for large systems. Before presenting the proposed parallelized implementation, this chapter starts with a review of the most demanding computations and how they are implemented in the toolbox of (Deisenroth et al., 2013b). The algorithm is divided in two parts to analyze the MATLAB implementation: (1) GP system identification and (2) model-based RL. Throughout this chapter the cart-pole system used. Conclusions on the results to the extend of large systems will finalize the chapter.

5-1 Part 1: GP system identification

For the training of the GP the `gpml` toolbox of Rasmussen and Williams (2006) is used. The training of the GP during the eight iterations of 2.5 seconds took only approximately 1.0 percent of the total computing time. This seems unnecessary, but for more complex systems larger data sets are necessary. For more than 300 data pairs ($n = 300$) the sparse GP model is used, see Section 3-5-3. Although, training of the GP evaluates all data pairs, the computations of the policy learning step are limited to 300 data pairs. For example, the GP training time for the cart-pole system with 1000 data pairs and is approximately 33 percent of the total time. Hence, reducing the computational time for GP training is rewarding for highly complex systems. For convenience the equations for GP training (Section 2-2-3) are repeated here:

$$\hat{\theta} \in \arg\max_{\theta} \log p(\mathbf{y}|\tilde{X}, \theta), \tag{5-1}$$

$$\log p(\mathbf{y}|\tilde{X}, \theta) = -\frac{1}{2}\mathbf{y}^T K_\theta^{-1}\mathbf{y} - \frac{1}{2}\log|K_\theta| - \frac{D}{2}\log 2\pi. \tag{5-2}$$

The gradient of the marginal likelihood with respect to the hyper-parameters is

$$\frac{\partial}{\partial\theta_j}\log(y|\mathbf{x}, \theta) = \frac{1}{2}\operatorname{tr}\left(\left(K_\theta^{-1}\mathbf{y}\mathbf{y}^T K_\theta^{-1} - K_\theta^{-1}\right)\frac{\partial K_\theta}{\partial\theta_j}\right). \tag{5-3}$$

The dominating computation for the finding the (sub)optimal hyper-parameters is the inverse of the covariance matrix $K_{\theta,ij} = K(X_i, X_j)+\sigma_\epsilon^2 I$. For the target dimension E finding the inverse of a positive definite symmetric (n by n) matrix using Gaussian elimination require $\mathcal{O}(En^3)$ operations, the overhead of the calculation of the derivatives is $\mathcal{O}(En^2)$ per hyper-parameter. The `gpml` toolbox uses an c implementation of the Cholesky inverse. The Cholesky factorization of a positive definite matrix is given as

$$A = LL^T = R^T R, \tag{5-4}$$

where L is a lower triangular matrix and R an upper triangular matrix, and $L^T = R$ holds. An inverse of the positive definite matrix A can be found as

$$A^{-1} = R^{-1}L^{-1} = L^{-T}L^{-1}. \tag{5-5}$$

This can be implemented as solving the system $Ax = b$. Take b equal to the identity matrix of appropriate dimensions,

$$A^{-1} = L^T\backslash(L\backslash I). \tag{5-6}$$

5-1-1 Parallel implementation of the GP system identification

Because training of the GP means training E *independent* Gaussian Processes, the architecture of the training is ideal for parallel computing, see for the simplified implementation Appendix B-1. As Amahdahl's law suggest is it assumed that the computation time, T_{comp}, will consist of a constant non parallelizable factor (minor computations and initializations), α and a parallelizable part. The parallelizable part is dominated by a matrix inverse. Hence, a simplified approximation of the computational time is given as τn^3, where n (n is number of data samples) and time constant τ is time required per data sample for the target dimensions E. Note, that the target dimension is integrated in the time constant τ,

$$T_{comp} = \alpha + \tau n^3. \tag{5-7}$$

By fitting on data, the constant α is set to one and τ to $5.2 \cdot 10^{-8}$. Because, the predictive distribution has a four dimensions in the cart-pole setup, the maximum useful cores is four. The idealistic assumption is that the parallelizing has a linear effect on the second part of the computation time equation Eq. (5-7). Neglecting the additional overhead time introduced by the parallel implementation and with N the number of cores the prediction of the computational time becomes

$$T_{comp} = \alpha + \frac{\tau n^3}{N}. \tag{5-8}$$

The fitted approximation and the prediction of the time decrease when using two and four cores is presented in Figure 5-1, the measured data (yellow solid with 95 percent confidence interval) of the original algorithm is fitted with the solid black approximation. The assumed ideal linear time reductions are presented in dot dashed blue for two cores and dotted red for four cores. In this ideal scenario a clear decrease of the computational time is predicted, when using multiple cores. The expectation is that the real time decrease is less severe.
In Figure 5-2a the results can be found of twenty training runs for ten to thousand training samples on a desktop computer with a Intel(R) Core(TM) i7 CPU 860 @2.80GHz. The black solid line is the original algorithm (labeled synchronous) using all computational power available and normal MATLAB commands. The green dotted line is the parallel implementation however, only one core is available. Hence, the computations are done synchronously. The time increase is caused by the overhead time of parallel computing and the lack of parallel options in the build in MATLAB commands. Although less then predicted, a significant time decrease is found for the parallel implementation using two (blue dotted) or four (red dotted) cores.
In Figure 5-2b the percentage of time decrease can be found for the two (blue dot solid) and four (red dotted) core parallel implementation. A optimal value for 100 data pairs is found of 53 percent for the two core and 68 percent for the four core implementations. Note, that the target dimension is four in the cart-pole system. So, using more than four cores will not further decrease the computational time.

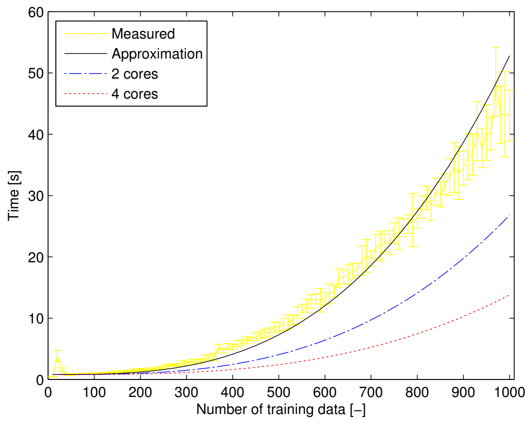
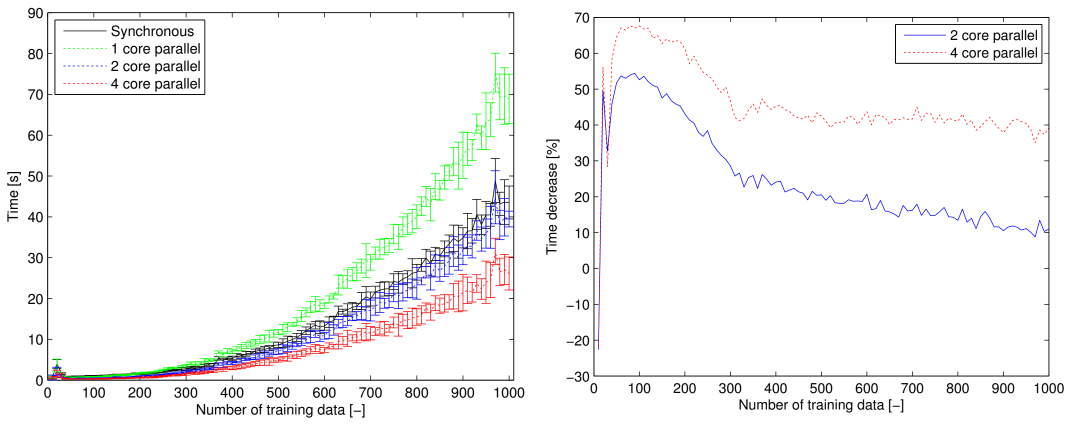


Figure 5-1: Computational time reduction: predicted time decrease for parallel GP training. The time duration of the original algorithm is measured for 20 GP learning optimizations sessions, and presented (solid yellow line) with the mean and 95% confidence interval ($\pm 2\sigma$). This data is fitted with the approximation (solid black line) of Eq. (5-8). The linear (ideal) time decrease when using two or four cores are given in the blue dot dashed and red dotted lines respectively.



(a) Time duration of parallel and synchronous GP training.

(b) Time reduction of the parallel training for two and 4 cores.

Figure 5-2: Computational time reduction: realized time decrease for parallel GP training. Value's are presented by their mean and 95% confidence intervals ($\pm 2\sigma$), required with 20 GP training sessions, using a Intel(R) Core(TM) i7 CPU 860 @2.80GHz CPU. In Figure 5-2a the time duration of the original (synchronous) training (black solid line) is compared with the one (green dotted line), two (blue dotted line) and four (red dotted line) core parallel implementation. In Figure 5-2b the time reduction in percentage of the sychronous training is given for the one (green solid), two (blue dot dashed) and four (red dotted) core parellel training.

For more than 100 data pairs the advantage of the parallel implementation degrades. For 1000 data pairs the reduction of the computational time decreases to 11 and 39 percent for the two and four core implementations respectively. The advantage degrades because the overhead time increases with the number of data pairs. Note, it needs to store bigger matrices. The slope of the four core implementation is flatter than the slope of the two core implementation, because the four core implementation breaks the matrices is smaller pieces. Hence, the overhead time has a smaller negative influence. For around 100 data pairs the highest decrease in percentage is found. However, for large amount of data pairs the absolute time reduce is the highest.

5-2 Part 2: Model-based Reinforcement Learning

Learning the control policy for the cart-pole set-up in 8 iterations with 2.5 seconds trials requires ±96.5 percent of the total computing time [1]. The most demanding computation during learning of the policy is computing the predictive distribution with, ± 53 percent of the total learning time. Computing the control input u takes ±30 percent (GP controller). Note, this is determined by the controller choice, the Radial Basis Function (RBF) controller or linear controller will be computational less demanding. The most demanding computations are the predictive distribution and the derivatives for prediction. Note, that the control policy is a deterministic GP model as well. Hence, the most demanding computations are the same, so the solution will be applicable to the GP controller as well. For finding the predictive distribution of the model the most demanding computation is with, ±37 percent of the prediction computation time, computing the so called Q matrix. Which is necessary to compute the predicting covariance matrix, see Section 3-2-1. The second most demanding computation involves finding the derivative of the input output covariance with respect to the covariance matrix, which takes ±14 percent of the prediction time. For convenience the equations for finding the Q matrix are given bellow

$$\text{cov}\left[\Delta^a_{t+1}, \Delta^b_{t+1}\right] = \begin{cases} \beta_a^T Q \beta_b - \mathbb{E}[\Delta^a_{t+1}]\mathbb{E}[\Delta^b_{t+1}], & a \neq b \\ \beta_a^T Q \beta_a - \mathbb{E}[\Delta^a_{t+1}]^2 + \sigma^2_{f,a} - \text{tr}\left((K_a + \sigma^2_{\epsilon,a} I)^{-1} Q\right), & a = b \end{cases}, \tag{5-9}$$

where Q is

$$Q_{ij} = \frac{K_a(\tilde{X}_i, \tilde{\mu}_t) K_b(\tilde{X}_j, \tilde{\mu}_t)}{\sqrt{|R|}} \exp\left(\frac{1}{2} z_{ij}^T R^{-1} \tilde{\Sigma}_t z_{ij}\right). \tag{5-10}$$

Which can be rewritten for a numerical stable implementation as

$$Q_{ij} = \frac{\exp\left(n^2_{ij}\right)}{\sqrt{|R|}}, \tag{5-11}$$

$$n^2_{ij} = 2(\log(\sigma_{f,a}) + \log(\sigma_{f,b})) - \frac{\zeta_i^T \Lambda_a^{-1} \zeta_i + \zeta_j^T \Lambda_b^{-1} \zeta_j - z_{ij}^T R^{-1} z_{ij}}{2}, \tag{5-12}$$

where the following is defined $R = \tilde{\Sigma}_t(\Lambda_a^{-1} + \Lambda_b^{-1}) + I$, $\zeta_i = \tilde{X}_i - \tilde{\mu}_t$, and $z_{ij} = \Lambda_a^{-1}\zeta_i + \Lambda_b^{-1}\zeta_j$.

5-2-1 Parallel implementation of the model-based RL

The gradient based optimization involves one optimization. Because, the prediction of a GP can be computational demanding and involves computing E *independent* Gaussian means, covariances and the cross-covariances of the inputs and the prediction, is it possible to parallelize these. However, parallelizing a loop results in extra overhead. This is the results of sending the data to the different processors and adding the results back together. Hence, to have advantage of a parallel loop the computations in the loop have to be demanding enough to justify this extra computing time. The simulation to learn the swing up and balance of the pendulum for the cart-pole system, was performed with trials of 2.5 seconds and eight iterations of the learning process. A total of 56275 calls were made to the computations for the predictive distribution and the derivatives which took a total of 984.983 seconds to compute. Clearly these computations, with an average of 0.0175 seconds, are not demanding enough to find a successful parallel loop in MATLAB. Even if the training data is taken for 60 seconds (600 data pairs) the computational time for computing the predictive distribution and the derivatives takes 0.16 seconds. Furthermore, the number of loop iterations – the prediction dimension E – is small.
In the prediction step the mean and the covariance prediction (both including derivatives and some extra

[1]The total computing time for Policy learning and the training of the GP is not 100 percent because of other requirements e.g., simulations of the 'real' dynamics.

computations) are separated in two for loops, see Algorithm (3.1). Both loops are parallelized separately to investigated the influence of both loops. In the simulations the number of function evaluations and line searches are limited too one. In this way a fair comparison can be made. The expectation that the computations are slowed down by the parallelization are met, see Figure 5-3. The computation time of the parallelized and non parallelized learning of the policy is presented. The parallel function for the predictive mean is presented with the label 'par $\mathbb{E}[\Delta_{t+1}]$' (green dot dashed line), the covariance is labeled as 'par cov $[\Delta_{t+1}]$' (red dotted line). In order to find the speed increase or decrease of these separate loops, the parallel implementation of these loops is done for one at a time.

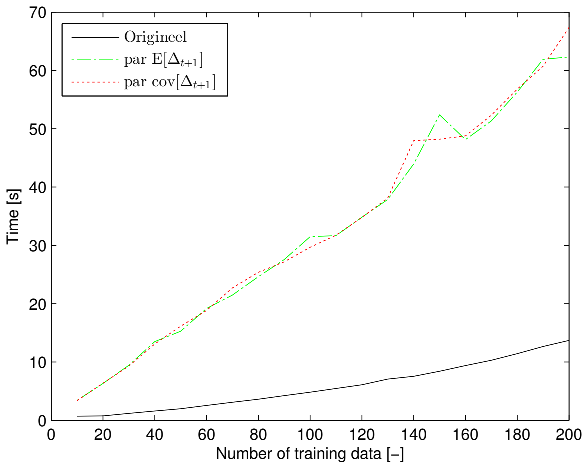


Figure 5-3: Computational time reduction: realized time increase for parallel policy learning. The figure presents the time duration of one line search of the policy learning optimization. The original policy learning is presented in solid black, the green dot dashed line is the parallel mean prediction and the red dotted line is the parallel covariance prediction.

5-3 Conclusion

The parallelization of the gradient based learning is impractical. Only a single optimization is required and the internal computations are not demanding enough to overcome the extra overhead created by separating and combining the matrices, resulting in an increasing of the computational time when parallelized. The parallelization of the GP training reduced the computational time for more than 100 data pairs. From 300 data pairs onwards a reductions in computational time for two and four cores is around 20 and 40 percent respectively. Hence, for the cart-pole system with 1000 data pairs a 13 percent reduction of the total computation time can be found. For systems with a higher prediction dimension E the computational time reduction will be higher. Hence, this parallel implementation increases the applicability of the PILCO algorithm for large systems.

Chapter 6

Robustness analyses

The work of the two previous chapters increased the applicability of the Probabilistic Inference for Learning COntrol (PILCO) algorithm for large systems, the first part of the main research objective. This chapter presents the investigation of the influence of different measurement noise characteristics on the performance. This work is conducted to gain insight in the robustness of the algorithm. The aim is to use this knowledge to make the PILCO algorithm applicable for systems with time varying measurement noise, the second part of the main research objective.
In this chapter all distributions used to identify the influence of the moments on the performance, are zero mean and can approximate the Gaussian distribution. Most of the distributions are chosen such that a single parameter changes one certain property. Before the influence of the measurement noise variance, skewness and kurtosis are presented, the simulation experiments are explained in detail. Subsequently, the influence of increasing 'extreme' – large absolute – values, while keeping a constant variance is presented. This chapter concludes with a discussion of the results.

6-1 Influence of different measurement noise characteristics on the performance

To identify the influence of: (1) variance, (2) skewness, (3) kurtosis, and (4) extreme values the impact of the measurement noise on the system has to be identified. A nonlinear real system with can be written as:

$$x_{t+1} = f(x_t, u_t), \tag{6-1}$$

$$y_t = x_t + \epsilon_t, \tag{6-2}$$

where the true system dynamics of Eq. (6-1) are unknown and the measurement noise is assumed to be zero mean Gaussian independent and identically distributed (iid). For $x \in \mathbb{R}^D$ the measurement noise distribution is

$$\epsilon_t \sim \mathcal{N}\left(\begin{bmatrix}0\\ \vdots \\ 0\end{bmatrix}, \begin{bmatrix}\sigma^2_{\epsilon,1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma^2_{\epsilon,D}\end{bmatrix}\right). \tag{6-3}$$

The system interaction – where the observations are disturbed with measurement noise – is shown in Figure 6-1. Note, that the the states are partly observable, resulting in a Partially Observable Markov Decision Process

(POMDP). Furthermore, the disturbance is fed to the system via the controller. In order to investigate the robustness of the algorithm for measurement noise the cart-pole system of Section 4-2-2 is used. This is a well known benchmark problem for Reinforcement Learning (RL) algorithms. The controller is learned using trails of four seconds and ten learning iterations. The measurement noise is independent and identically distributed Gaussian distributed per state, $\mathcal{N}(0, 0.01^2)$. The influence of the different measurement noise moments on the performance of the learned controller is investigated by applying different noise distributions on the system. Note, that the system itself does not change, only the measurement noise ϵ_t. Hence, a change in environment in this chapter means a change in the probability distribution from which the measurement noise is sampled. The results are given for 100 trials of four seconds per parameter change. In order to compare the performance a reference of again 100 trials of the original system is used. So, the system was perturbed with a independent identically distributed zero mean Gaussian measurement noise of $\mathcal{N}(0, 0.01^2)$ per state. The mean of the total cost – used as performance measure in this chapter – of these hundred trials is 6.92.

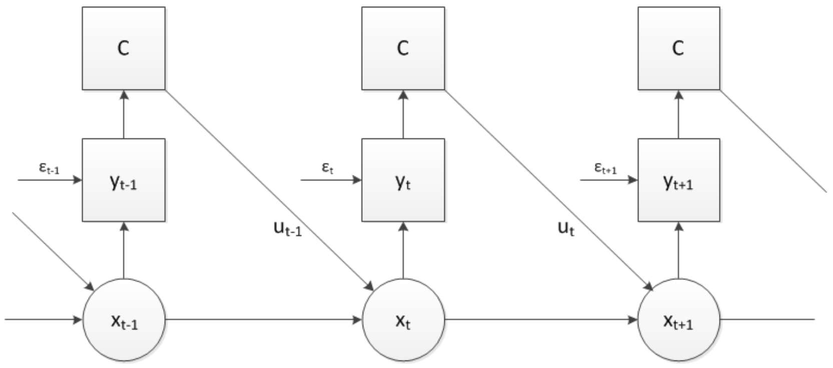
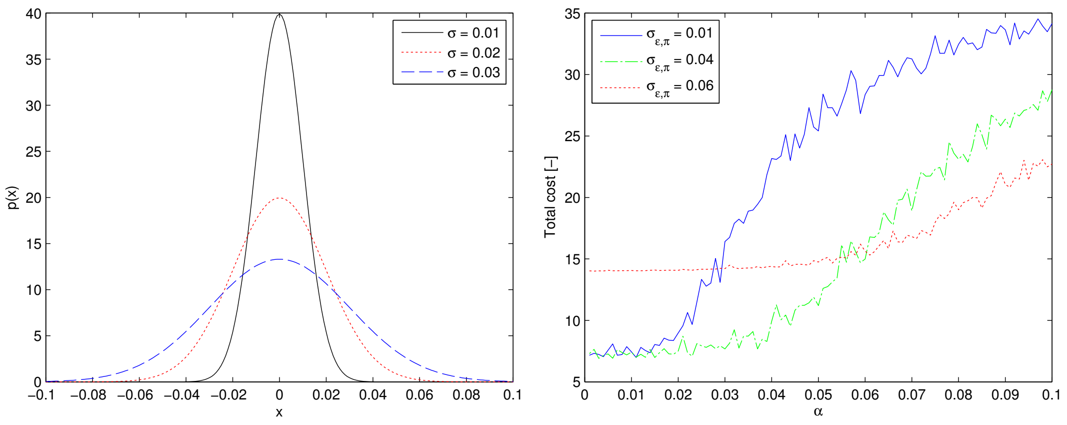


Figure 6-1: Schematics of the Partially Observable Markov Decision Process (POMDP). The observations (y_t) of the states (x_t) are disturbed with measurement noise ϵ_t. Hence, the observations and not the real states are available for control. The measurement noise is passed on via the control input u_t by the controller C to the next state (x_{t+1}).



(a) Gaussian probability distribution for different standard deviation.

(b) Total cost for increasing standard deviation of the zero mean Gaussian measurement noise.

Figure 6-2: Robustness analyzes: the influence of measurement noise variance on the cost. In Figure 6-2a the zero mean Gaussian probability distribution is given for different standard deviations (σ). In Figure 6-2b the performance of three controllers is presented. The controllers are learned for different Gaussian measurement noise: blue solid line for $\sigma_{\epsilon,\pi} = 0.01$, the green dot dashed line for $\sigma_{\epsilon,\pi} = 0.04$ and the dotted red line for $\sigma_{\epsilon,\pi} = 0.06$.

6-1-1 1: Measurement noise variance

Using the normal distribution the influence of the variance can be tested. By changing the variance, the mean, skewness, and kurtosis are kept equal to zero. In Figure 6-2a the influence on the probability distribution is shown. By increasing the noise variance, larger absolute values are chosen with a higher probability. Hence, worse performance is expected.

In Figure 6-2b the performance of three learned controllers are shown as the mean of the total cost of 100 trials. The blue line the is the learned controller for $\sigma_{\epsilon,\pi} = 0.01$, the green dot dashed line is the controller for $\sigma_{\epsilon,\pi} = 0.04$ and the dotted red line for the $\sigma_{\epsilon,\pi} = 0.06$ case. All controllers are tested with 100 trials per case for the cart-pole system with a zero mean iid Gaussian measurement noise with a variance from 0.001^2 to 0.1^2.

The expectation is that the controller will be optimal for there own environment and will be less optimal for other environments. The general perception is that a controller learned for an environment with large noise components will be conservative in an environment with small noise compared with a controller developed for that specific case.

Therefor, is it remarkable that the $\sigma_{\epsilon,\pi} = 0.04$ controller has a similar performance as the $\sigma_{\epsilon,\pi} = 0.01$ controller – total cost of 7.42 compared with 7.44 respectively – for a variance of 0.01^2. Section 6-2 will elaborate on this finding. For the $\sigma_{\epsilon,\pi} = 0.06$ controller the expected behavior is found. The controller is less aggressive than the $\sigma_{\epsilon,\pi} = 0.01$ and 0.04 controllers. Resulting in a poor performance for small measurement noise variances. For measurement noise variances larger than 0.06^2 this controller performs significantly better than the other two controllers. The performance of the $\sigma_{\epsilon,\pi} = 0.01$ is (reasonable) stable for standard deviations between 0.001-0.016, fluctuating between 6.986 and 8.088. For standard deviations larger than 0.016, the total cost increases. For 0.02 (a 100% increase of standard deviation) a 21% increase in total cost is found. However, 0.03 standard deviation (200% increase) shows a 121% increase.

The results of the total cost of the $\sigma_{\epsilon,\pi} = 0.01$ controller are coherent with the found Root Mean Square (RMS) errors, see Figure 6-3. These RMS errors are found using 100 trials. The solid black line is the 'control' experiment. It shows the RMS error of 100 trials using the same measurement noise as during learning, $\mathcal{N}(0, 0.01^2)$. The RMS errors seem to be high even for variances where the total cost is low. This is caused by a few trials that completely fail. More important is that the error increases for bigger variances. Note, that the experiments of the reference and $\alpha = 0.01$ (dotted blue line) have the same conditions. However, the results differ. This is because the control has one complete failure within the 100 trials oppose to four for the trials of $\alpha = 0.01$.

6-1-2 2: Measurement noise skewness

The influence of the skewness of the measurement noise on the performance of the algorithm is tested with the *Skew normal distribution*. This distribution gives the opportunity to change the skewness, with the parameter, α. For a higher value of the parameter α, a higher skewness is obtained. However, the skewness lies on the interval of $[-1\ 1]$. For an infinite skewness parameter the skewness is 1. In Figure 6-4a the skew normal distribution is given for $\alpha = -10, 0, 10$. For a negative value for α a left skewed distribution is obtained and for a positive visa versa. Selecting zero gives the normal distribution. Normally, the mean and variance of the distribution will drift for different values of α. However, with the following relations the mean and variance can be kept constant:

$$\mu = \xi + \omega\delta\sqrt{\frac{2}{\pi}}, \qquad \text{and} \qquad \sigma^2 = \omega^2\left(1 - \frac{2\delta^2}{\pi}\right), \tag{6-4}$$

where

$$\delta = \frac{\alpha}{\sqrt{1+\alpha^2}}, \tag{6-5}$$

and ξ is the location and ω is the scale of the distribution. Note, ξ and ω can be chosen such that the variance and the mean are kept constant.

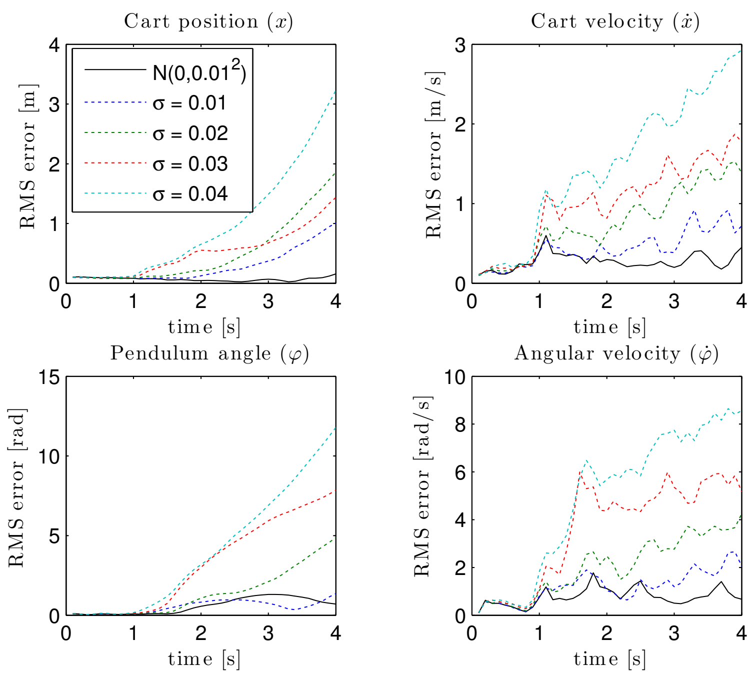
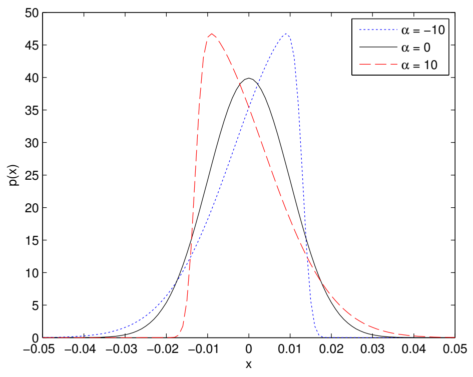
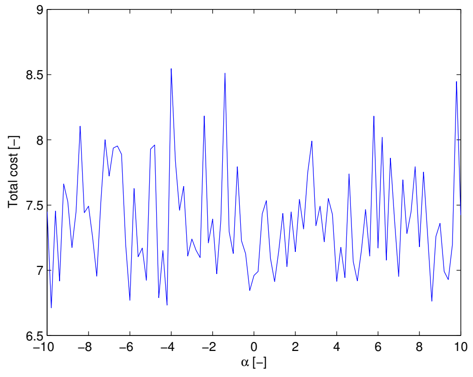


Figure 6-3: Robustness analyzes: the influence of measurement noise variance on the RMS error of 100 trials and the preset predictions of a trial after learning. The used model and controller are learned for Gaussian measurement noise of $\epsilon_t \sim \mathcal{N}(0, 0.01^2)$. The reference – the RMS error of 100 trials using the same measurement noise as during learning – is given in solid black. The other trials where disturbed with zero mean Gaussian measurement noise with a standard deviation of 0.01 (dotted dark blue), 0.02 (dotted green), 0.03 (dotted red), 0.04 (dotted light blue).



(a) Skew normal probability distribution for different skewness parameter α.



(b) Total cost for different skewness parameter α of the zero mean Skewed normal measurement noise.

Figure 6-4: Robustness analyzes: the influence of measurement noise skewness on the cost. In Figure 6-4a the zero mean Skew normal probability distribution is given for different skewness parameter, α. For α equal to zero the original Gaussian distribution is obtained ($\mathcal{N}(0, 0.01^2)$). In Figure 6-4b the mean of the total cost of 100 trials disturbed with zero mean Skewed normal measurement noise is shown. The skewness parameter is varied from -10 to 10. The used controller is learned for Gaussian measurement noise of $\epsilon_t \sim \mathcal{N}(0, 0.01^2)$

In Figure 6-4b the total cost for the different skewness parameters is shown. The mean of the total cost does not show any correlation with the different parameters. The values are all between 6.7 en 8.5 which imply good behavior. Note, that for $\alpha = 0$ the Skew normal distribution is equal to the original Gaussian distribution $\mathcal{N}(0, 0.01^2)$ The higher value of the total cost for a particular parameter is caused by the number of complete failures. Complete failures are a result of stochasticity (them measurement noise) of the system. However, with this data no trend can be found between the stochasticity and the total cost. In order to illustrate the independence of the parameter on the behavior only parameters where the total cost is between 6.7 and 6.8 are selected. The parameter vector is $\alpha = \begin{bmatrix} -9.8 & -6 & -4.6 & -4.2 & 8.6 \end{bmatrix}$, which is an arbitrary selection of parameters.
The same analysis applies to Figure 6-5, where the RMS errors of 100 trials per skewness are given. The results for the extremes of α, the parameter responsible for the best ($\alpha = -9.8$) and the worst ($\alpha = -4$) performance and the value $\alpha = 0$ for which the distribution is equal to the original distribution is enlarged. The RMS error are coherent with the total cost found in Figure 6-4b. The RMS error of $\alpha = -9.8$ shows the least errors on the cart position and pendulum angle. The error of $\alpha = -4$ gives the highest errors for all states at the end time and for the whole trajectory for the pendulum angle and angular velocity. Note, that the trajectories of the velocity errors are similar. During the swing-up phase (0-1.1 seconds) the errors of the different trials are similar and follow the same trajectory. The trajectory can be explained as follows. At 1.1 seconds the pendulum is standing upright. However, the pendulum overshoots from 1.1 seconds till 1.7 seconds. This behavior results in the small error of the pendulum angle and cart position while having a peak in the error of the velocities. The second peak in the velocities is caused by the overshoot in the other direction. From 2 to 4 seconds the pendulum has minor oscillations resulting in different errors.

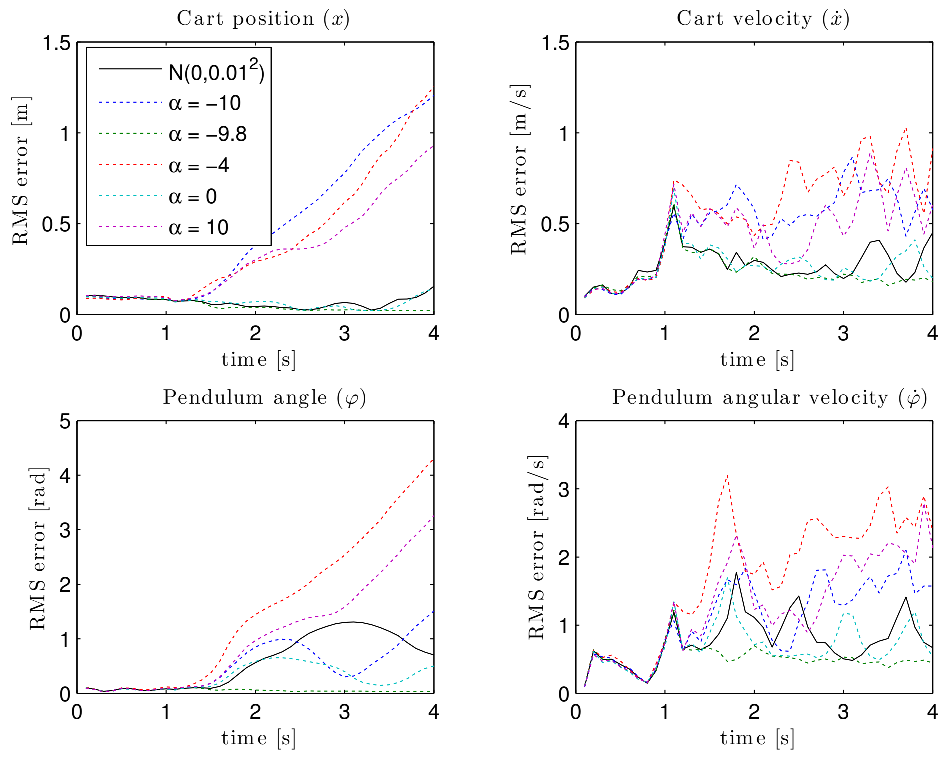


Figure 6-5: Robustness analyzes: the influence of measurement noise skewness on the RMS error of 100 trials and the preset predictions of a trial after learning. The used model and controller are learned for Gaussian measurement noise of $\epsilon_t \sim \mathcal{N}(0, 0.01^2)$. The reference – the RMS error of 100 trials using the same measurement noise as during learning – is given in solid black. The other trials where disturbed with zero mean Skew normal measurement noise with a standard deviation of 0.01 and skewness parameter α of: -10 (dotted dark blue), -4 (dotted green), 0 (dotted red), 10 (dotted light blue).

6-1-3 3: Measurement noise excess kurtosis

The excess kurtosis of a distribution is the normalized fourth standardized moment. Hence, the excess kurtosis of the normal distribution equals zero[1]. In this research the terms excess kurtosis and kurtosis are used interchangeably for simplicity reasons, and refer to the following equation

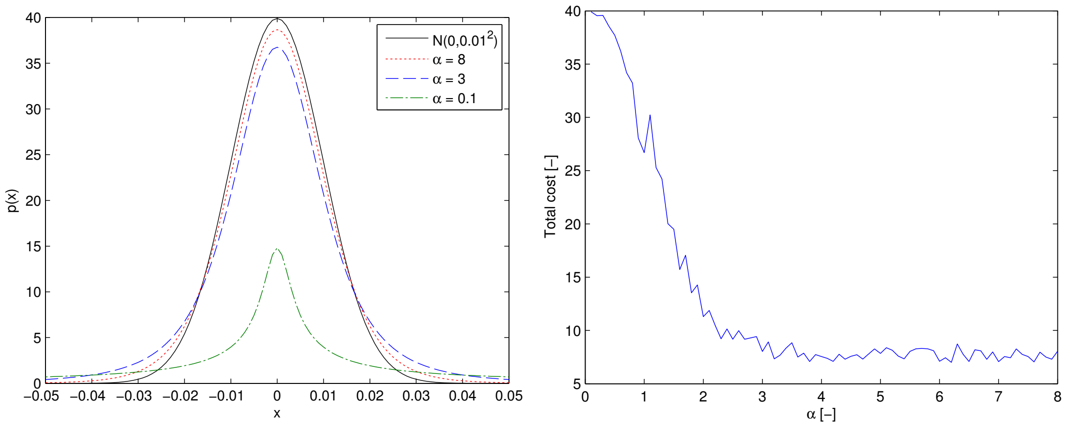
$$\beta_2 = \frac{\mathbb{E}[(X-\mu)^4]}{\mathbb{E}[(X-\mu)^2]^2} - 3. \tag{6-6}$$

The kurtosis is normalized to ensure that the kurtosis of the normal distribution is equal to zero, hence the 'minus 3' in Eq. (6-6). With the Student-T distribution the kurtosis of the distribution can be adjusted by selecting the degree of freedom, α, see Figure 6-6a. For infinite degrees of freedom the distribution is equal to the Gaussian distribution. For values going to zero the kurtosis increases, which means that the change on extreme values increases. The kurtosis can not independently change from the variance as the following relations show:

$$\sigma^2 = \begin{cases} \infty & \text{for } 1 < \alpha \leq 2 \\ \dfrac{\alpha}{\alpha - 2} & \text{for } \alpha > 2 \end{cases}, \tag{6-7}$$

$$\beta_2 = \begin{cases} \infty & \text{for } 2 < \alpha \leq 4 \\ \dfrac{6}{\alpha - 4} & \text{for } \alpha > 4 \end{cases}. \tag{6-8}$$

Note, that the variance and kurtosis are only defined for certain domains. The total cost per degree of freedom of the Student-T distributed measurement noise is given in Figure 6-6b. The total cost converges to the total cost found by a the original Gaussian measurement noise distribution. From 3 degrees of freedom the total cost stays between 7 and 9. Which is high compared with the $\sigma_{\epsilon,\pi} = 0.01$ controller of the Gaussian distribution of



(a) Student-T pdf for different degrees of freedom compared with $\mathcal{N}(0, 0.01^2)$.

(b) Total cost when subjected with zero mean Student-T measurement noise.

Figure 6-6: Robustness analyzes: the influence of measurement noise kurtosis on the cost. In Figure 6-6a the zero mean Student-T probability distribution is given for different degrees of freedom, α. For α equal to infinity the original Gaussian distribution is obtained ($\mathcal{N}(0, 0.01^2)$). In Figure 6-6b the mean of the total cost of 100 trials disturbed with zero mean Student-T measurement noise is shown. The degrees of freedom are varied from 0.1 to 8. The used controller is learned for Gaussian measurement noise of $\epsilon \sim \mathcal{N}(0, 0.01^2)$.

[1]The kurtosis first published by Pearson (1905) is equal to three for the normal distribution $\beta_2 = 3$

Figure 6-2b, where the converged values are between 7 and 8. This is the result of the higher kurtosis, which mean that extreme values are likelier to happen. Hence, bigger disturbances which could lead to total failure are likelier to happen. The best performance is found for 6.2 degrees of freedom with a total cost of 7.01.
In Figure 6-7 the RMS errors of the states are shown. In order to illustrate the influence of the degrees of freedom the following values are selected: $\alpha = \begin{bmatrix} 1.9 & 3.2 & 6.2 & 8 \end{bmatrix}$. For higher degrees of freedom the distribution has lower change of extreme values and approximates the Gaussian distribution better. Hence, the expectation is that for higher degrees of freedom a lower total cost is found. Therefore, is it remarkable that the errors for $\alpha = 8$ are bigger than the errors of $\alpha = 3.2$ or 6.2. However, in Figure 6-6b the expectation is met with a clear convergence for increasing degrees of freedom. The total cost which is related to the RMS errors of Figure 6-7 show a perturbed line. These perturbations are the result of the stochasticity of the measurement noise. This stochastic behavior is responsible for the RMS errors and cause that for $\alpha = 8$ larger errors are found than for $\alpha = 6.2$. Furthermore, for $\alpha = 6.2$ the best performance is found with a total cost of 7.01 while for $\alpha = 6.3$ a total cost of 8.74 is found. Note, that in the velocity plots of Figure 6-7 similar trajectories for all degrees of freedom is found till completion of the swing-up (1.1 seconds).

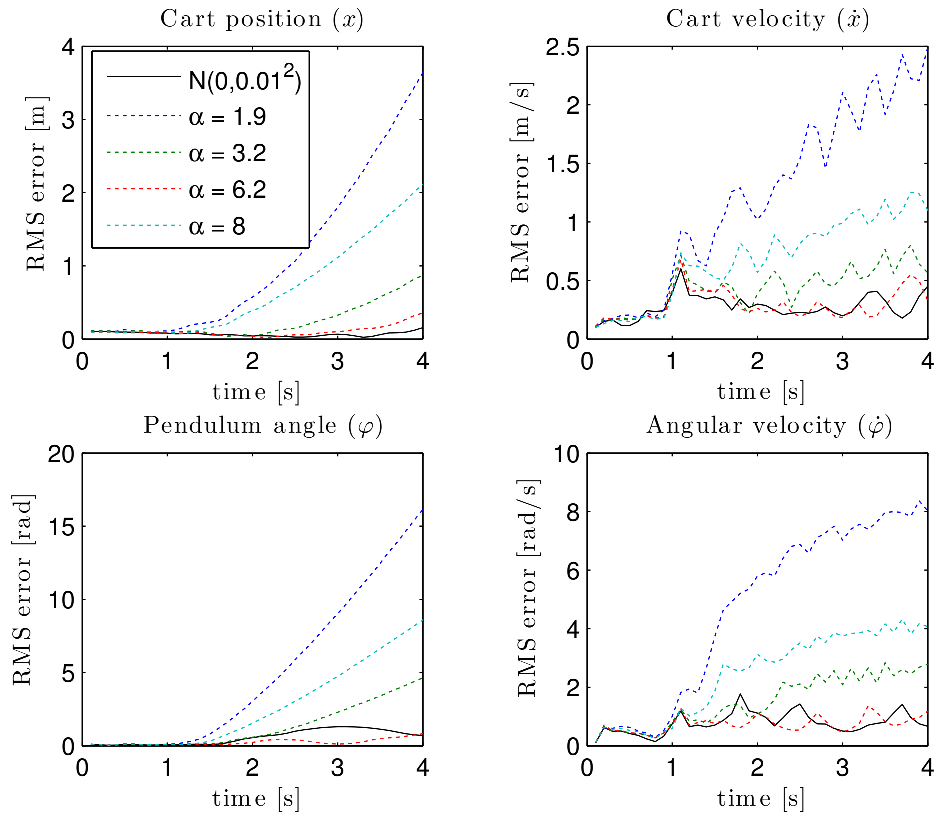


Figure 6-7: Robustness analyzes: the influence of measurement noise kurtosis on the RMS error of 100 trials and the preset predictions of a trial after learning. The used model and controller are learned for Gaussian measurement noise of $\epsilon_t \sim \mathcal{N}(0, 0.01^2)$. The reference – the RMS error of 100 trials using the same measurement noise as during learning – is given in solid black. The other trials where disturbed with zero mean Student-T measurement noise with degrees of freedom α of: 1.9 (dotted dark blue), 3.2 (dotted green), 6.2 (dotted red), 8 (dotted light blue). The variance of the Student-T distribution is 0.01^2 if $\alpha = \infty$.

6-1-4 4: Measurement noise extreme values

To investigate the influence of extreme values of the measurement noise independently of the variance the Salt & Pepper distribution is used. In Figure 6-8a the Salt & Pepper probability distribution is given for different values α, which are the locations of the distribution. In order to maintain a constant variance ($\sigma_\epsilon^2 = 0.01^2$) a third location is introduced at zero. The variance is kept constant using

$$\sigma_\epsilon^2 = \sum_{i=1}^{n} p_i (x_i - \mu)^2. \tag{6-9}$$

For three locations, α_{min}, α_0, α_{plus}, a zero mean distribution and $\alpha_{min}^2 = \alpha_{plus}^2 = \alpha^2$ holds: $p(\alpha_{min}) = p(\alpha_{plus}) = p$ and $p_0 = p(\alpha_0) = 1 - 2p$. The probability p is found by rewriting Eq. (6-9) to

$$p = \frac{\sigma_\epsilon^2}{2\alpha^2}. \tag{6-10}$$

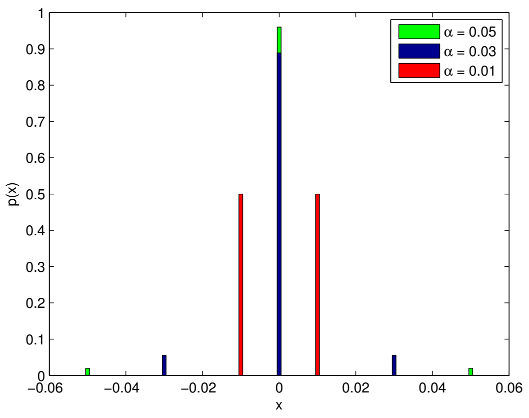
The growing probability p_0 and decreasing probability p for increasing absolute locations of the Salt & Pepper distribution can be seen in Figure 6-8a. Note, that this only hold for locations equal or larger than the standard deviation $\alpha \geq \sigma_\epsilon$. In Figure 6-8b three controllers, learned with different Gaussian measurement distributions, are given for increasing absolute locations of the Salt & Pepper distribution. The $\sigma_{\epsilon,\pi} = 0.01$ controller is given in solid blue, the $\sigma_{\epsilon,\pi} = 0.04$ in dot dashed green and the $\sigma_{\epsilon,\pi} = 0.06$ in dotted red. The good performance of the $\sigma_{\epsilon,\pi} = 0.01$ for all locations of the Salt & Pepper distribution is not strange. By keeping variance constant and increasing the absolute distance of the locations, the probability on a extreme value becomes low. Therefore, a aggressive controller performs well for these distributions. Although, for a absolute locations smaller than $\pm 0.035^2$ the $\sigma_{\epsilon,\pi} = 0.04$ controller gives a more stable performance.
However, the decrease in the performance of the $\sigma_{\epsilon,\pi} = 0.04$ controller is remarkable. The increase of total cost is caused by the cautiousness of the controller, because it considers more measurement noise than present. Note, that the change on a non zero value decreases with increasing locations e.g., for ± 0.05 as the change on either a locations is 2 percent, for ± 0.1 the change is 0.5 percent. Resulting in less optimal control input. For the ± 0.1 locations this implies in practice that over 100 trials 144 times a non zero noise entry is given. Hence, 144 of the (40 time steps times 4 states times 100 trials) 16000 data points. This is equal to 0.9 percent, which correspond with the probability distribution. That a non zero noise value occurs does not imply that the consequences are critical. The influence on the performance system of a non zero noise value on the velocity states are less than for the angle of the pendulum. A non zero noise value on the angle of the pendulum only happened 30 times (0.2%). Therefor, is the $\sigma_{\epsilon,\pi} = 0.04$ controller is to conservative and gives the $\sigma_{\epsilon,\pi} = 0.01$ controller a better overall perforamnce.
The results of Figure 6-8b and Figure 6-2b show similar behavior for $\sigma_{\epsilon,\pi} = 0.04$ for small absolute locations and variances respectively. The $\sigma_{\epsilon,\pi} = 0.06$ controller shows the similar behavior as the $\sigma_{\epsilon,\pi} = 0.04$ controller, it is to conservative for small measurement noise variances, resulting in a higher total cost. This total cost is similar to the total cost found in Figure 6-2b for variances up to 0.06^2.

6-2 Discussion

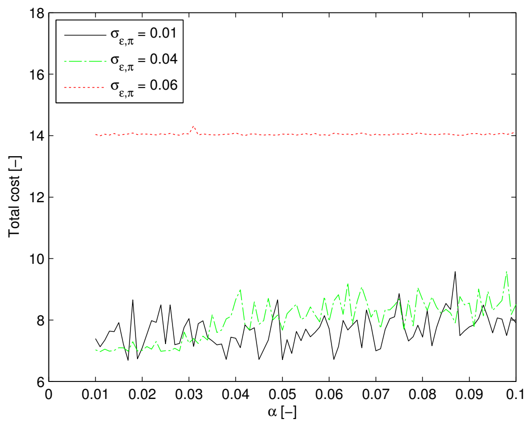
In Figure 6-2b the – more conservative – $\sigma_{\epsilon,\pi} = 0.04$ controller performs better than the $\sigma_{\epsilon,\pi} = 0.01$ for large noise variance, which is expected. However against expectations, controllers perform evenly well for small measurement noise variances. From this result one would suggest that optimizing the controller for a slightly higher variance can result in a more robust controller without having to compromise on performance. However, this is task and system dependent and not true in general.

Sensitivity vs Robustness

Stabilizing a pendulum implies controlling a system to the vertical upward equilibrium and maintaining the system in that state. Stabilizing a system in an equilibrium is an easier task in general than tracking a reference. In a tracking task, a certain reference has to be followed by the system. A fast reacting controller will perform



(a) The Salt & Pepper probability distribution for different locations.



(b) Comparison of total cost of two learned controllers for Salt & Pepper measurement noise.

Figure 6-8: Robustness analyzes: the influence of extreme values of the measurement noise on the cost. In Figure 6-8a the zero mean Salt & Pepper probability distribution is shown for different locations α, 0.01 in red, 0.03 in blue and 0.05 in green. In Figure 6-8b the performance of three with Gaussian distributed measurement noise learned controllers is given. The solid black line is the $\sigma_{\epsilon,\pi} = 0.01$ controller, the green dot dashed line is the $\sigma_{\epsilon,\pi} = 0.04$ controller and dotted red line is the $\sigma_{\epsilon,\pi} = 0.06$ controller.

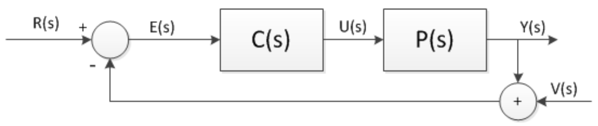


Figure 6-9: Basic feedback control system. The reference is denoted as $R(s)$, the error as $E(s)$, $U(s)$ the control input, $Y(s)$ the output, $V(s)$ the measurement noise, $C(s)$ the controller and $P(s)$ the plant.

better than a more conservative one. The drawback is that a more aggressive controller is also more sensitive to noise. In order to illustrate the trade-off between performance and robustness a linear example of the a basic feedback system is given in Figure 6-9. The output $Y(s)$ and error $E(s)$ are given as:

$$Y(s) = \frac{P(s)C(s)}{1+P(s)C(s)}R(s) - \frac{P(s)C(s)}{1+P(s)C(s)}V(s), \tag{6-11}$$

$$E(s) = R(s) - Y(s) = \frac{1}{1+P(s)C(s)}R(s) + \frac{P(s)C(s)}{1+P(s)C(s)}V(s), \tag{6-12}$$

where the reference is denoted as $R(s)$, $U(s)$ the control input, $V(s)$ the measurement noise, $C(s)$ the controller and $P(s)$ the system (plant). The goal is generally to minimize the error $E(s)$. Hence, the *sensitivity function* $\mathcal{S}(s) = \frac{1}{1+P(s)C(s)}$ and *complementary sensitivity function* $\mathcal{T}(s) = \frac{P(s)C(s)}{1+P(s)C(s)}$ have to be small to track the reference and attenuate the measurement noise. However this is a conflicting property, because

$$\frac{1}{1+P(s)C(s)} + \frac{P(s)C(s)}{1+P(s)C(s)} = 1. \tag{6-13}$$

Which implies that if one function is small the other function has to be large large. Hence, the error can not be zero if measurement noise is present[2]. Note, that $\frac{P(s)C(s)}{1+P(s)C(s)} = 1$ is preferred for reference tracking, but it

[2]When different requirements need to hold for different frequency bands, good solutions can be found.

also increases the influence of the measurement noise on the error. Although it is hard to see from the results – because of the specific task and system used – the same trade-off between performance and robustness holds for the cart-pole system.

6-3 Conclusions

Although, the skewness of the distribution as tested in the above configurations has no significant influence on the performance of the algorithm, the performance decreases significantly for an increasing variance and/or kurtosis. When a kurtosis is selected of less than four degrees of freedom are selected for the Student-T distribution the performance decreases. However, for more than four degrees of freedom the algorithm performs well. Hence, the performance of the algorithm for the cart-pole set-up is robust for finite kurtosis values, see Eq. (6-7).
Similar performance is found for a variance increase from 0.01 to 0.016, further increase of the variance result in a decrease of the performance measure of the algorithm. This shows that the algorithm is sensible for extreme measurement noise values, while being robust for one sided values (positive or negative skewness). For both increasing kurtosis and variance holds that extreme values are more likely to occur. Because the 'tails' of a probability distribution of with a high kurtosis are long, the probability on an extreme value increases slower for increasing kurtosis than for increasing variance. Although, these slopes are different the conclusion is the same, the extreme values cause the controller to fail.
The results of the Salt & Pepper measurement noise distribution verify this. The extreme values of the measurement noise are increased, while keeping a constant variance of 0.01^2. Although, the $\sigma_{\epsilon,\pi} = 0.04$ controller performs optimal and stable if the locations are absolute smaller than 0.03 ($\alpha < |0.03|$), the more aggressive $\sigma_{\epsilon,\pi} = 0.01$ controller performs better overall. This shows that correct information of the variance of the measurement noise leads to better performance of the controller.
Concluding these results: (1) the extreme values of the disturbance cause the controller to fail, (2) correct information of the variance of the measurement noise leads to better performance of the controller. Many systems wear and tear during their lifetime, this wear and tear introduces different uncertainties over time. Hence, tracking the variance of the measurement noise and reacting on changes can lead to a better overall performance. The next chapter elaborates this theory.

Chapter 7

A-PILCO

The work presented in the previous chapter found that: (1) the extreme values of the disturbance cause the controller to fail, (2) correct information of the variance of the measurement noise leads to better performance of the controller. Important to realize is that the variance of the noise is used as a fixed parameter, $\sigma^2_{\epsilon,\pi}$, for the Gaussian Process (GP) controller, see Eq. (3-49).
Wear and tear can change the system dynamics during their lifetime (Banta, 1988). These changes in system dynamics can be modeled as changing uncertainties, resulting in a less optimal controller. Hence, time varying uncertainties will decrease the performance of the controller. Therefore, tacking the variance of the measurement noise and reacting on changes can lead to a better performance. This chapter proposes the Adaptive-Probabilistic Inference for Learning COntrol (A-PILCO) framework an extension to Probabilistic Inference for Learning COntrol (PILCO) to track the variance and decide when to relearn the model and controller parameters.

7-1 The A-PILCO algorithm

Over time the measurement noise characteristics can change. In order to make the algorithm robust for changes of the measurement noise some additions tot he algorithm are needed, see for a simplified implementation Appendix B-2. In this research the change of the distribution is limited to change of the variance of the zero mean independent and identically distributed (iid) Gaussian measurement noise. The cart-pole system of Section 4-2-2 is used throughout this chapter. During control of a system, the model is not used. However, the model is available and can be utilized to find an approximation of the variance of the measurement noise. This is of importance because the controller needs this variance as a fixed parameter, see Eq. (3-49). The GP model, models these dynamics as Eq. (3-9), and can be written for the discrete case as:

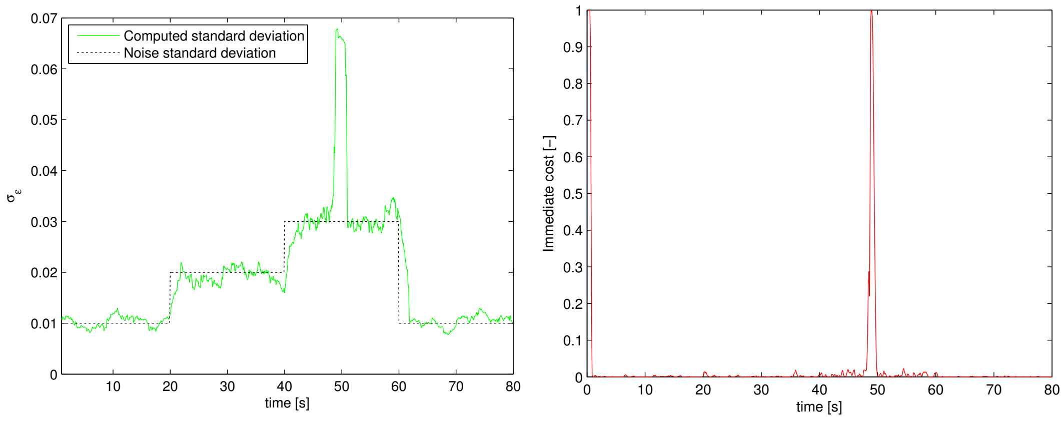
$$\hat{x}_{t+1} = \mu_{t+1} = \hat{x}_t + \mathbb{E}[\Delta_{t+1}], \tag{7-1}$$
$$\hat{y}_{t+1} = \hat{x}_{t+1}. \tag{7-2}$$

Assuming that the model simulates the true system dynamics with high accuracy ($\hat{x_t} \sim x_t$) the measurement noise error can be isolated,

$$\hat{\epsilon}_t = y_t - \hat{y}_t. \tag{7-3}$$

The error $\hat{\epsilon}_t$ is assumed to be zero mean. So for N data samples holds ($\hat{\epsilon}_i \sim \mathcal{N}(0, \hat{\Sigma}_\epsilon)$, for $i = 1 : N$). From these data samples the variance can be computed. The number, N of data used to find the variance determines

the quality of the approximation of the measurement noise variance. More data result in a better estimate of the variance. However, if a large N is selected the changes in measurement noise are noticed slowly. Which results in a long reaction time. Selecting a small N makes the algorithm to adjust fast. A major disadvantage of a fast responding algorithm is that it is sensitive for one or two extreme values. Hence, there is a trade-off between accuracy and the reaction time of the algorithm.
In Figure 7-1a the tracking of the standard deviation of the cart-pole system is shown for $N = 20$ and $\omega = \begin{bmatrix} 1/3 & 1/3 & 0 & 1/3 \end{bmatrix}$, see Eq. (7-5). The measurement noise standard deviation is varied in time. At fifty seconds the controller reacts to aggressive and the pendulum falls down, which can be seen as the peak in the cost function in Figure 7-1b. The pendulum is immediately swung back up again by the controller. However, in this short period the variance estimation is inaccurate, this can be prevented by choosing a larger value of N. In order to prevent that the algorithm starts the whole learning process because of a few extreme data points a threshold must be derived. Because the ϵ_t for $t = 1 : N$ is a sampling from a normal distribution, the χ^2-test is used to test how likely it is, that the approximated variance per state, $\hat{\Sigma}_\epsilon$, is the original variance of Eq. (6-3) (Levine et al., 2007). The χ^2-test per state for N data samples is given as



(a) Tracking of the standard deviation. **(b)** The cost of the corresponding trial.

Figure 7-1: Tracking and decision making: tracking the standard deviation for time varying measurement noise. In Figure 7-1a the standard deviation is tracked using 20 data points ($N = 20$) per estimation. The black dotted line is the true standard deviation of the measurement noise, the green solid line is the approximated standard deviation. Figure 7-1b shows the corresponding total cost of the trial.

$$\chi^2_{N-1} = \frac{(N-1)\hat{\sigma}^2_\epsilon}{\sigma^2_{\epsilon,\pi}}. \tag{7-4}$$

Note, that the variance in the denominator $\sigma^2_{\epsilon,\pi}$ is a parameter of the controller which is fixed for learning, see Eq. (3-49). The confidence intervals of the χ^2 distribution are well known and can be found in tables. In the cart-pole application the 99% confidence interval is used[1]. The variance of the measurement noise, σ^2_ϵ is found per state from N time steps of $\epsilon_t = y_t - \hat{y}_t$. For different systems the noise has different influence per state. In the cart-pole system 30 time steps are used to approximate the variance, $N = 30$.
The approximated variance using the data of the angular velocity has a offset, see Figure 7-2a. Due to this offset the variance computation of the angular velocity exceeds its bounds regularly. Therefore, this state is excluded from the decision making and variance approximation. In Figure 7-2 the standard deviation is changed from 0.01 to 0.02 at 10 seconds. In both figures an initialization phase can be distinguished. During this initialization the approximation of the standard deviation is inaccurate. Therefore, the decision making is only valid from the number of time steps chosen to approximate the standard deviation N.
In Figure 7-2b shows that the χ^2 values for the three states exceed there bounds in a small time period after the variance changes. Therefore, the algorithm will learn a new model and controller if the all three states exceed there bounds.

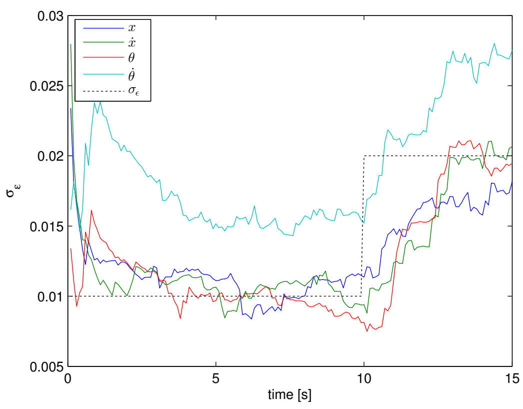
[1]See for a table of χ^2 confidence intervals Appendix A-2

Approximating the measurement noise variance

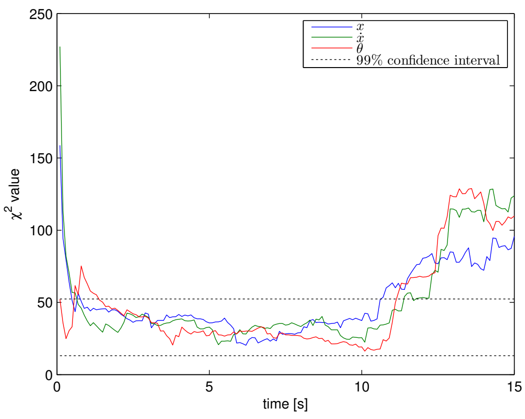
If the χ^2 values cross their bound the A-PILCO will start an variance approximation trial. This trial is designed to accurately approximate the variance of the system. The trial takes 50 seconds and the control input is zero. Because, without any actuation all the fluctuations 'measured' are caused by noise. The variance is approximated from the three states by

$$\hat{\sigma}_\epsilon^2 = w \operatorname{var}[\epsilon_i], \qquad i = 1 : N, \tag{7-5}$$

where $\omega = \begin{bmatrix} 1/3 & 1/3 & 0 & 1/3 \end{bmatrix}$ are the weights of the states. Hence, equal weights for the states are used. Using Eq. (7-5) the variance for a change from $\sigma_\epsilon^2 = 0.01^2 \rightarrow 0.02^2$ is approximated over all time steps as $\hat{\sigma}_\epsilon^2 = 0.0203^2$. Note, that for different systems different weights of the states, total number of data N and confidence intervals are appropriate.



(a) Tracking of the standard deviation per state.



(b) Tracking the χ^2 values per state.

Figure 7-2: Tracking and decision making: tracking the standard deviation and the corresponding χ^2 values. The standard deviation increase from 0.01 to 0.02 at ten seconds. In Figure 7-2a tracking of the standard deviation per state of the cart-pole system with $N = 30$ is presented (solid lines). The true standard deviation of the measurement noise is shown in dotted black. In Figure 7-2b the tracking of the χ^2 values per state (solid lines) are given. The 99% confidence interval is shown with the dotted black lines.

7-2 Validation of the GP model

The A-PILCO algorithm assumes that the model simulates the system dynamics accurate, such that Eq. (7-3) holds. However, the angular velocity of in Figure 7-2a does not approximate the standard deviation well and suggests a model error. This is the also the argument to omit the angular velocity from the decision making and variance approximation, see Section 7-1 and 7-1. Therefor, the model has to be verified. The GP model approximates the non linear dynamics of the system. In case of a model error the approximation can be written as:

$$\hat{x}_{t+1} = \hat{f}(x_t, u_t) + g(x_t, u_t), \tag{7-6}$$

$$\hat{y}_t = \hat{x}_t. \tag{7-7}$$

If the approximation is perfect $g(x_t, u_t) = 0$ and $y_t - \hat{y}_t = \epsilon_t$, and ϵ_t has the same characteristics as the measurement noise. From Figure 7-2a it seems that the angular velocity estimate has an error, i.e. $g(x_t, u_t) \neq 0$. For further analyzes of the model error, the relative increase of the measured standard deviation is computed.

In Figure 7-3 the mean value of the standard deviation per state is shown as the solid lines. The trial takes 20 seconds and at 10 seconds the variance of the iid Gaussian measurement noise increases from 0.01^2 to 0.02^2. These values are computed using the all data except the data from the initialization phase of the standard deviation approximation ($t < N$ at $t = 0$ and $t = 100$). The found values for the two cases are extrapolated to show a clear image of the increase of the estimates. The relative differences are given in Table 7-1. For the results of the first two rows (variance increases up to 0.03^2) the approximations of the relative differences are accurate. This implies that the error in the angular velocity (remember Figure 7-2a) is a constant, $g(x_t, u_t) = c$. For larger increases of the variances the estimates of the velocity states blow up. These results indicates that for changes up to 0.03^2 the model is able to approximate all the states accurate except the angular velocity. The angular velocity estimates have a constant modeling error. The model is verified using two methods: (1) the *Variance Accounted For (VAF)* and (2) the *residual test*.

Table 7-1: Relative difference of the approximated standard deviation per state, for increased measurement noise variance. For a trial of 20 seconds the original $\sigma_{\epsilon,\pi} = 0.01$ controller is used. However, the variance of the iid Gaussian measurement noise is increased at 10 seconds as indicated in the first column.

σ_ϵ^2	cart position [m]	cart velocity [m/s]	angle of the pendulum [rad]	angular velocity [rad/s]
$0.01^2 \to 0.02^2$	0.0092	0.0092	0.0099	0.0106
$0.01^2 \to 0.03^2$	0.0172	0.0214	0.0216	0.0266
$0.01^2 \to 0.04^2$	0.0329	0.1399	0.0417	0.3230
$0.01^2 \to 0.05^2$	0.0472	0.0832	0.0489	0.1772
$0.01^2 \to 0.06^2$	0.0632	1.1538	0.1049	1.2797
$0.01^2 \to 0.07^2$	0.0604	0.1456	0.0667	0.3294

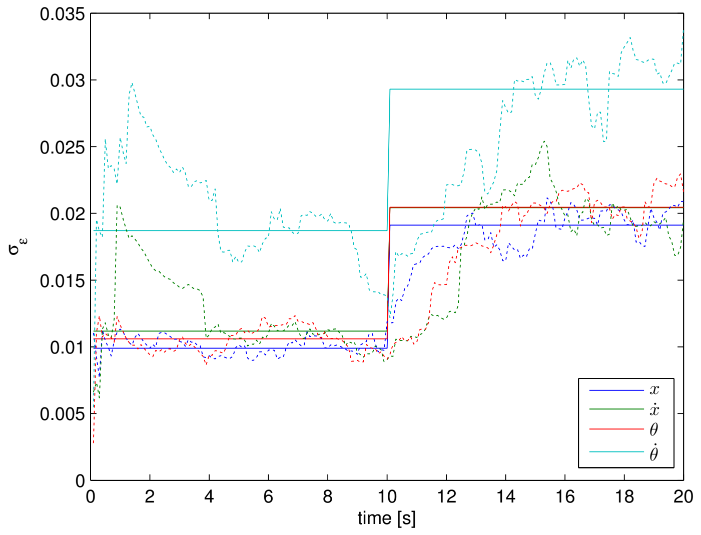


Figure 7-3: Model validation: the relative difference of the approximated standard deviation per state, for increased measurement noise variance. The dotted lines are the tracked values of the standard deviations. The solid lines are mean of the approximated values per case: $\sigma_\epsilon = 0.01$ for $t < 10$ and $\sigma_\epsilon = 0.02$ for $t > 10$. Equal colours for the dotted and solid lines correspond to the same state: blue is the cart position , green the cart velocity, red the angle of the pendulum and light blue the angular velocity of the pendulum.

7-2-1 1: Variance Accounted For

To analyze the accuracy of the model the *VAF* is used (Verhaegen and Verdult, 2007). The VAF indicates how well the model fits the true data. The VAF is computed as

$$\text{VAF} = \left(1 - \frac{\text{var}\,[y - \hat{y}]}{\text{var}\,[y]}\right) \times 100\%. \tag{7-8}$$

In Table 7-2 the VAF values per state are given for a cart-pole trial of 100 seconds (1000 data points). Note, that the model is tested with different data than it is learned, *cross-validation*. The influence different variances of the iid Gaussian measurement noise, σ_ϵ^2 on the VAF is investigated by increasing the measurement noise, while the controller is kept the same. The values of Table 7-2 indicated that the model simulates the true system dynamics accurate, and the model error can be neglected. This is in conflict with the measurements of the standard deviation, remember Figure 7-2a.

Table 7-2: Model validation: VAF per state of the cart-pole system. Trials of 100 seconds (1000 input-output pairs) of the $\sigma_{\epsilon,\pi} = 0.01$ controller for different measurement noise variances σ_ϵ^2 are used to compute the VAF.

σ_ϵ^2	cart position [%]	cart velocity [%]	angle of the pendulum [%]	angular velocity [%]
0.01^2	92.3344	99.8145	99.8103	99.9433
0.02^2	85.6107	99.7021	99.3682	99.9310
0.03^2	99.9999	99.7713	99.9908	99.5042
0.04^2	99.9991	81.3672	99.9999	98.3155
0.06^2	99.9992	87.6440	99.9999	98.6248

7-2-2 2: Residual test

Another method to verify the accuracy of the model is to check the distribution of the error, the *residual test*. If the model is accurate the error is given as Eq. (7-3), which should be equal to the measurement noise. A histogram of the error vector should match the measurement noise distribution. In Figure 7-4 the histogram per state and the probability distribution of the original Gaussian measurement noise ($\mathcal{N}(0, 0.01^2)$) is shown for the trials used to compute the VAF (1000 data points). The distribution of the angular velocity does not resembles the original distribution, however the VAF is high (99.94% see Table 7-2). In Figure 7-5 this difference in results is even more clear. The distributions are given for the trial where the variance is equal to 0.04^2, while maintaining the original $\sigma_{\epsilon,\pi} = 0.01$ controller. Note, that the same trial is used to compute the VAF with 0.04^2 measurement noise variance. The model estimate of the cart position is stil acurate, however the pendulum angle and velocities are not. The error distribution of the pendulum angle is left skewed and looks like a Skew normal distribution. The error distributions of the velocities looks like a Student-T distribution with a high kurtosis, i.e. long tails. Remarkable is the minor decrease in VAF for the pendulum angle and the velocities, while the residual test shows a model mismatch.

7-2-3 Conclusions

The minor decrease of the VAF, while increasing the variance of the measurement noise are the result of the relative small variance of the measurement noise compared to the values of the states. The large difference in values for the states result in a big variance of the output. Which result in a high VAF, see Eq. (7-8). Therefore, the VAF values can not verify the model, however the residual test can, remember Figure 7-4. Although, it is found using the residual test that the angular velocity estimates are not accurate, – hence, Eq. (7-3) does not hold for the angular velocity – the model can be used for learning in the (A-)PILCO algorithm. Note, that for the decision making and approximations of the variance the angular velocity has to be omitted or the model error has to be filtered out. Note, that this conclusion is system and task dependent. For other systems different values for N

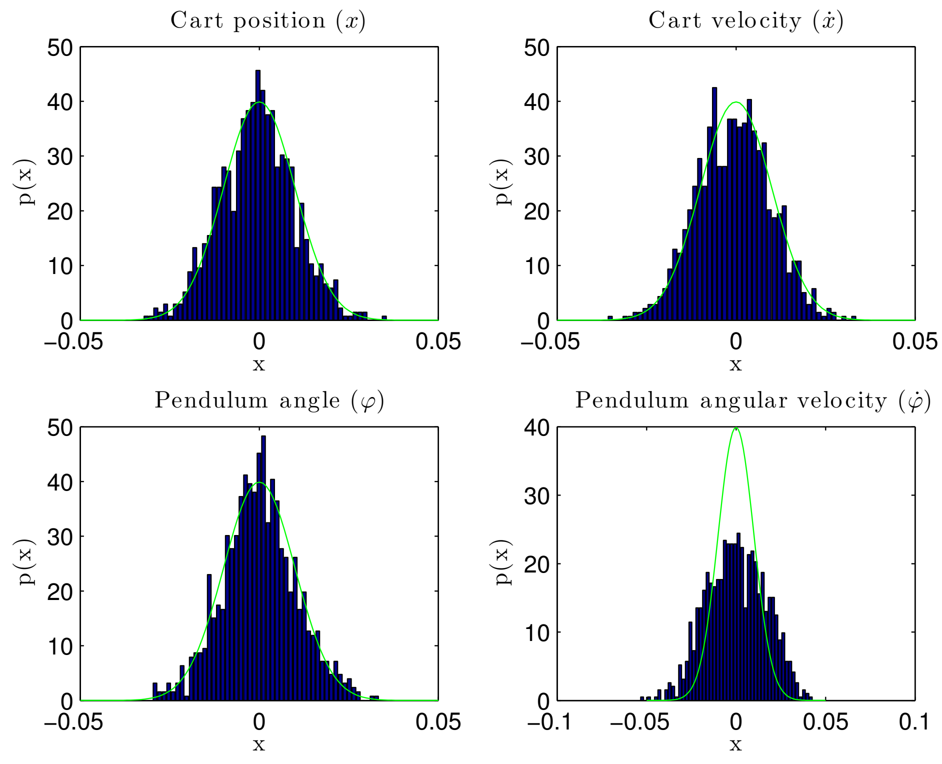


Figure 7-4: Model validation: the probability distribution of residual error of the cart-pole system disturbed with $\mathcal{N}(0, 0.01^2)$ measurement noise per state. The probability distribution of the error per state $\tilde{\epsilon}_t$ is presented with the blue bars, the green solid line is the probability distribution of the true error $p(\epsilon_t) = \mathcal{N}(0, 0.01^2)$.

7-3 Results

Expected is that correct information of the variance of the measurement noise leads to the optimal controller. Remember, that the controller variance is set a priori before learning the other controller parameters. Figure 7-6 shows that these expectations are met. The blue line is the original PILCO algorithm, which implies that the controller is learned for the original measurement distribution of $\mathcal{N}(0, 0.01^2)$. The dotted red line is the performance of A-PILCO, where controller noise variance is set equal to the measurement noise variance, $\sigma_{\epsilon,\pi} = \sigma_\epsilon$. The results of A-PILCO are dependent on the convergence of the learning algorithm. Due to stochasticity and the non-convex optimization problem the learned controller parameters are not equal after separate learning sessions. Hence, the performance can vary for separate learning session. However, Figure 7-6 shows that the performance of A-PILCO is significantly better than the performance of PILCO for time varying measurement noise. Note, that during learning of the new controller knowledge of the old controller is still available. This knowledge can be used to increase the learning convergence of the new controller.

Implementing knowledge

When the three χ^2 values – corresponding to the angle of the pendulum, position and velocity of the cart – exceeds the bound the algorithm re-learns the model and controller. However, the controller corresponding with the previous measurement noise variance is known. This controller can be used for initial trials. Hence, instead of trials with a random control input – $u_t = \mathcal{U}(-u_{max}, u_{max})$ – the algorithm can use trials with the old controller.
In Figure 7-7 the variance of the iid Gaussian measurement noise is changed from 0.01^2 to 0.04^2. The learning session with no implemented knowledge is presented as the blue solid line The red dotted line presents the

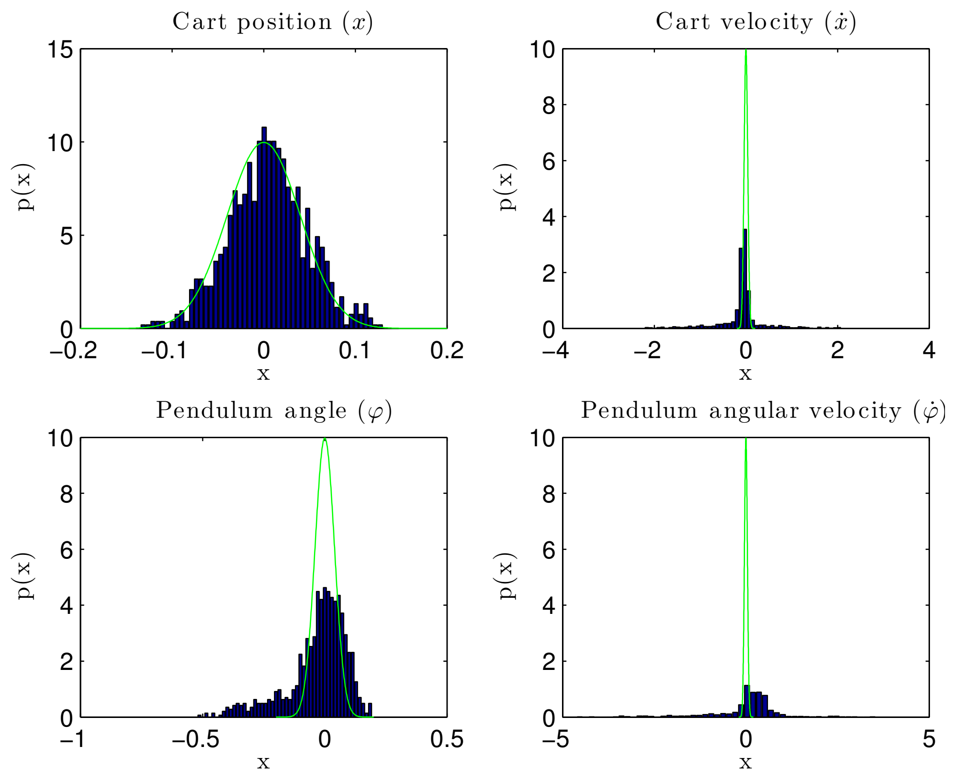


Figure 7-5: Model validation: the probability distribution of residual error of the cart-pole system disturbed with $\mathcal{N}(0, 0.04^2)$ measurement noise per state. The probability distribution of the error per state $\tilde{\epsilon}_t$ is presented with the blue bars, the green solid line is the probability distribution of the true error $p(\epsilon_t) = \mathcal{N}(0, 0.04^2)$.

learning session when one trial with the old $\sigma_{\epsilon,\pi} = 0.01$ controller is used. The learning session with two trials of the old controller is presented with the green dashed line. The new controller is learned after each trial, commencing after the second initialization trial. Using the old controller more diverse and usable data of the system dynamics is obtained, resulting in a more complete model. Because of the better model, better solutions can be found. Note, that the algorithm needs less iterations to find a good solution, when more knowledge is implemented. Furthermore, is the convergence more stable. Suggesting a deeper valley of the solution space is found for these controllers. Note, that for both cases the solution converses to the same value. The policy search is non-convex and every optimization can lead to a different solution. By implementing knowledge – using the old controller – more important system dynamics can be captured within the data. Better knowledge of the system dynamics could lead to faster convergence and a better policy.

7-4 Conclusions

In the previous chapter is found that: (1) the extreme values of the disturbance cause the controller to fail, (2) correct information of the variance of the measurement noise leads to better performance of the controller. This information is used to develop A-PILCO, an framework to handle time varying measurement noise. A-PILCO initiates a new learning session if the measurement noise variance exceeds its bounds. By defining the 99% confidence interval of the χ^2 distribution, a threshold is derived for the desicion making. The performance of A-PILCO shows a significant inprovement for environments with time varying measurment noise. Furthermore, implementing knowledge a priori – the old controller is used for an initial learning trial – could lead to faster convergence and better policies.

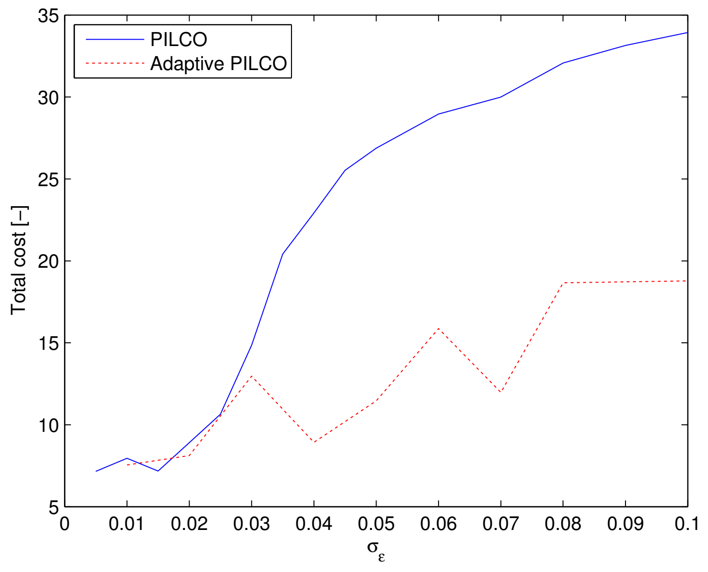


Figure 7-6: A-PILCO: performance comparison of PILCO and A-PILCO for increasing measurement noise variances. The mean of the total cost of 100 trials is given for measurement noise variance from $\sigma_\epsilon^2 = 0.01^2$ to 0.1^2 . The blue solid line is the performance of PILCO optimized for $\sigma_\epsilon^2 = 0.01^2$. The red dotted line is the performance of A-PILCO.

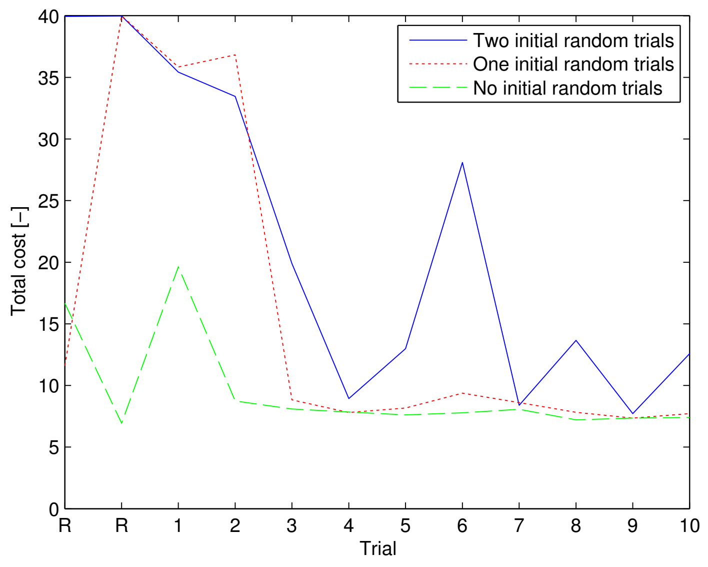


Figure 7-7: A-PILCO: convergence comparison of policy learning with and without prior knowledge. The learning iteration sequences of the $\sigma_{\epsilon,\pi} = 0.04$ controller for the cart-pole system with iid zero mean Gaussian measurement noise with a variance of 0.04^2 is presented. The first two trials are initialization trials. The case where both initial trials are with a random control input $u_t = \mathcal{U}(-u_{max}, u_{max})$ – without prior knowledge – is given in solid blue. The case where the first trial is with the old $\sigma_{\epsilon,\pi} = 0.01$ controller – including prior knowledge – is presented with the red dotted line. The green dashed case uses the old controller for both initialization trials.

Chapter 8

Conclusions and recommendations

The Probabilistic Inference for Learning COntrol (PILCO) algorithm has been made applicable for a larger set of systems. By reducing the computational time of the identification process the algorithm is made applicable for large systems, i.e. systems with a higher state dimension. Furthermore is Adaptive-Probabilistic Inference for Learning COntrol (A-PILCO) developed. A-PILCO is an adaptive extension to PILCO to cope with time varying measurement noise by initiating a new learning process if the measurement noise variance exceeds its bounds. In this chapter the conclusions will be presented first. The recommendations for further research will be presented in Section 8-2.

8-1 Conclusions

The conclusions will be presented according to the four sub-objectives presented in Section 1-2.

1. Identify if the Gaussian Process (GP) controller has learning advantages over the Radial Basis Function (RBF) controller.
2. Reduce the computational time of the algorithm by parallel computing.
3. Identify the influence of different measurement noise characteristics to gain insight in the robustness of the algorithm.
4. Develop an framework that can cope with time varying measurement noise.

8-1-1 Sub-objective 1: Identify if the GP controller has learning advantages over the RBF controller.

In Chapter 4 this sub-objective is addressed. When the correct number of basis functions is chosen, both controllers find good policies for a second order system and the cart pole system. However if the difficulty of the optimization process is increased by increasing the number of basis functions, the GP controller shows advantages. The algorithm using the RBF controller did not find any solution. Although the algorithm was not able to find the optimal solution, using the GP controller a reasonable suboptimal solution was found. The indirect representation of the weighting factors have a smoothening effect on the optimization function, this effect eases the optimization. Hence, the GP controller has better convergence properties than the RBF controller.

8-1-2 Sub-objective 2: Reduce the computational time of the algorithm by parallel computing.

Chapter 5 elaborates on sub-objective 2. The gradient based learning of the controller parameters implies one optimization. The computations in the optimization are not demanding enough to justify the extra overhead time introduced by the parallel implementation. However, the framework of the GP training is ideal for parallel computing, because the computations of the GP training involve E independent optimizations, where E is the target dimension. Although the GP learning is less time demanding than the policy learning part of the PILCO algorithm, the time decrease of the GP training itself can be decreased up to 40 percent for a four dimensional system. For small systems the GP training part is negligible, 1% of the total coputation time is used to train the GP models for the cart-pole system for eigth iterations of 2.5 seoconds (total of 20 seconds). However, if the number of training data is increased to 1000 the GP training takes 33% of the total computation time. Hence, for large and/or complex systems the parallelization can reduce the total computational time significantly.

8-1-3 Sub-objective 3: Identify the influence of different measurement noise characteristics to gain insight in the robustness of the algorithm.

Sub-objective 3 is presented in Chapter 6. The performance of the algorithm is hardly influenced by the skewness of the measurement noise distribution. The kurtosis and variance have a clear effect on the performance. Note, that the kurtosis was not tested separately from the variance. The algorithm shows a decrease in performance for increasing variance or kurtosis. This implies that extreme values of the measurement noise decrease the performance of the algorithm. Furthermore, the performance of the algorithm remains equal when the measurement noise extremes are increased while keeping a constant variance. This can be explained by the rapid decreasing probability of the extreme values. A measure for the probability of an extreme value is the variance. Note, that the measurement noise variance is directly used by the GP controller as a fixed parameter.

8-1-4 Sub-objective 4: Develop an framework that can cope with time varying measurement noise.

In Chapter 7 sub-objective 4 is presented. From the investigation of the influence of different moments on the distribution show a decrease in performance for increasing measurement noise variance. Furthermore, the measurement noise variance is used as a fixed parameter in the GP controller. It is important for an optimal behavior that used variance of the controller corresponds with the true measurement noise variance. Therefore, an extension is made for the algorithm, A-PILCO. This extension tracks the variance of the measurement noise and initiate a new learning process if the measurement noise variance of the individual states exceeds their confidence bounds. Assuming an accurate model, the residual error of the true system minus the model is Gaussian distributed. Therefore, the confidence bounds are found using the χ^2 distribution. By choosing the number of data samples and the confidence interval wisely, the number of false relearn initiations can be limited. In this way the (negative) influence of time varying measurement noise on the performance can be minimized.

8-2 Recommendations

The recommendations are split in two categories: real experiment recommendations, and extensions and improvements.

8-2-1 Further research – real life implementations

In this thesis the performance evaluations of the algorithms are limited to simulations of a second-order system and a cart-pole system. Real life experiments should verify and extend these results.

1. Identify to which extend the PILCO algorithm with parallel implementation for the GP training is applicable for large systems by real life experiments.
2. The performance of A-PILCO should be verified on a real life system with time varying measurement noise.

8-2-2 Further research – extensions and improvements

The PILCO algorithm uses batches of data to find the (sub)optimal controller. Hence, the interaction with the system is interrupted by the algorithm with computational demanding learning processes. An algorithm that can learn while interacting with the system (online learning) is more ideal. It is not realistic that PILCO can be made online with minor improvements, because of the high computational load. Recommendations to reduce the computation load and an possible online solution are given below:

1. To reduce the computational load of the controller learning process, the trade off between accuracy and computational load has to be further investigated. In the algorithm the prediction of the next state using moment matching is the most computational demanding. However, an accurate prediction is vital for learning. In the recent to publish paper (Deisenroth et al., 2013a) a linear estimator is compared with moment matching. Although the linear estimator is computational less expensive (4-5 times), the performance is poor. After 15 to 20 seconds of experience (6 to 8 learning iterations) with the cart-pole system, only 83% of the trials where succesful with the linear estimator compared with 95% when using moment matching. Therefore, more learning iterations are required which decrease the advantage of the computational speed. The linear estimator finds a smaller variance for its predictions than moment matching does. Making the prediction more certain, which increases the probability of failure due to mode-bias.
   Good estimations are important to the algorithm. Therefore, an investigation to multimodal prediction methods, such as the Gaussian sum estimators (Anderson and Moore, 2005), the Expectation Correction (Barber and Chickering, 2006) or the recent developed Multi-Modal Filter (M-MF) (Kamthe et al., 2013) are interesting. The methods are using a sum of Gaussian approximations to find an estimate the posterior. Where, Expectation Correction is developed for switching linear systems and involves a forward (filtering) and backward (smoothing) pass. These methods and possibly computational more expensive. However, if it reduces the number of necessary learning iterations it could reduce the total computational time.
2. In further research the computational load of the sparse GP training can be reduce using the single step global sparse implementation. The sparse implementation of the GP training requires a full GP training first, followed by selecting the m^1 best Gaussian kernels. Hence, two synchronous optimizations are performed. This two step GP learning approach is more computational intensive than using a single sparse approximation, however it solves the problem of over- or underfitting and sequential data. This sequential data problem can be avoided when global basis functions are used. Lázaro-Gredilla et al. (2010) show promising results with their global Sparse Spectrum Gaussian Process (SSGP) algorithm using trigonometric basis functions.
3. Develop an online PILCO framework that interacts with the system, trains a GP model, and learns a controller in parallel. Hester and Stone (2012) proposes and shows good results for controlling an autonomous vehicle using Real-Time Model-Based Reinforcement Architecture (RTMBA) . Which is – as the name suggests – a model-learning architecture which is able to run in real time, because of a parallel framework. There is no prior knowledge available to the algorithm and a Random Forest model was used (Hester and Stone, 2010). Like the PILCO, this algorithm takes the uncertainty of the model into account.

[1] $m << N$, where N is the number of training data

Appendix A

Mathematical Tools

In this appendix two tools are given: (1) the method to augment angular states with the complex values, and (2) the χ^2 confidence intervals.

A-1 2: Augmented states

To improve the accuracy of the Gaussian Process (GP) model angles can be represented by there complex values during learning, i.e. there sin(·) and cos(·) values. The *full state* is augmented with the*augmented state*:

$$x = \begin{bmatrix} x_1 \\ \theta \end{bmatrix}, \qquad x^{augm} = \begin{bmatrix} \sin(\theta) \\ \cos(\theta) \end{bmatrix}, \qquad x^{full} = \begin{bmatrix} x_1 \\ \theta \\ \sin(\theta) \\ \cos(\theta) \end{bmatrix}. \tag{A-1}$$

During learning only the complex expressions of the angle, and not the angle itself, are used for the training inputs. The 'original' expression is used for the training targets. Note, that when using the complex notation the state dimension of the training inputs will increase by one. Following the same example, with u_t as control input, the training inputs, $\tilde{X}$, and targets, $\mathbf{y}$, will consist of the following vectors:

$$\tilde{X} = \begin{bmatrix} x_1 \\ \sin(\theta) \\ \cos(\theta) \\ u \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} x_1 \\ \theta \end{bmatrix}. \tag{A-2}$$

This augmented state can also be helpful in other calculations, such as the cost of the current state. In order to to use this augmented state the distribution of the state $p(x_t) \sim \mathcal{N}(\mu_t, \Sigma_t)$ must be pushed through the trigonometric functions. The distribution of the augmented state can be found using these properties. The distribution of the augmented state, $p(x_t^{augm}) = \mathcal{N}(\mu_t^{augm}, \Sigma_t^{augm})$, can than be found as

$$p(x_t^{augm}) = \mathcal{N}\left(\begin{bmatrix} \mathbb{E}[\sin(\theta)|\mu_t, \Sigma_t] \\ \mathbb{E}[\cos(\theta)|\mu_t, \Sigma_t] \end{bmatrix}, \begin{bmatrix} \mathrm{var}\left[\sin(\theta)|\mu_t, \Sigma_t\right] & \mathrm{cov}\left[\sin(\theta), \cos(\theta)|\mu_t, \Sigma_t\right] \\ \mathrm{cov}\left[\sin(\theta), \cos(\theta)|\mu_t, \Sigma_t\right]^T & \mathrm{var}\left[\cos(\theta)|\mu_t, \Sigma_t\right] \end{bmatrix}. \right) \tag{A-3}$$

The mean of can be found as:

$$\mathbb{E}[\sin(\theta)|\mu_t,\Sigma_t] = \int \sin(\theta)p(\theta)ds = \exp\left(-\frac{\Sigma_t}{2}\right)\sin(\mu_t), \tag{A-4}$$

$$\mathbb{E}[\cos(\theta)|\mu_t,\Sigma_t] = \int \cos(\theta)p(\theta)ds = \exp\left(-\frac{\Sigma_t}{2}\right)\cos(\mu_t). \tag{A-5}$$

For the covariance matrix of the augmented state, the variance of the sinus and cosines, and the covariance between them are required. The expression required for the covariance matrix of the augmented state can be found as:

$$\mathrm{var}\left[\sin(\theta)|\mu_t,\Sigma_t\right] = \mathbb{E}[\sin(\theta)^2|\mu_t,\Sigma_t] - \mathbb{E}[\sin(\theta)|\mu_t,\Sigma_t]^2, \tag{A-6}$$

$$\mathrm{var}\left[\cos(\theta)|\mu_t,\Sigma_t\right] = \mathbb{E}[\cos(\theta)^2|\mu_t,\Sigma_t] - \mathbb{E}[\cos(\theta)|\mu_t,\Sigma_t]^2, \tag{A-7}$$

$$\mathrm{cov}\left[\sin(\theta),\cos(\theta)|\mu_t,\Sigma_t\right] = \mathbb{E}[\sin(\theta)\cos(\theta)|\mu_t,\Sigma_t] - \mathbb{E}[\sin(\theta)|\mu_t,\Sigma_t]\mathbb{E}[\cos(\theta)|\mu_t,\Sigma_t]. \tag{A-8}$$

Using

$$\mathbb{E}[\sin(\theta)^2|\mu_t,\Sigma_t] = \int \sin(\theta)^2 p(\theta)ds = \frac{1}{2}(1-\exp\left(-2\Sigma_t\right)\cos(2\mu_t)), \tag{A-9}$$

$$\mathbb{E}[\cos(\theta)^2|\mu_t,\Sigma_t] = \int \cos(\theta)^2 p(\theta)ds = \frac{1}{2}(1+\exp\left(-2\Sigma_t\right)\cos(2\mu_t)), \tag{A-10}$$

$$\mathbb{E}[\sin(\theta)\cos(\theta)|\mu_t,\Sigma_t] = \int \sin(\theta)\cos(\theta)p(\theta)ds = \int \frac{1}{2}\sin(2\theta)p(\theta)ds, \tag{A-11}$$

$$= \frac{1}{2}\exp\left(-2\Sigma_t\right)\sin(2\mu_t), \tag{A-12}$$

the expression of the distribution of the full state becomes

$$p(x_t^{full}) = \mathcal{N}\left(\begin{bmatrix}\mu_t\\ \mu_t^{augm}\end{bmatrix}, \begin{bmatrix}\Sigma_t & \mathrm{cov}\left[x_t, x_t^{augm}|\mu_t,\Sigma_t\right]\\ \mathrm{cov}\left[x_t, x_t^{augm}|\mu_t,\Sigma_t\right]^T & \Sigma_t^{augm}\end{bmatrix}.\right) \tag{A-13}$$

Though, $\mathrm{cov}\left[x_t, x_t^{augm}|\mu_t,\Sigma_t\right]$ is still unknown. This covariance between the (original) state and the augmented state can be seen is the covariance between the augmented state the the corresponding angle of the state:

$$\mathrm{cov}\left[x_t, x_t^{augm}|\mu_t,\Sigma_t\right] = \mathrm{cov}\left[x_t,\theta|\mu_t,\Sigma_t\right]\mathrm{cov}\left[\theta, x_t^{augm}|\mu_t,\Sigma_t\right], \tag{A-14}$$

$$\mathrm{cov}\left[\theta, x_t^{augm}|\mu_t,\Sigma_t\right] = \begin{bmatrix}\mathbb{E}[\cos(x)|\mu_t,\Sigma_t] & \mathbb{E}[\sin(x)|\mu_t,\Sigma_t].\end{bmatrix} \tag{A-15}$$

Because of the independence of the other variables, in the state with the augmented state, only the covariance values corresponding to the original angle (including covariances of this angle and the states) and the augmented state will be non zero. To select the covariance values of the angle, the corresponding column vector can be sliced out. Or multiply the state covariance matrix with with an sparse matrix where only the covariance values of the angle with the augmented complex angle in the bottom row:

$$\mathrm{cov}\left[x_t, x_t^{augm}|\mu_t,\Sigma_t\right] = \Sigma_t \begin{bmatrix}\emptyset & \emptyset\\ \mathbb{E}[\cos(x)|\mu_t,\Sigma_t] & -\mathbb{E}[\sin(x)|\mu_t,\Sigma_t]].\end{bmatrix} \tag{A-16}$$

A-2 2: Chi-Square confidence interval table

The Chi^2 confidence intervals are required for decision making of Adaptive-Probabilistic Inference for Learning COntrol (A-PILCO). If the states exceed their confidence intervals an new learning session is initiated. Below are the confidence intervals given up to 30 degrees of freedom. The confidence intervals for higher degrees of freedom can be found, for instance at http://www.medcalc.org/manual/chi-square-table.php.

Table A-1: Tracking and decision making: χ^2 confidence intervals for up to 30 degrees of freedom.

P **DF**	**0.995**	**0.975**	**0.20**	**0.10**	**0.05**	**0.025**	**0.02**	**0.01**	**0.005**	**0.002**	**0.001**
1	0.0000393	0.000982	1.642	2.706	3.841	5.024	5.412	6.635	7.879	9.550	10.828
2	0.0100	0.0506	3.219	4.605	5.991	7.378	7.824	9.210	10.597	12.429	13.816
3	0.0717	0.216	4.642	6.251	7.815	9.348	9.837	11.345	12.838	14.796	16.266
4	0.207	0.484	5.989	7.779	9.488	11.143	11.668	13.277	14.860	16.924	18.467
5	0.412	0.831	7.289	9.236	11.070	12.833	13.388	15.086	16.750	18.907	20.515
6	0.676	1.237	8.558	10.645	12.592	14.449	15.033	16.812	18.548	20.791	22.458
7	0.989	1.690	9.803	12.017	14.067	16.013	16.622	18.475	20.278	22.601	24.322
8	1.344	2.180	11.030	13.362	15.507	17.535	18.168	20.090	21.955	24.352	26.124
9	1.735	2.700	12.242	14.684	16.919	19.023	19.679	21.666	23.589	26.056	27.877
10	2.156	3.247	13.442	15.987	18.307	20.483	21.161	23.209	25.188	27.722	29.588
11	2.603	3.816	14.631	17.275	19.675	21.920	22.618	24.725	26.757	29.354	31.264
12	3.074	4.404	15.812	18.549	21.026	23.337	24.054	26.217	28.300	30.957	32.909
13	3.565	5.009	16.985	19.812	22.362	24.736	25.472	27.688	29.819	32.535	34.528
14	4.075	5.629	18.151	21.064	23.685	26.119	26.873	29.141	31.319	34.091	36.123
15	4.601	6.262	19.311	22.307	24.996	27.488	28.259	30.578	32.801	35.628	37.697
16	5.142	6.908	20.465	23.542	26.296	28.845	29.633	32.000	34.267	37.146	39.252
17	5.697	7.564	21.615	24.769	27.587	30.191	30.995	33.409	35.718	38.648	40.790
18	6.265	8.231	22.760	25.989	28.869	31.526	32.346	34.805	37.156	40.136	42.312
19	6.844	8.907	23.900	27.204	30.144	32.852	33.687	36.191	38.582	41.610	43.820
20	7.434	9.591	25.038	28.412	31.410	34.170	35.020	37.566	39.997	43.072	45.315
21	8.034	10.283	26.171	29.615	32.671	35.479	36.343	38.932	41.401	44.522	46.797
22	8.643	10.982	27.301	30.813	33.924	36.781	37.659	40.289	42.796	45.962	48.268
23	9.260	11.689	28.429	32.007	35.172	38.076	38.968	41.638	44.181	47.391	49.728
24	9.886	12.401	29.553	33.196	36.415	39.364	40.270	2.980	45.559	48.812	51.179
25	10.520	13.120	30.675	34.382	37.652	40.646	41.566	44.314	46.928	50.223	52.620
26	11.160	13.844	31.795	35.563	38.885	41.923	42.856	45.642	48.290	51.627	54.052
27	11.808	14.573	32.912	36.741	40.113	43.195	44.140	46.963	49.645	53.023	55.476
28	12.461	15.308	34.027	37.916	41.337	44.461	45.419	48.278	50.993	54.411	56.892
29	13.121	16.047	35.139	39.087	42.557	45.722	46.693	49.588	52.336	55.792	58.301
30	13.787	16.791	36.250	40.256	43.773	46.979	47.962	50.892	53.672	57.167	59.703

Appendix B

Matlab code

The simulations result presented in this theses are acquired using MATLAB. For the PILCO and A-PILCO simulations the `pilcoV0.9` toolbox is used (Deisenroth et al., 2013b). The code was modified if needed. Two functions will be presented below: (1) the parallel implementation of the GP training, and (2) the tracking and decision making of the A-PILCO algorithm.

B-1 1: Parallel implementation of the GP training

```
%% train.m
% *Summary:* Parallel training of a full GP model with SE covariance
% function (ARD). Modified from the PILCO toolbox pilcoV0.9 of Deisenroth
% et al., 2013. Which is publicly available at
% http://mloss.org/software/view/508/.
%
%   function [gpmodel nlml] = train(gpmodel, dump, iter)
%
% *Input arguments:*
%
%   gpmodel                  GP structure
%     inputs                 GP training inputs
%     targets                GP training targets
%     hyp (optional)         GP log-hyper-parameters
%     induce (optional)      pseudo inputs for sparse GP
%   dump                     not needed for this code, but required
%                            for compatibility reasons
%   iter                     optimization iterations for training
%                            [full GP, sparse GP]
%
% *Output arguments:*
%
%   gpmodel                  updated GP structure
%   nlml                     negative log-marginal likelihood
%
% Last modification: 2014-01-13
% by: Koen van Witteveen
%%

function [gpmodel nlml] = train(gpmodel, dump, iter)

% 1. Initialization

% 2. Parallel train full GP
fprintf('Parallel train hyper-parameters of full GP ...\n');

iter = iter(1); hyptemp = gpmodel.hyp;
inputstemp = gpmodel.inputs; targetstemp = gpmodel.targets;

parfor i = 1:E % train each GP separately
  fprintf('GP %i/%i\n', i, E);
    [hyptemp(:,i), v] = minimize(lh(:,i), @hypCurb, iter, covfunc, ...
      inputstemp, targetstemp(:,i), curb);
  nlml(i) = v(end);
end
gpmodel.hyp = hyptemp;
end % function
```

B-2 2: The A-PILCO implementation

```
%% rollout.m
% *Summary:* Simplified code to demonstrate A-PILCO. The code is written
% to fit PILCO toolbox pilcoV0.9 of Deisenroth et al., 2013. Which is
% publicly available at http://mloss.org/software/view/508/.

% The trajectory is computed using the system dynamics and using the
% GP model. The standard deviation and correpsonding chi-squared values
% are tracked. The interaction with the system is aborts if the
% chi-squared values exceeds there bounds. The bounds, chi-squard
% confidence intervals, are inputs of this function.
%
% function [x y L latent adaptout] = rollout(start, policy, H,...
%                                              plant, cost, adaptin)
%
% *Input arguments:*
%
%   start       vector containing initial states (without controls)
%   policy      policy structure
%     .fcn        policy function
%     .p          parameter structure (if empty: use random actions)
%     .maxU       vector of control input saturation values
%   H           rollout horizon in steps
%   plant       the dynamical system structure
%     .subplant   (opt) additional discrete-time dynamics
%     .augment    (opt) augment state using a known mapping
%     .constraint (opt) stop rollout if violated
%     .poli       indices for states passed to the policy
%     .dyno       indices for states passed to cost
%     .odei       indices for states passed to the ode solver
%     .subi       (opt) indices for states passed to subplant function
%     .augi       (opt) indices for states passed to augment function
%   cost          cost structure
%   adaptin       adaptive inputs structure
%     .Nb           number of data samples for variance approxiamtion
%     .w            state weights for st. dev approximation
%     .bound        chi-square bounds
%     .GPmodel      GP model
%
% *Output arguments:*
%
%   x           matrix of observed states
%   y           matrix of corresponding observed successor states
%   L           cost incurred at each time step
%   latent      matrix of latent states
%   adaptout    adaptive outputs structure
%     .yhat        GP model prediction
%     .stdev        approximated standard deviation
%     .STdev        true standard deviation of the noise
%     .sigma_contr standard devaiation of the controller
%     .T            chi-squared values
%
% Last modification: 2014-01-13
% by: Koen van Witteveen
%%

function [x, y,yhat, L, latent,stdev,STdev,sigma_contr,T] = ...
                    rollout_adaptive(start, policy, H, plant, cost,
                        adaptin)

  % 0. initializations

for i = 1:H %  generate trajectory

  % 1. Apply policy
    u(i,:) = policy.fcn(policy,x(i,1:D),zeros(length(D)));

  % 2. Simulate dynamics using 'real' dynamics
  next = simulate(state(1:E), u(i,:), plant);

  % 2.1 Predict next state using GP model
  adaptout.yhat(i+1,:) = propagate_adaptive(state(1:D)', 0*eye(D),...
                                        plant, adaptin.GPmodel, u(i,:));

  % 3. Update state and add measurement noise
  state = next;
  x(i+1,:) = state + plant.noisefcn(1:E,plant);

  % 5. Compute Cost
  if nargout > 2
    L(i) = cost.fcn(cost,state(1:full)',zeros(length(full)));
  end
```

```matlab

  % 6. Compute the variance of the residual error
  if i > Nb % number of data samples used for approximation
    vartest(i,:) = var(x(i-Nb+2:i+1,simi) - adaptout.yhat(i-Nb+2:i+1,:));
  else
    vartest(i,:) = var(x(1:i+1,simi) - adaptout.yhat(1:i+1,:));
  end

  % 7. Chi-squared value  Eq. (7-4)
  adaptout.T(i,:) = ((adaptin.Nb-1)*vartest(i,:))/(policy.sigma_contr^2);

  % 8. Tracking the standard deviation
  adaptout.stdev(i,1) = sqrt(adaptin.w*vartest(i,:)'); % Eq. (7-5)
  adaptout.STdev(i,1) = plant.sigma;
  adaptout.sigma_contr(i,1) = policy.sigma_contr;

  % 9. Stop if the variance of the residual error exceeds bounds
  if  i > Nb && sum(adaptout.T(i,[1 2 4]) > adaptin.bound(1,1))==0
       disp('Control variance is off - Define new variance')
      break
  elseif i > Nb && sum(adaptout.T(i,[1 2 4]) < adaptin.bound(1,2))==0
      disp('Control variance is off - Define new variance')
      break
  end

end % time steps

  %10. Set ouputs

end % function
```

Bibliography

Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *In Advances in Neural Information Processing Systems 19*, page 2007. MIT Press.

Anderson, B. D. O. and Moore, J. B. (2005). *Optimal Filtering*. Dover Publications.

Atkeson, C. G. and Santamaria, J. C. (1997). A Comparison of Direct and Model-Based Reinforcement Learning. In *In International Conference on Robotics and Automation*, pages 3557–3564. IEEE Press.

Atkeson, C. G. and Schaal, S. (1997). Robot learning from demonstration. In Douglas H. Fisher, J., editor, *Machine Learning: Proceedings of the Fourteenth International Conference (ICML '97)*, pages 12–20. Morgan Kaufmann, San Fransisco, CA.

Banta, L. (1988). A self tuning navigation algorithm. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 1313–1314 vol.2.

Barber, D. and Chickering, M. (2006). Expectation correction for smoothed inference in switching linear dynamical systems. *Journal of Machine Learning Research*, 7:2006.

Daw, N., 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. and Rasmussen, C. (2009). Efficient Reinforcement Learning for Motor Control. In *10th International PhD Workshop on Systems and Control, a Young Generation Viewpoint*, Hluboka nad Vltavou, Czech Republic.

Deisenroth, M. P. (2010). *Efficient Reinforcement Learning using Gaussian Processes*. Karlsruhe series on intelligent sensor-actuator-systems / karlsruherinstitut für technologie, intelligent sensor-actuator-systems laboratory, Karlsruher Institut für Technologie.

Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2013a). Gaussian Processes for Data-Efficient Learning in Robotics and Control. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) , accepted for publication.

Deisenroth, M. P., McHutchon, A., Hall, J., and Rasmussen, C. E. (2013b). PILCO policy search framework. `http://mloss.org/software/view/508/`.

Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *In Proceedings of the International Conference on Machine Learning*.

Ernst, D., Stan, G.-B., Gongalves, J., and Wehenkel, L. (2006). Clinical data based optimal STI strategies for HIV: a reinforcement learning approach. In *Decision and Control, 2006 45th IEEE Conference on*, pages 667 –672.

Grondman, I., Buşoniu, L., Lopes, G. A., and Babuška, R. (2012). A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Transactions on Systems Man and Cybernetics—Part C:Applications and Reviews.*

Hester, T. and Stone, P. (2010). Real Time Targeted Exploration in Large Domains. In *The Ninth International Conference on Development and Learning.*

Hester, T. and Stone, P. (2012). Learning and Using Models. In Wiering, M. and Otterlo, M., editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 111–141. Springer Berlin Heidelberg.

Joel, D., Niv, Y., and Ruppin, E. (2002). Actor-critic models of the basal ganglia: new anatomical and computational perspectives. *Neural Networks*, 15(4âĂŞ6):535– 547.

Kamthe, S., Peters, J., and Deisenroth, M. P. (2013). Multi-modal filtering for non-linear estimation. *Cornell University Library.*

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, pages 2619–2624 Vol.3. IEEE.

Kolter, Z., Plagemann, C., Jackson, D. T., Ng, A., and Thrun, S. (2010). A Probabilistic Approach to Mixed Open-loop and Closed-loop Control, with Application to Extreme Autonomous Driving. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Anchorage, Alaska, USA.

Körding, K. and Wolpert, D. (2004). Bayesian integration in sensorimotor learning. *Nature*, 427(6971):244–247.

Körding, K. and Wolpert, D. (2006). Bayesian decision theory in sensorimotor control. *Trends in Cognitive Sciences*, 10(7):319 – 326. <ce:title>Special issue: Probabilistic models of cognition</ce:title>.

Kuvayev, L. and Sutton, R. (1996). Model-Based Reinforcement Learning with an Approximate, Learned Model. In *In Proceedings Of The Ninth Yale Workshop On Adaptive And Learning Systems*, pages 101–105.

Lázaro-Gredilla, M., Quiñonero Candela, J., Rasmussen, C. E., and Figueiras-Vidal, A. R. (2010). Sparse Spectrum Gaussian Process Regression. *J. Mach. Learn. Res.*, 99:1865–1881.

Levine, D. M., Berenson, M. L., Stephan, D., Krehbiel, T. C., and Ng, P. T. (2007). *Statistics for Managers Using Microsoft Excel (5th Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Lewis, F. and Vrabie, D. (2009). Reinforcement learning and adaptive dynamic programming for feedback control. *Circuits and Systems Magazine, IEEE*, 9(3):32 –50.

Miall, R. and Wolpert, D. (1996). Forward Models for Physiological Motor Control. *Neural Networks*, 9(8):1265–1279. <ce:title>Four Major Hypotheses in Neuroscience</ce:title>.

Ninness, B. and Henriksen, S. (2010). Bayesian system identification via Markov chain Monte Carlo techniques. *Automatica*, 46(1):40 – 51.

Ninness, B., Wills, A., and Mills, A. (2013). UNIT: A Freely Available System Identification Toolbox. *IFAC Control Engineering Practice*, 21(5):631–644.

Pearson, K. (1905). ŞDAS FEHLERGESETZ UND SEINE VERALLGEMEINER-UNGEN DURCH FECHNER UND PEARSON.Ť A REJOINDER. *Biometrika*, 4(1-2):169–212.

Randløv, J. and Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471.

Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning.* MIT Press.

Rummery, G. A. and Niranjan, M. (1994). On-Line Q-Learning Using Connectionist Systems. Technical report, Cambridge University Engineering Department.

Schaal, S. (1997). learning from demonstration. In *advances in neural information processing systems 9*, pages 1040–1046. mit press.

Schneider, J. (1997). Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning. In *Advances in Neural Information Processing Systems 9,*, page 1047, San Francisco. Morgan Kaufmann.

Simão, H. P., Day, J., George, A. P., Gifford, T., Nienow, J., and Powell, W. B. (2009). An approximate dynamic programming algorithm for large-scale fleet management: A case application. *Transportation Science*, 43(2):178–197.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press.

Tesauro, G. (1994). TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, 6(2):215–219.

Verhaegen, M. and Verdult, V. (2007). *Filtering and system identification: a least squares approach*. Cambridge university press.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge, England.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292. 10.1007/BF00992698.

Glossary

List of Acronyms

RL	Reinforcement Learning
POMDP	Partially Observable Markov Decision Process
RBF	Radial Basis Function
GP	Gaussian Process
PILCO	Probabilistic Inference for Learning COntrol
SE	Squared Exponential
RTMBA	Real-Time Model-Based Reinforcement Architecture
CPU	Controller Processor Unit
RMS	Root Mean Square
A-PILCO	Adaptive-Probabilistic Inference for Learning COntrol
VAF	Variance Accounted For
ML-II	type II maximum likelihood
OE	Output Error
CG	Conjugate Gradient
pdf	probability density function
BFGS	Broyden-Fletcher-Goldfarb-Shanno
iid	independent and identically distributed
SSGP	Sparse Spectrum Gaussian Process
M-MF	Multi-Modal Filter

List of Symbols

(A)-PILCO Symbols

$\mathbf{y}$	training targets matrix $\mathbf{y} = [\Delta_1, ..., \Delta_n]^T$
χ_i^2	Chi-squared value for i degrees of freedom
Δ	target $\Delta_t = x_t - x_{t-1} + \epsilon_t \in \mathbb{R}^E$
δ_{pq}	Kronecker delta for time indices p and q
ℓ	characteristic-length scale
ϵ_t	measurement noise
Λ	relevance determination matrix $\Lambda = \text{diag}\left(\ell_i^2, ..., \ell_{D+F}^2\right)$

μ_t	mean of the state
π_t	unsaturated control input $\pi_t = \pi(x_t) \in \mathbb{R}^F$
$\S(x_t)$	saturation function
σ_f^2	variance of the latent function $f(x)$
$\sigma_{\epsilon,\pi}^2$	variance of the GP controller
σ_ϵ^2	variance of the noise
σ_c^2	Width of the cost function
θ	hyper-parameter vector
$\tilde{\cdot}$	all variables with a tilde are constructed of the tuple $(x_t, u_t) \in \mathbb{R}^{D+F}$
$\tilde{X}$	training input matrix $\tilde{X} = [\tilde{X}_1, ..., \tilde{X}_n]$
$\tilde{X}_i$	training input vector $\tilde{X}_i = (x_i, u_i)$
$\tilde{x}_t$	test input tuple $\tilde{x}_t = (x_t, u_t)$
$J^\pi(\cdot)$	expected return using the current policy
K	covariance matrix from the covariance function $K_{ij} = k(x_i, x_j)$
$k(x_i, x_j)$	covariance function
$m(x)$	mean function
n	number of training data
$u_t \in \mathbb{R}^F$	saturated control input $u_t = S(\pi(x_t))$
u_{max}	maximum control input
$x_t \in \mathbb{R}^D$	state
$y_t \in \mathbb{R}^E$	observation of a system

GP Controller Symbols

$\mathbf{y}_\pi \in \mathbb{R}^n$	training targets
ℓ	width of the characteristic-length scale
Λ	relevance determination matrix $\Lambda = \text{diag}\left(\ell_i^2, ..., \ell_D^2\right)$
$\pi(x_t, \psi)$	GP controller
ψ	hyper-parameter vector
$\sigma_{\epsilon,\pi}^2$	controller variance, ideally equal to the measurement noise
$\sigma_{f,\pi}^2$	'variance function' has to be set to one $\sigma_{f,\pi}^2 = 1$
n	number of training data
$X_\pi \in \mathbb{R}^{n*D}$	training inputs

RBF Controller Symbols

ℓ	width of the characteristic-length scale
Λ	relevance determination matrix $\Lambda = \text{diag}\left(\ell_i^2, ..., \ell_D^2\right)$
$\pi(x_t, \psi)$	RBF controller
ψ	hyper-parameter vector
n	number of training data
$w \in \mathbb{R}^n$	weighting vector
$x_\pi \in \mathbb{R}^D * n$	centers of basis functions

Cart-Pole System Symbols

$\dot{\varphi}$	[rad/s] angular velocity of the pendulum
$\dot{x}$	$[m/s]$ cart velocity
φ	$[rad]$ angle of the pendulum measured anti-clockwise from hanging down
b	$[N/m/s]$ friction coefficient between the cart and ground
g	$[m/s^2]$ acceleration of gravity
l	$[m]$ length of the pendulum
M	$[kg]$ mass of cart
m	$[kg]$ mass of pendulum
x	$[m]$ cart position

Index