Learning from Demonstration with Gaussian Processes

Master of Science (M.Sc.) Thesis in Computer Science
submitted by

Markus Schneider

September 2009

Thesis Committee

Dr. Wolfgang Ertel
Dr. Holger Voos

Abstract

In the last years robots turned from simple preprogrammed tools into highly flexible machines which are very complex. Therefore it is very difficult to program such a robot and other human-robot interfaces are needed. This is one reason why *Learning from Demonstration* (LfD) had become a major topic in the context of robotics. This thesis presents a probabilistic framework for robot skill acquisition using *Gaussian processes*. The proposed approach is able to extract all necessary informations for a reproduction from a relatively small number of demonstrations and is also capable to observe the common characteristics of the task.

Acknowledgments

First of all I want to thank my adviser *Wolfgang Ertel* who made this interesting thesis possible and for his guidance during the whole time.

I also have to thank *Christian Bieber, Richard Cubek, Philipp Ertle, Tobias Fromm, Joachim Feßler, Arne Usadel* and *Michel Tokic* for all the advice, support and help. All other members from the Robotics department of the University Ravensburg-Weingarten for creating a nice atmosphere filled with interesting discussions. I also want to thank *Sylvain Calinon*. His work inspired this thesis and helped me a lot with my questions.

Finally, I want to thank all my friends and family for supporting me on my way! My girlfriend for her understanding during this thesis and my roomie *Miguel Gasca* for the interesting discussions.

Contents

List of Symbols

Learning from Demonstration

n	Number of Trajectories
T	Trajectory length
N	Dataset size (nT)
M	Number of objects
D	Dimension of input space
$\mathcal{D}$	Dataset $\mathcal{D} = \{(\xi_t^{(i)}, \boldsymbol{\xi}_s^{(i)}) \vert i = 1, \ldots, N\}$
$\boldsymbol{\xi}$	State variable $\boldsymbol{\xi} = (\xi_t^{(i)}, \boldsymbol{\xi}_s^{(i)})$ $(\in \mathbb{R}^D)$
ξ_t	Temporal part of state variable $(\in \mathbb{R})$
$\boldsymbol{\xi}_s$	Spatial part of state variable $(\in \mathbb{R}^{D-1})$
$\xi_{s,i}$	i-th component of spatial variable $(\in \mathbb{R})$
$\theta_1, \ldots, \theta_k$	Joint space coordinates (angles) for the k joints
$\boldsymbol{\theta}$	Vector of joint space coordinates
$x, y, z, \alpha, \beta, \gamma$	Task space coordinates (robot pose)
$x_1, \ldots, x_6$	Task space coordinates (alternative notation)
$\boldsymbol{x}$	Vector of task space coordinates
$\dot{z}$	first derivative of z
$\boldsymbol{I}$	The identity matrix
$\boldsymbol{J}$	The Jacobian matrix
$\boldsymbol{J}^\dagger$	The pseudoinverse of the Jacobian $\boldsymbol{J}^\dagger = (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T$
$\hat{\boldsymbol{\theta}}$, $\boldsymbol{\Sigma}^{\hat{\theta}}$	Mean and variance estimation for the joint angles
$\hat{\boldsymbol{x}}$, $\boldsymbol{\Sigma}^{\hat{x}}$	Mean and variance estimation for the pose

Gaussian Processes

$\vert\boldsymbol{M}\vert$	Determinant of matrix $\boldsymbol{M}$
$\mathcal{GP}$	Gaussian process: $f \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$
$\mathcal{D}$	Dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}) \vert i = 1, \ldots, n\}$

$\boldsymbol{M}^T$	Transpose of $\boldsymbol{M}$
D	Dimension of input space
$\|\boldsymbol{x}\|$	Euclidean length of vector $\boldsymbol{x}$
$\propto$	Proportional to
$\approx$	Approximately equal to
$\sim$	Distributed according to
$\nabla_{\boldsymbol{\theta}}$	Partial derivatives with respect to $\boldsymbol{\theta}$
$\text{cov}[\boldsymbol{f}_*]$	Gaussian process posterior covariance
δ_{pq}	Kronecker delta function. $\delta_{pq} = 1$ if $p = q$ and 0 else
$\mathbb{E}[g(\boldsymbol{x})]$	Expectation of $g(\boldsymbol{x})$
y	Noisy observation
$f(\boldsymbol{x})$ or $\boldsymbol{f}$	Gaussian process latent function values, $\boldsymbol{f} = [f(x_1), \ldots, f(x_n)]^T$
$\boldsymbol{f}_*$	Gaussian process posterior prediction
$\bar{\boldsymbol{f}}_*$	Gaussian process posterior mean
$\boldsymbol{I}$ or $\boldsymbol{I}_n$	Identity matrix (size $n \times n$)
$k(\boldsymbol{x}, \boldsymbol{x}')$	Covariance function evaluated at $\boldsymbol{x}$ and $\boldsymbol{x}'$
$\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$	Covariance matrix (size $n \times n$) of training inputs
$\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*)$	Covariance matrix (size $n \times n_*$) of training and test inputs
$\boldsymbol{R}(\boldsymbol{X})$	Noise covariance matrix (size $n \times n$) of training inputs
$\boldsymbol{R}(\boldsymbol{X}_*)$	Noise covariance matrix (size $n_* \times n_*$) of test inputs
$\boldsymbol{K}_y$	$\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I}$, noise included covariance matrix
$\log(a)$	Natural logarithm
l	Characteristic length scale
$m(\boldsymbol{x})$	Gaussian process mean function
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
n	Number of training inputs
n_*	Number of test inputs
σ_n^2	Noise variance
$\boldsymbol{\theta}$	Vector of hyperparameters
$\text{tr}\,(\boldsymbol{M})$	Trace of matrix $\boldsymbol{M}$
$\mathcal{X}$	Input space
$\boldsymbol{X}$	The design matrix (size $D \times n$) of training inputs
$\boldsymbol{X}_*$	Matrix of training inputs (size $D \times n_*$)

List of Figures

List of Algorithms

CHAPTER 1

Introduction

1.1 Motivation

The work with a robot raises the question of how one can endow the machine with the ability to perform a complex task. One possibility is to program the robot or informally: you tell the robot explicitly what to do. This is usually too hard or time consuming for even simple tasks. Most often this can be seen in industrial environments where nearly all situations that might occur can be considered and the robot has only a limited workspace. For autonomous robots the complexity of pure programming increases to a different order of magnitude. An example is the Honda humanoid robot [Hirai et al., 1998] which behavior (simple locomotion and basic object manipulation) took about 10 years to develop. The same results can be achieved faster today, but nevertheless the complexity of programming a robot rises nonlinear with the degrees of freedom.

The other extreme is autonomous learning a task from scratch. This is usually done with *Reinforcement Learning* (RL) [Sutton and Barto, 1998]. Learning proceeds by exploring different actions in every state and the use of value functions to organize the search for good policies. Whereas this is very promising for small problems it is unfeasible for nearly all behaviors of interest. The state and action space that the robot is exposed to is simply too high dimensional[1]. For instance, a humanoid robot has 30 or more degrees of freedom. Therefore it is necessary to provide more information of the task or the behavior, else it would take too much time and memory.

The solution is obvious: Task-specific expert knowledge about the intention has to be contributed in advance to scale down the space of possibilities [Schaal, 1999].

1 This is called the course of dimensionality.

Instead of learning isolated, a robot is now able to benefit from the experience of a teacher (this can be a human or another robot). A way to do this is Robot Learning from Demonstrations. Learning from Demonstrations is in the middle of the two extreme approaches: Programming and Learning. It can be seen as programming by ”showing” instead of ”telling”.

Once a task had been demonstrated, how can we encode this data in a way that it is useable for the robot? The actuators must be mapped from the teacher to the actuators of the agent/robot, the observations must be converted to the agent's internal representation. Usually there is also a lot of noise that influences the sensor data. Consequently other representations than simple tabular mappings from states to actions have to be used. Such a representation will be discussed in chapter 2.

Another problem is that normally the teacher is not perfect and therefore also the demonstrations are biased. In order to fulfill a task as good as possible the robot must be able to improve the demonstrated behavior. It is also required to adapt earlier learned skills to changed environment or a different context. This leads again to the field of learning robots, but now the search space for the learning algorithms is extremely constrained by the teacher. This is still ongoing research but a lot of success has been made in the last years. For example [Guenter et al., 2007], [Atkeson and Schaal, 1997], [Peters, 2007], [Peters and Schaal, 2007] [Peters et al., 2007] and others used Reinforcement Learning to improve demonstrated behavior. Although this is a very interesting topic it will be not part of this thesis.

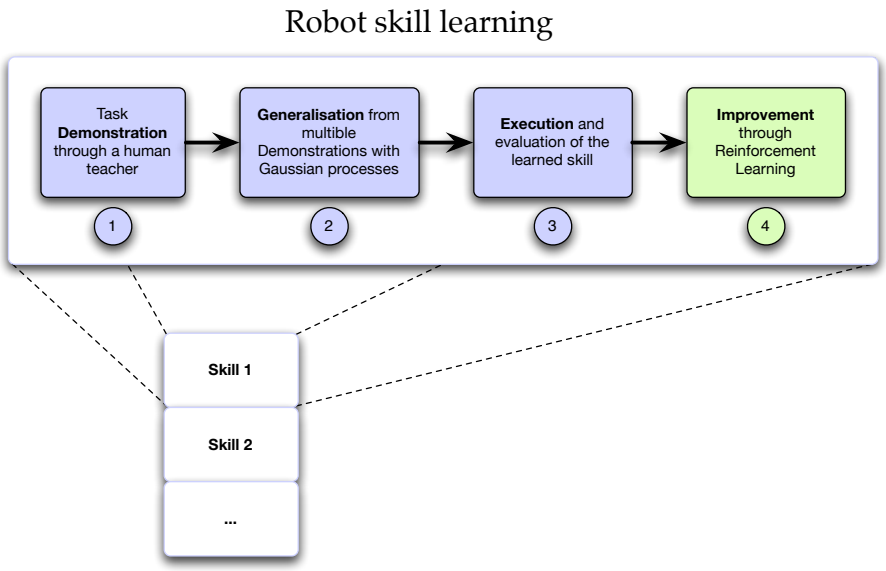


Figure 1.1: This figure illustrates the use of different techniques to build a robot skill database. In steps (1) to (3) a new skill is presented to the robot. Step (4) equips the robot with self improvement capabilities. The overall goal is to reuse already learned skills and adapt or combine them in new situations.

1.2 Learning from Demonstration

Learning from Demonstration (LfD) is a technique that allows a teacher to demonstrate a desired behavior to a learner which is then able to reproduce the behavior. This section will shortly summarize the most important aspects of LfD. A detailed discussion can be found in the *Handbook of Robotics* [Billard et al., 2008, chapter *Robot Programming by Demonstration*] and in the article *A survey of robot learning from demonstration* [Argall et al., 2009]. For a discussion on computational approaches see [Schaal et al., 2003].

1.2.1 Historical Overview

In the last years robots turned from simple preprogrammed tools into highly flexible and complex machines. There is a lot of expert knowledge necessary in both, the robot behavior and the task to fulfill, in order to program a robot. Simulations are used to track positions and velocities of the robot to see possible problems and to predefine actions for all exceptions that might occur. There is also a lot of hand-tuning to improve, for example, the cycle time of a robot arm or to make the robot more energy efficient. This costs a lot of time and money and is one of the reasons why most of the robots are only applicable in industrial environments with very well known processes and predefined work space. Better human-robot interfaces are needed to make robots useable in less structured, uncertain and human daily environments even for laymen. If we look at humans and animals then it seems to be very reasonable to transfer knowledge for solving a problem with the help of demonstration and imitation. This is one reason why Learning from Demonstration had become a major topic in the context of robotics. With Learning from Demonstration a human can show the desired actions directly rather than encoding it in a bunch of coordinates and artificial commands.

The history of Robot Learning from Demonstration (LfD) goes back 20 years to the beginning of the 1980s. It was also referred to as *teach-in*, *guiding* or *play-back*. Basically the robot was moved manually or by teleoperation and the sequence of positions, velocities and orientations were recorded. A set of relevant waypoints was extracted and the reconstruction of the movement was done by interpolation of these points.

Due to noise of sensor data and variability of the human motion it appeared necessary to develop a method to merge multiple demonstrated movements. This is the interface between *Programming by Demonstration* (PbD) and *Machine Learning*. The field of research moved from simple playback to generalization and interpretation of demonstrations. Due to this change the terms *Learning from Demonstration* (LfD), *Learning by Demonstration* (LbD) and *Imitation Learning* are very common in recent years.

1.2.2 Human Robot Interaction

There are several ways how the necessary data can be collected. They can roughly be divided into two groups: *demonstration* and *imitation*. In the first case, the demonstration is performed on the robot itself. The robot's sensors can be used to observe states and actions. This can be done by directly acting on the robot or by remote teleoperation. For more complex robots or movements this can become very difficult and is not always possible. For example, it is nearly impossible to control all joints of a full humanoid robot with a remote control. A more biologically inspired approach is *imitation*, where the robot is not involved in task execution, but observes the teacher. For example a human can show a behavior and the learner records the body movements through a motion capture environment (sensors directly attached to the human or through cameras and other sensors).

The main restriction of multiple demonstrations is the small set of data. Usually there are not more than five to ten demonstrations by the teacher. That means it is essential to make efficient use of these data. There is another problem arising especially in the context of robotics, called the *correspondence problem* [see Alissandrakis et al., 2002]. These are the issues related to the different mechanics of the teacher and the imitator. The movements of the teacher have to be mapped to the embodiment of the robot which is still a non-trivial task. Münch et al. [1994] admitted, that the teacher has to be aware of his role and is responsible to provide a sufficient number of demonstrations considering the mechanics of the robot. Recent research tries to make the learner more and more independent of the instructor's teaching skills.

1.2.3 Robot Skill Encoding

Current approaches to represent a robot skill can be divided broadly in two abstraction layers called *symbolic encoding* and *trajectory encoding*. An encoding at a symbolic level describes a sequence of primitives that are already known and given in advance. In contrast, *trajectory encoding* is focused on the low level part of the task (e.g. positions, velocity, acceleration, torques, current). There is no clear boundary between these two approaches and many different definitions can be found in literature. However *symbolic encoding* uses a much higher level abstraction than *trajectory encoding* to describe a skill/task.

Symbolic encoding can be seen as a high-level form of Learning from Demonstration. Generally it is assumed that the robot is already equipped with a set of low level skills or motor primitives. For example, this could be a simple movement like opening the gripper or also a more complex sequence like *move-to-object* or *grasp-object*. The main focus here is to bring the primitives in a proper execution order or to derive planning capabilities.

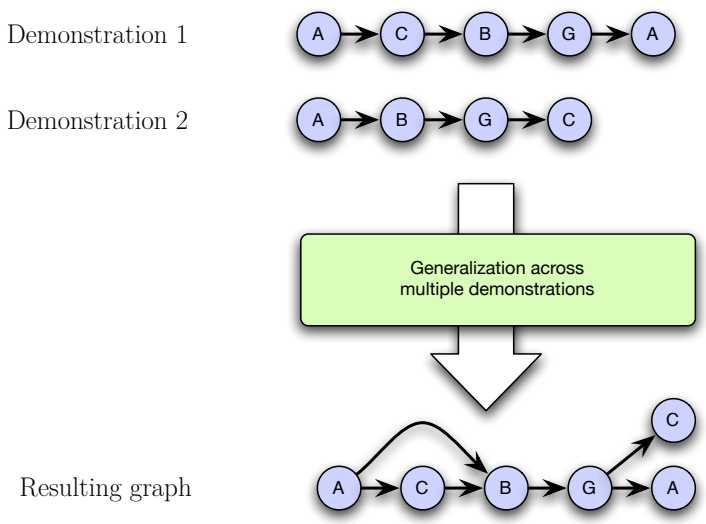


Figure 1.2: The figure shows graph representations of a two demonstrations. The third graph at the bottom is the result after merging *Demonstration 1* and *Demonstration 2*.

In most cases a graph like representation of the task and the environment is created where each state is a node of the graph and actions are direct links between these states (see Figure 1.2). This graph will then be analyzed to extract subgoals and task constraints as in [Ekvall and Kragic, 2006, Münch et al., 1994, Nicolescu and Mataric, 2003, Pardowitz et al., 2007]. Most often this ends in a derivation of a set of if-then-else rules. With a graph based approach it is also straightforward to decompose the task into subgoals and use a hierarchical structure to solve more complex problems. Most often *Hidden Markov Models* (HMM) are used to accomplish this [Hovl et al., 1996]. Symbolic encoding is very promising and a lot of work had been done in the last years. See also [Alissandrakis et al., 2007, Friedrich et al., 1996].

Trajectory encoding tries to approximate the underlying teacher policy (the state-to-action mapping) directly and is less goal oriented than the symbolic approach. This is also very often referred to as *motor primitive* or *action primitive*. Typically regression techniques are used to retrieve a generalized version of these low-level motions (joint positions, dynamics, torques as in Figure 1.3).

Calinon et al. [2006] used *Gaussian Mixture Regression* (GMR) to find a controller for the task reproduction (see also [Calinon, 2007, Calinon and Billard, 2007a,b]). Ude [1993] used *Natural Vector Splines* to approximate robot trajectories, Schaal et al. [2002] successfully demonstrated the use of *Locally Weighted Projection Regression* (LWPR) in a LfD environment (see also [Vijayakumar and Schaal, 2000]).

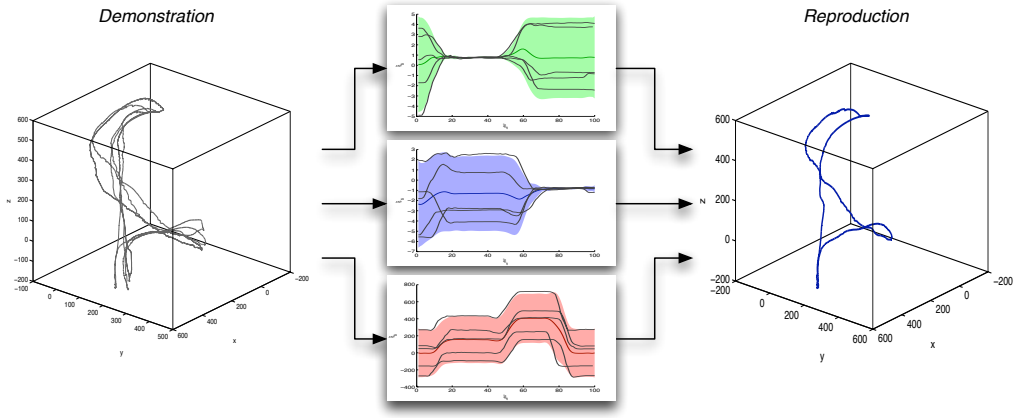


Figure 1.3: Trajectory encoding (middle panel) using task space coordinates

1.2.4 Policy Improvement

Once the robot learned a new skill, the question is how it should behave if it is confronted with a new or unseen context. In early approaches the agent asked the teacher explicitly. A better solution would be to allow the robot to extract the necessary information of the task out of the demonstrations. This so constructed task dependent skill is much more flexible than the context dependent one. Hwang, Choi, and Hong [2006] used genetic algorithms to do this, where most other researchers focus on Reinforcement Learning to improve behavior (e.g. Ijspeert, Nakanishi, and Schaal [2002a,b], Peters, Vijayakumar, and Schaal [2003]).

1.2.5 Summary

Demonstrating can be seen as a common language to transfer information between different platforms. Humans and robots (or other types of systems) can communicate this way. Various robot skills/behaviors can be combined in a sequential or hierarchical way to equip the robot with more complex abilities. Learning approaches can be sped up dramatically compared to pure learning from scratch because the underlying essence of multiple demonstrations can be extracted. This also increases the robot's ability to cope with changes in the environment. It also seems to be very obvious to use Learning from Demonstration if we look at nature where this strategy is very common. Therefore recent developments are biologically inspired and study the imitation abilities of "intelligent" animals and humans.

There are still several open questions: How to map the teacher's movements to the mechanics of an agent? Which is the appropriate internal representation of states and actions? How to extract the correct context of relevant objects in the scene?

Another problem is the accurate observation e.g. vision, tracing, segmentation. What can be done to reduce the dimensionality of the received observations? It has to be distinguished between relevant and irrelevant information. Adaption of a skill to a changed environment is still ongoing research and very difficult. There are several approaches that are more or less successful. There are only very few demonstrations and therefore learning techniques that require a large amount of training data are not appropriate. How can the learner still derive a good policy even if the teacher's performance on the task is very bad?

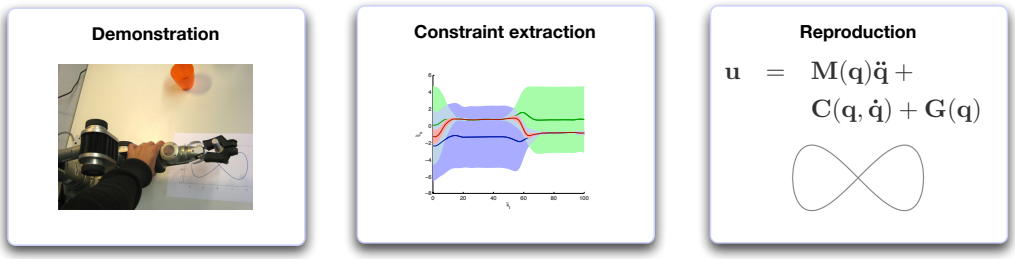


Figure 1.4: Seeing, **Understanding** and **Doing** are the requirements for a robot to learn from demonstration. These properties will be discussed in detail in the rest of this thesis.

1.3 Thesis Outline and Objective

This thesis focuses on *Gaussian process models* and how they can be used in a robot *Learning from Demonstration* framework. Gaussian processes are powerful, probabilistic non-parametric machine learning techniques. We will show, that this approach can be used to extract the key features of a task from multiple demonstrations. This can be used to extract informations about *what* has to be learned. We will show, that the Gaussian process approach can achieve similar results as the *Gaussian Mixture Model/Gaussian Mixture Regression* methods from [Calinon, 2007].

Chapter 2 introduces *Gaussian process models*, a relatively new approach in machine learning context. We will explain the fundamental theory applied to regression problems including a discussion about covariance functions. We also give a short outline of automatic hyperparameter estimation.

Chapter 3 presents the *Learning from Demonstration framework* used in this thesis. We start to extend the basic Gaussian process models in order to make them applicable to robot movement encoding. We explain how constraints can be extracted from demonstrations and how to derive a controller that follows these constraints, even for new situations.

Chapter 4 describes the *experimental setup* we used to evaluate the theoretical results. We first show, that Gaussian processes can be used to derive a controller for a simple reproduction task. In the second experiment the robot has to extract several task constraints and apply them to new situations.

Chapter 5 summarizes the results and proposes some ideas for future projects.

CHAPTER 2

Gaussian Processes

This chapter presents a relatively new approach for regression. Gaussian processes ($\mathcal{GP}$) are generalizations of Gaussian probability distributions. Most of the chapter is inspired by the book of Rasmussen and Williams [2006].

Gaussian processes had also been studied in the field of spatial geostatistics, but there it better known as *kriging*, named after the South African mining engineer Krige (1951). Therefore it is surprising that these methods are relatively new in the context of machine learning.

In his thesis, Rasmussen [1996] compared the performance of Gaussian process regression and other regression techniques. Gibbs [1997] also presents an excellent review on Gaussian process regression in this thesis. Short tutorials on regression with Gaussian processes can be found in the book of Bishop [2006] and Williams [1997].

Gaussian process models are also related to bayesian multilayer perceptron networks as investigated by Neal [1996], followed by MacKay [1997] and Williams [1998].

Section 2.1 gives an introduction to Gaussian processes for regression and compares it to ordinary regression techniques.

Section 2.2 describes the Gaussian process theory and its application to regression problems. Classification is not part of this thesis and the reader is referred to the literature mentioned above.

Section 2.3 explains the most common covariance functions in machine learning and how these functions can be combined to form new ones.

Section 2.4 outlines the necessary steps for automatic hyperparameter estimation and defines a metric to compare different Gaussian process models. The role of the marginal likelihood is studied in more detail.

Section 2.5 is a very short summary of nonlinear conjugate gradient methods and line search.

2.1 Introduction

The goal of supervised learning methods is to learn a mapping[1] from some input space to an output space given a data set of observations (the training data set). In general, the data set $\mathcal{D}$ of n observations of a process will be denoted by $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}) | i = 1, \ldots, n\}$. The input of the process is $\boldsymbol{x}$ and the output (or target) is y. Usually the input $\boldsymbol{x}$ is a vector of real numbers. For example, this can be features of an image or the position of joint angels of a robot arm. In most cases the output y is a scalar. For a regression problem y is continuous and discrete for a classification problem (this thesis will only cover the regression problem[2]). Mostly the output values in the data set are not the true value of the underlying function, but noise corrupted versions of them.

Given this training data set we want to make predictions for new input data we have not seen in the training set. An example illustration of this problem can be seen in Figure 2.1, where the blue circles are given and we want to predict the function value at $x_* = 0.75$ (red diamond). Typically, parametric models are used for this task. In the parametric model approach the information given by the training data is encoded in the parameters. After a so-called training phase the training data can be discarded. The work with a parametric model requires making an assumption about the underlying function. For example we can expect the underlying function to be linear (e.g. $f(x) = mx + c$) and then use *least squares regression* or *Bayesian maximum likelihood estimate* to fit the function to the training data. It is obviously a great problem that a specific class of functions (e.g. linear functions, polynomial functions, etc.) has to be chosen in advance. If the class of functions is much simpler than the target function then it will not be able to model it and poor predictions will be the result. We will have *high bias*[3] on the training data. Whereas if we choose a more complex class then *high variance/overfitting* is much more likely. As a result also the noise in the training data will be approximated and the performance on new test points will be poor.

Gaussian processes ($\mathcal{GP}$s) differ from this approach. A Gaussian process is a generalization of a *multivariate Gaussian distribution* to infinitely many variables. One point of view is, that a Gaussian process specifies a prior probability over every possible function. Functions with desired properties, functions that are more likely

1 For historical reasons, this is often called the *hypothesis*.

2 More information about the application of Gaussian processes to classification problems can be found in MacKay [1997], Rasmussen and Williams [2006], Williams and Barber [1998]

3 This is also very often denoted as *underfitting* the data.

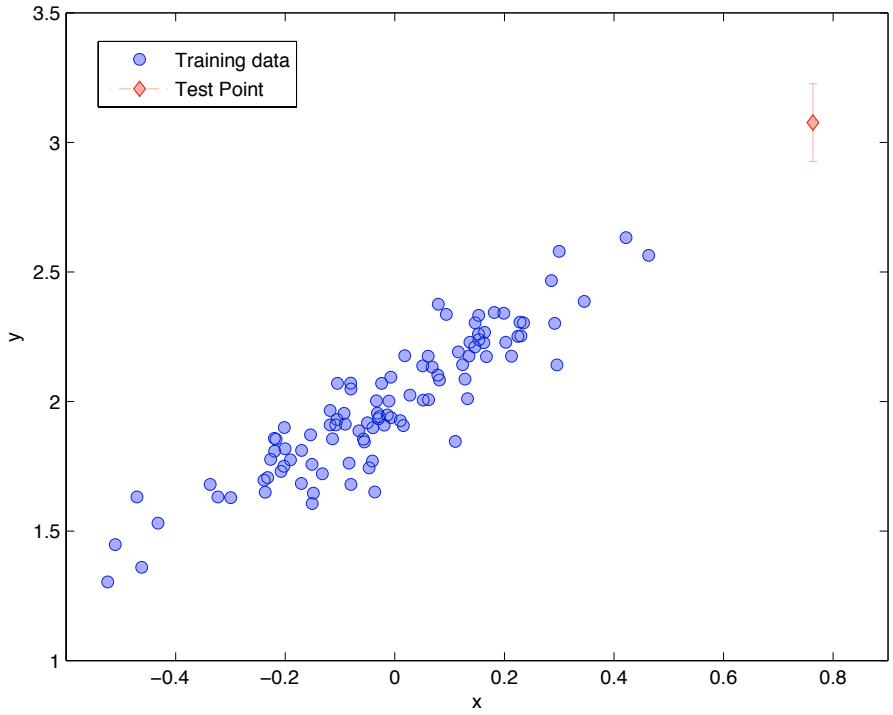
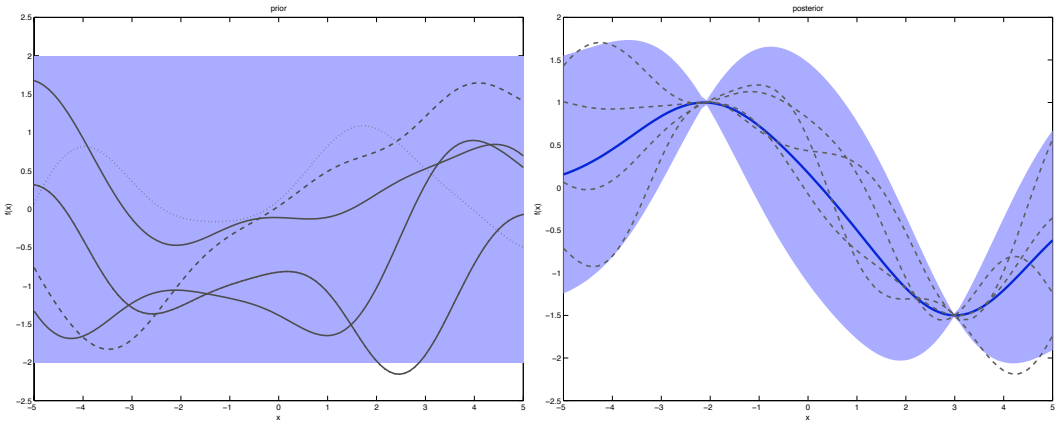


Figure 2.1: A simple example of a regression problem. Given the noisy training data (cicles) we want to make a prediction for the output of $x_* = 0.75$ (diamond).

to represent the true underlying function are given higher probabilities. There is no more a restriction to a single class. The theory behind Gaussian processes provides us with the mathematical tools to deal with these infinite dimensions.

The five functions in Figure 2.2a had been randomly drawn from the *prior* distributions over function. All of these functions have one characteristic in common: they are very smooth. This is because the underlying Gaussian process used in this example gives higher prior probabilities to functions with this property. The prior probabilities define what kind of functions are expected to be observed, before seeing any data and are chosen in advance. In addition to the smoothness property it was assumed that the mean value at a specific input x is zero over all functions. Note that not the overall mean of one single function is 0, but the values $f(x)$ of many functions at a fixed point x.

In general, functions sampled from a random process are not of very high interest. In Figure 2.2b two training data points $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)})\}$ have been added to the process. The dotted lines are functions sampled from the new distribution. Now the random functions match the training point values exactly. Putting



(a) prior **(b)** posterior

Figure 2.2: The left panel shows five randomly sampled functions from the prior distribution over functions. The right panel illustrates the new posterior distribution, after two training points has been added to the process. The solid line marks the mean and the dotted lines are again functions randomly sampled from the distribution. The shaded region denotes twice the standard deviation for the inputs in both plots.

higher probabilities to this kind of functions causes this effect. But they still differ at positions with no data points given. It can also be noticed that the variance between the functions is higher the farther they are away from the training points. This "uncertainty" is denoted by the shaded region and is two times the pointwise standard deviation. It is small very close to the points and grows with distance. The solid line represents the mean over all functions. The new distribution is called the *posterior* over functions and is again gaussian.

As indicated before, the prior belief over functions is very important. It is not restricted to the smoothness property of a function, but also a periodic trend and observation noise can be introduced. This is done by the *covariance function* of the Gaussian process, which is of major interest. Different examples of covariance functions and their behavior will be discussed in detail in section 2.3.

Unlike in the parametric approach, Gaussian processes have no parameters that are adjusted during the training phase. Therefore it is called a *non-parametric* model. That does not mean that a *GP* is completely free form. There are parameters like the mean function and the covariance function, but they are not modified as new training data is added. This is a very elegant way to introduce prior information, as it will be explained later.

Many machine learning methods are special cases of Gaussian processes or they are strongly related. For example Neal [1996, chap. 2] described how Gaussian

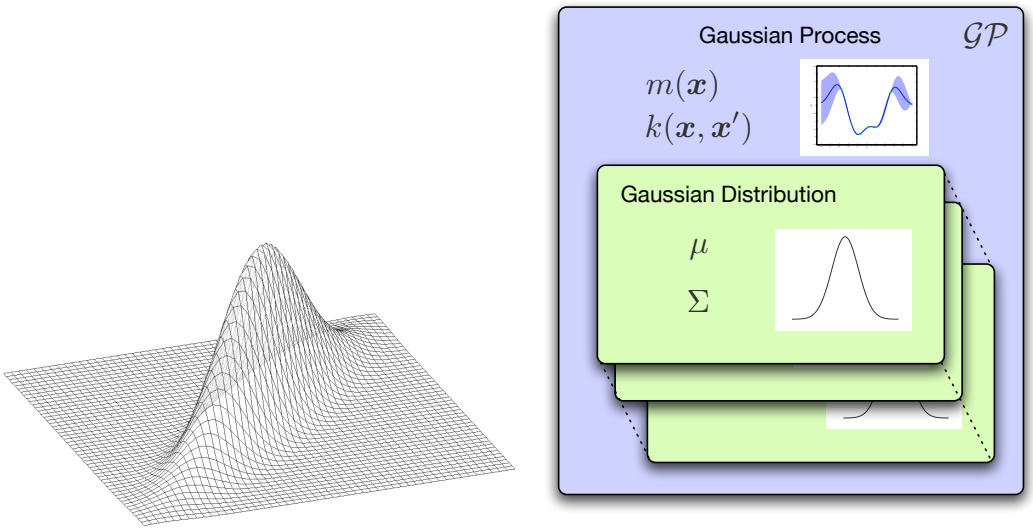
processes could be interpreted as a *neural network* prior in the limit of an infinitely large hidden layer. The same results can be obtained from [Bishop, 2006, chap. 6] and [Rasmussen and Williams, 2006, chap. 6]. There are many parallels to *support vector machines* and *spline methods*, but the interested reader is referred to the literature for more detailed information.

2.2 Gaussian Process Models

This section will give a formal definition of a Gaussian process and shows how Gaussian processes can be used for regression problems. It will explain how training data is introduced and how inference is done in the noise free case and also with noisy observations.

Definition 2.2.1. *A Gaussian process ($\mathcal{GP}$) is a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions.*

That means a Gaussian process generates data such that any finite subset of it follows a multivariate Gaussian distribution.



(a) Gaussian distribution **(b)** Gaussian process

Figure 2.3: Panel (a) shows a Gaussian distribution with zero mean and covariance Matrix $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 1 & 0.9 \\ 0.9 & 1 \end{smallmatrix}\right]$. In panel (b), a Gaussian process as a generalization of Gaussian distributions.

A Gaussian process is fully specified by its *mean* function $m(\boldsymbol{x})$ and *covariance* function $k(\boldsymbol{x}, \boldsymbol{x}')$[1], in contrast to a Gaussian distribution specified by a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$ with $\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Furthermore, a Gaussian process is a probability distribution over functions, whereas a Gaussian distribution is a distribution over vectors. Therefore a $\mathcal{GP}$ is a generalization of a Gaussian distribution to infinite dimensionality. The notation

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')) \tag{2.1}$$

will refer to a real process $f(\boldsymbol{x})$ distributed as a $\mathcal{GP}$ with mean and covariance function as

$$m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] \tag{2.2a}$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))] \tag{2.2b}$$
$$= \mathrm{cov}[f(\boldsymbol{x}), f(\boldsymbol{x}')] \tag{2.2c}$$

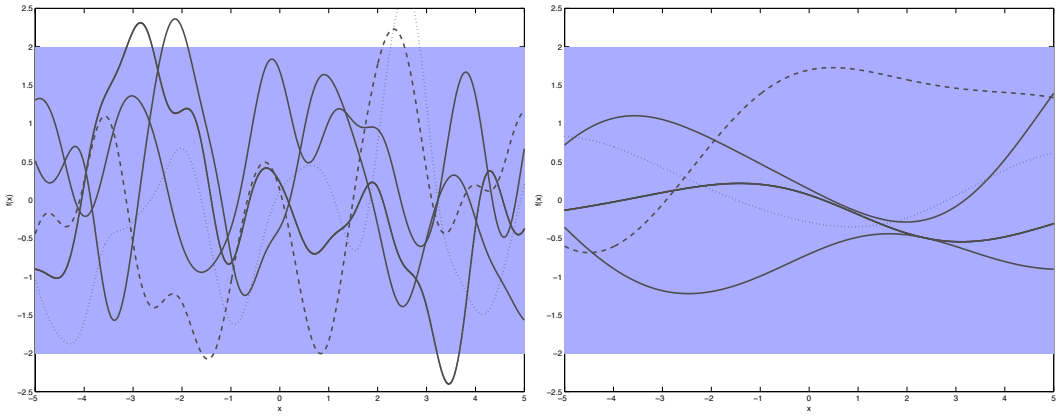
respectively. Usually a constant mean function of $m(\boldsymbol{x}) = 0$ is assumed. This is mostly done for simplicity of notation but also reasonable if there is no prior knowledge about the mean available.[2] To agree with the data the original problem has to be transformed (e.g. subtracting the mean out of it).[3] See [Rasmussen and Williams, 2006] for the formulas with respect to nonzero mean functions.

As a simple example for a Gaussian process a widely used covariance function is chosen, given by the exponential of a quadratic form, namely the *Radial Basis Function* or *Squared Exponential* (SE) to give

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \exp\left(-\frac{\|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2}{2l^2}\right), \tag{2.3}$$

with l defining the *characteristic length scale*[4]. This and other covariance functions will be discussed in more detail in section 2.3, but for now, note that the function takes two arguments from the input space and defines the covariance between its function values in the output space (see also Figure 2.6). The SE has its maximum of 1 where both, $\boldsymbol{x}_p$ and $\boldsymbol{x}_q$ are equal, meaning $f(\boldsymbol{x}_p)$ and $f(\boldsymbol{x}_q)$ are perfectly correlated, and decreases rapidly with growing distance (in the input space) of the two arguments (then $k(\boldsymbol{x}_p, \boldsymbol{x}_q) \approx 0$). Informally spoken, this makes the function look smooth, because neighbors are similar. The characteristic length scale parameterizes the slope of decrease. With higher length scale a point $\boldsymbol{x}_i$ can still "see" other points that are farther away.

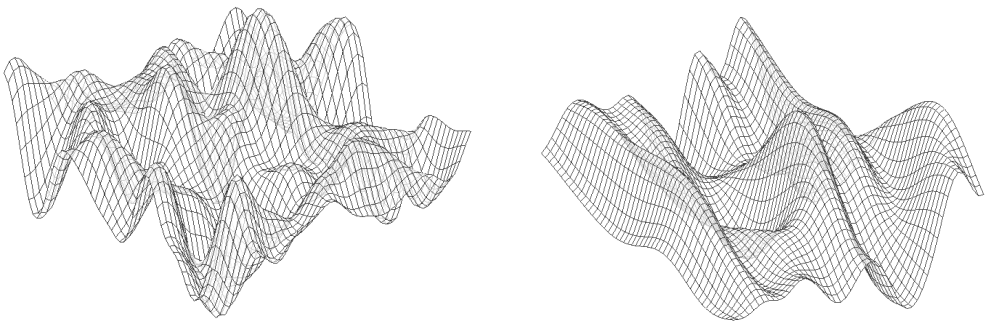
1 Also called the second-order statistics.
2 This is equivalent to choosing the weights of a parametric model to be zero mean at the beginning.
3 It is also possible to make the estimation of the mean function part of the model.
4 In literature also very often called the *bandwidth parameter* and denoted by τ.



(a) small length scale **(b)** large length scale

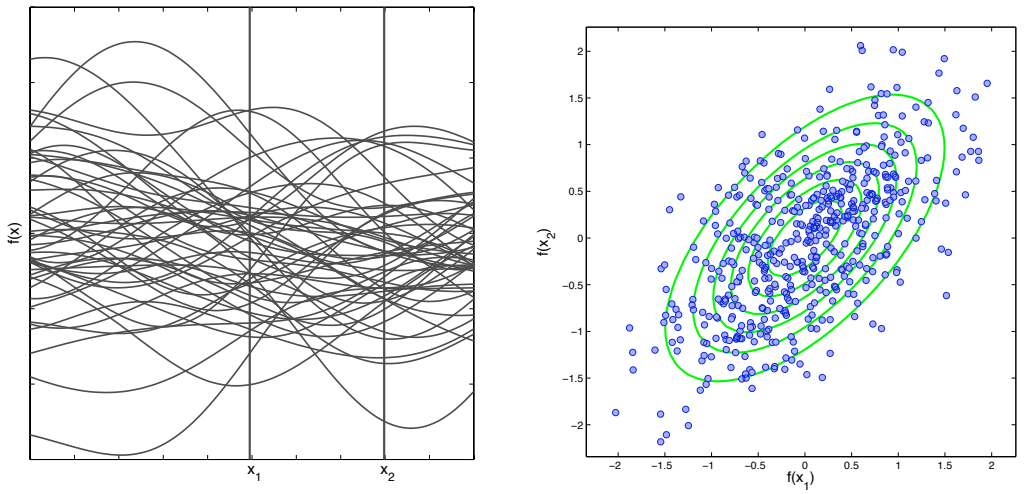
Figure 2.4: Random samples from a Gaussian process prior with squared exponential covariance function. The left panel shows a small SE with small length scale value (0.5), whereas the right panel has a much larger value (3).

The effect of different length scales can be seen in Figure 2.4. The five functions had been randomly sampled from a Gaussian process prior with the squared exponential covariance function. A small length scale of 0.5 (left) is compared to a larger length scale of 3 (right). The function values in panel (b) only vary slowly. The larger length scale causes the correlation of function values of distant points.



(a) same length scale **(b)** different length scale

Figure 2.5: A plot of a sample from prior distribution in a 2-d input space. On the left, the length scale parameter for each dimension is the same. On the right, the length scale in one dimension is higher than in the other.



(a) prior **(b)** empirical evaluation

Figure 2.6: Empirical evaluation of the covariance function: panel (a) shows 50 functions randomly sampled from the Gaussian prior distribution with zero mean and covariance function k given by a squared exponential covariance function. On panel (b), 500 function values of x_1 and x_2 are plotted against each other together with the contour plot for the empirical distribution of the data. It has a zero mean and a covariance of $\mathrm{cov}[\{(f(x_1)^{(i)}, f(x_2)^{(i)})\}_{i=1}^{500}] \approx k(x_1, x_2)$.

It is also possible to use a different length scale for each dimension of the input.[1] Figure 2.5 is a plot of a sample from prior distribution in a 2-d input space. In panel (a) the length scale parameter for each dimension is the same, whereas in panel (b) the length scale in one dimension is higher than in the other. Note that the function varies in one dimension much more frequently than in the other.

2.2.1 Regression

Random samples from the Gaussian process prior are useful to see what kind of distribution over functions is implied by the covariance function, but these are not of primary interest. The goal of regression is to make predictions of target variables for new inputs, given a set of training data.

As previously defined the observed target values y are modeled as the true function values $f := f(\boldsymbol{x})$ and some observation noise given by

$$y = f + \epsilon, \tag{2.4}$$

1 It will be shown later how this can be used to do *Automatic Relevance Detection* (ARD).

where ϵ is a random noise variable that is independent, identically distributed (i.i.d.) for each observation. Here a Gaussian distribution for the noise is assumed with zero mean and variance σ_n^2:

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2). \tag{2.5}$$

The training data set comprising n input points together with the corresponding observations is denoted by $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}) | i = 1, \ldots, n\}$. In the noise-free case the real function value is known and therefore $\{(\boldsymbol{x}^{(i)}, f^{(i)}) | i = 1, \ldots, n\}$ is accessible. For now, the latter case is assumed and observation noise will be considered in the next step. To simplify notation all training inputs $\{\boldsymbol{x}^{(i)}\}_{i=1}^{n}$ are collected in a so called *design matrix* $\boldsymbol{X}$ of size $D \times n$, where D is the dimension of the input space $\mathcal{X}$. The same way, the matrix of test inputs $\{\boldsymbol{x}_*^{(i)}\}_{i=1}^{n_*}$ is defined as $\boldsymbol{X}_*$.

The key in Gaussian processes is to consider the training data $\boldsymbol{f} = \{f^{(i)}\}_{i=1}^{n}$[1] and the new testing points (prediction values) $\boldsymbol{f}_* = \{f_*^{(i)}\}_{i=1}^{n_*}$ as a sample from the same multivariate Gaussian distribution as

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \\ \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) \end{bmatrix}\right). \tag{2.6}$$

Assuming there are n training points and n_* test points, then the matrix $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$ is the $n \times n$ covariance matrix resulting from $k(\boldsymbol{X}, \boldsymbol{X})$. $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*)$ is the $n \times n_*$ matrix of covariances between training and test points and $\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})$, $\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*)$ are analogous. For example:

$$\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & k(\boldsymbol{x}_n, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix}. \tag{2.7}$$

Since $\boldsymbol{f}$ and $\boldsymbol{f}_*$ are jointly distributed, it is possible to condition the prior (Equation 2.6) on the observations and ask how likely predictions for the $\boldsymbol{f}_*$ are. Fortunately Gaussian distributions have nice properties to do this:

$$\boldsymbol{f}_* | \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{f} \sim \mathcal{N}(\bar{\boldsymbol{f}}_*, \operatorname{cov}[\boldsymbol{f}_*]), \quad \text{with} \tag{2.8a}$$

$$\bar{\boldsymbol{f}}_* = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}) \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})^{-1} \boldsymbol{f} \tag{2.8b}$$

$$\operatorname{cov}[\boldsymbol{f}_*] = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}) \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})^{-1} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*). \tag{2.8c}$$

1 Note that this is only true for the noise-free observations. In the noisy observation case $\boldsymbol{y} = \{f^{(i)} + \epsilon^{(i)}\}_{i=1}^{n}$ will be used.

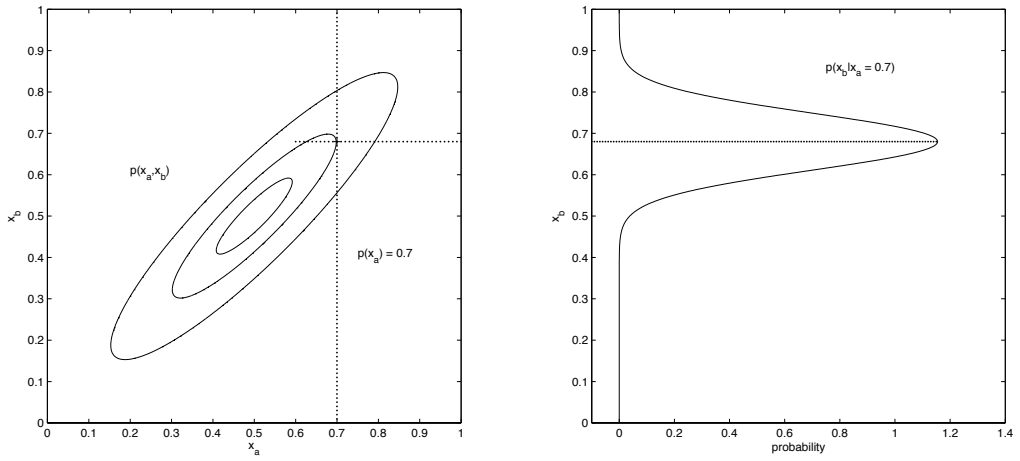


Figure 2.7: Illustration of a conditional distribution: The left panel shows a contour plot of a joint Gaussian distribution $[x_a, x_b]^T \sim \mathcal{N}(\mu, \Sigma)$. The value of the training point x_a is observed. The vertical dotted line denotes the conditioning on this value. The predictive distribution for x_b can be seen on the right.

Note that the joint posterior distribution is again a Gaussian distribution with mean $\bar{\boldsymbol{f}}_*$ and covariance $\text{cov}[\boldsymbol{f}_*]$, both depending on $\boldsymbol{X}_*$ (See section A.3 for a detailed explanation and proof).

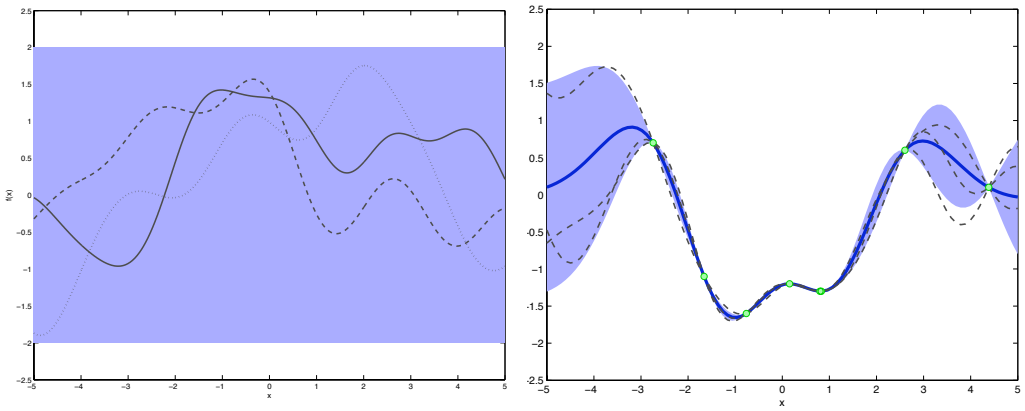
An application is illustrated in Figure 2.8. The prior distribution (left) was conditioned on a data set with seven observations (right) as in Equation 2.8a. The blue line and the shaded region are denoting the mean and the two times pointwise standard deviation. Functions sampled from this posterior are forced to pass the training points exactly as the deviation on these is zero.

It is straight forward to redefine Equation 2.8b and Equation 2.8c to the case where only noisy observations $y = f(\boldsymbol{x}) + \epsilon$ are given. For independent, identically distributed noise[1] as assumed in Equation 2.5 a noise term can be added to the covariance function as

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) + \sigma_n^2 \delta_{pq}, \tag{2.9}$$

with δ_{pq} as the *Kronecker delta*[2]. As a result, a diagonal matrix with noise $\sigma_n^2 \boldsymbol{I}$ is added to all $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$. The joint distribution of the observations and the test points

1 It is also possible to consider much more complicated noise assumptions.
2 Kronecker delta δ_{pq} is 1 if $p = q$ and 0 if $p \neq q$.



(a) prior **(b)** posterior

Figure 2.8: Panel (a) shows functions drawn from the prior distribution from a Gaussian Process with squared exponential covariance function. The mean over the function values is zero. The posterior distribution after conditioning the prior on seven training points can be seen in panel (b). Here the blue line is the mean over function values. In both plots, the shaded region denotes two times the pointwise standard deviation.

under the prior from Equation 2.6 is now

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2 \boldsymbol{I} & \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*) \\ \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*) \end{bmatrix}\right). \tag{2.10}$$

Also the predictive distribution changes only slightly giving **the key result of Gaussian process theory**:

$$\boldsymbol{f}_* | \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}(\bar{\boldsymbol{f}}_*, \operatorname{cov}[\boldsymbol{f}_*]), \quad \text{with} \tag{2.11a}$$

$$\bar{\boldsymbol{f}}_* = \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{y} \tag{2.11b}$$

$$\operatorname{cov}[\boldsymbol{f}_*] = \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*). \tag{2.11c}$$

In terms of the Gaussian process definition we can write

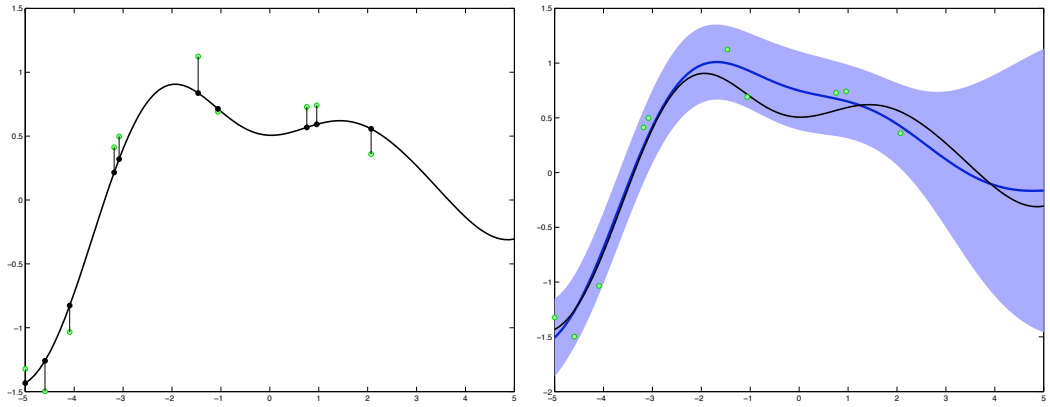
$$\begin{aligned} f(\boldsymbol{x})|\boldsymbol{X},\boldsymbol{y} \sim \mathcal{GP}(m_{post}(\boldsymbol{x}) &= k(\boldsymbol{x},\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y}, \\ k_{post}(\boldsymbol{x},\boldsymbol{x}') &= k(\boldsymbol{x},\boldsymbol{x}') - k(\boldsymbol{x},\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}k(\boldsymbol{X},\boldsymbol{x}')) \end{aligned} \tag{2.12}$$

for the Gaussian process posterior. Note that the covariance function does not depend on the observations $\boldsymbol{y}$, but only on the inputs. The first term is the *prior variance* giving the variance before any training data has been given. The second (positive) term includes all information that $\boldsymbol{X}$ and the noise term introduced to the

system. In contrast, the mean function is linearly dependent on the observations, called a *linear predictor*.

Algorithm 2.1 Gaussian Process Regression Algorithm as described in [Rasmussen and Williams, 2006]

1: **input:** $\boldsymbol{X}$ (inputs), $\boldsymbol{y}$ (observations), $k(\cdot,\cdot)$ (covariance function), σ_n^2 (noise level), $\boldsymbol{X}_*$ (test points)
2: $\boldsymbol{L}$ = CHOLESKY($\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})+ \sigma_n^2\boldsymbol{I}$)
3: $\boldsymbol{\alpha}$ = $(\boldsymbol{L}^T)^{-1}(\boldsymbol{L}^{-1}\boldsymbol{y})$
4: $\bar{\boldsymbol{f}}_* = \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*)^T\boldsymbol{\alpha}$
5: $\boldsymbol{v} = \boldsymbol{L}^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*)$
6: $\mathbb{V}[\boldsymbol{f}_*] = \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*) - \boldsymbol{v}^T\boldsymbol{v}$
7: $\log p(\boldsymbol{y}|\boldsymbol{X}) = -\frac{1}{2}\boldsymbol{y}^T\boldsymbol{\alpha} - \sum_i \log \boldsymbol{L}_{ii} - \frac{n}{2}\log 2\pi$
8: **return:** $\bar{\boldsymbol{f}}_*$ (mean), $\mathbb{V}[\boldsymbol{f}_*]$ (variance), $\log p(\boldsymbol{y}|\boldsymbol{X})$ (log marginal likelihood)



(a) sampling **(b)** regression

Figure 2.9: Panel (a) shows the generation of noisy training data. The black line is a latent function and the black points on this line are the $f(\boldsymbol{x}^{(i)})$ obtained by evaluating the function at a set of 9 input values $\boldsymbol{x}^{(i)}$. The green dots are the observations $y^{(i)}$ obtained by adding Gaussian noise (vertical lines) to the function values. In panel (b) is an illustration of Gaussian process regression together with the training points (green) and the underlying latent function (black) from panel (a). The blue line is the mean of the Gaussian process predictive distribution. The shaded region is two times the pointwise standard deviation.

2.3 Covariance Functions

The previous section showed how to derive the predictive distribution of a Gaussian process when the covariance function is given. This section will describe different covariance functions and methods to combine them. This thesis will only cover the discussion of covariance functions of a real valued vector space as input. For other input spaces (e.g. strings) and a much more detailed description of covariance functions in general see [Bishop, 2006, chapter 6] and [Rasmussen and Williams, 2006, chapter 4]. Nearly all covariance functions have open parameters to control the behavior. Methods to optimize these parameters will be discussed in section 2.4.

As mentioned before, the choice of the covariance function is very critical for the Gaussian process model. It describes the assumptions about the latent function to learn and therefore is also referred to as the prior. If these assumptions are wrong (as a result of an inappropriate covariance function), then the Gaussian process predictor will not be able to make correct inference. An example can be seen in Figure 2.10.

Until now, only the squared exponential covariance function had been used, but

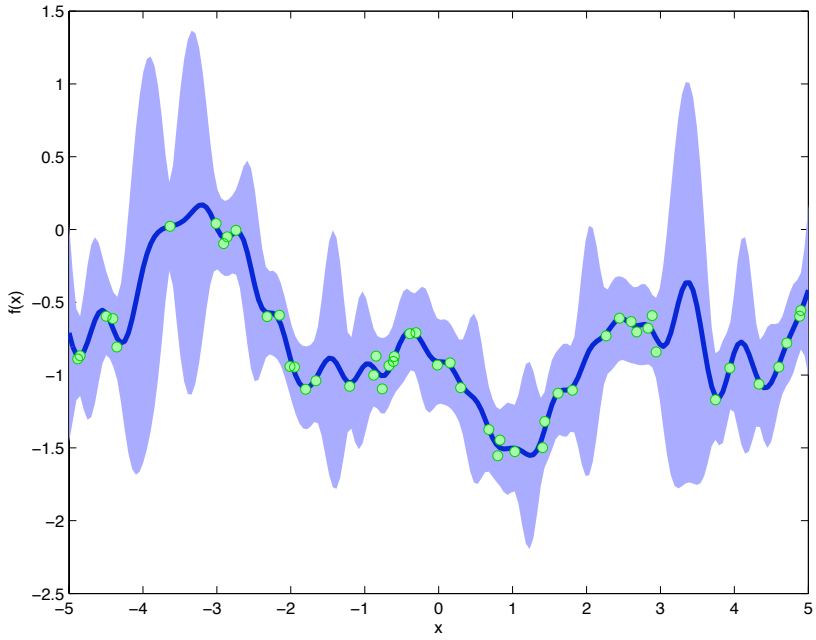


Figure 2.10: A Gaussian process with squared exponential covariance function: $k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \sigma_f^2 \exp(-(\|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2)/(2l^2)) + \sigma_n^2 \delta_{pq}$ with length-scale l^2, signal variance σ_f^2 and noise term chosen randomly. The $\mathcal{GP}$ perform bad on the given data set. Possible problems are a) a too short length scale and b) an underestimated noise level. See section 2.4 for automatic hyperparameter estimation and Figure 2.21 for the performance of the estimated hyperparameters on this example.

there are many other possibilities. In order to generate a valid covariance matrix[1], there are some constraints for the choice of a covariance function. From the definition of the covariance matrix and also the literature can be seen that the function must be symmetric in the arguments, so that $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$. Another necessary condition is that the covariance matrix must be positive semidefinite for all possible choices of the input set. Formally, let $\boldsymbol{K}$ be the matrix whose entries K_{ij} are given by $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$, then the matrix is called *positive semidefinite* if $\boldsymbol{y}^T \boldsymbol{K} \boldsymbol{y} \geq 0$ for all $\boldsymbol{y} \in \mathcal{X}$. The matrix is *positive definite* if $\boldsymbol{y}^T \boldsymbol{K} \boldsymbol{y} > 0$ for all $\boldsymbol{y} \in \mathcal{X} \setminus \{0\}$.

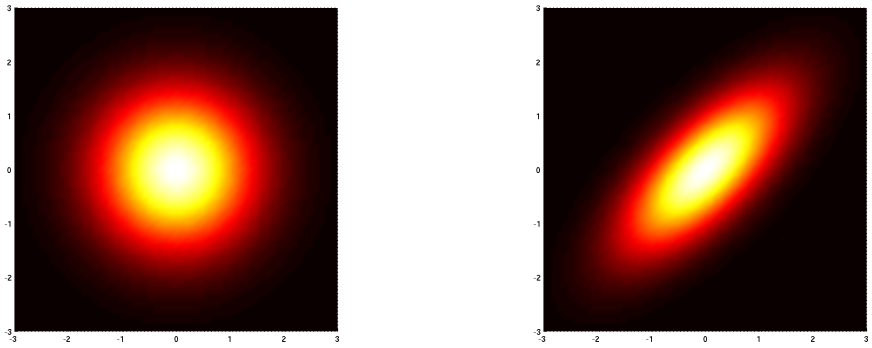
A covariance function is called *stationary* if it is invariant to translations in the input space (see also Figure 2.18 for an example of a stationary and non-stationary covariance matrix). Formally, $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x} + \boldsymbol{t}, \boldsymbol{x}' + \boldsymbol{t})$ for any $\boldsymbol{t} \in \mathcal{X}$. If the covariance function is stationary and the mean function is constant, the Gaussian

1 The matrix whose elements are given by $k(x_i, x_i)$ is most often called the *Gram matrix* in kernel literature.

process is called stationary[1]. The set of stationary covariance functions includes the two subsets of *isotropic* and *anisotropic* functions. Isotropic covariance functions are functions only of the Euclidean distance $\|\boldsymbol{x} - \boldsymbol{x}'\|$. Anisotropic covariance function are functions of a norm $\|\boldsymbol{x} - \boldsymbol{x}'\|_{\boldsymbol{M}}^2 = (\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{M}^{-1} (\boldsymbol{x} - \boldsymbol{x}')$ for some positive semidefinite matrix M. The iso-correlation contours for isotropic functions are spherical, whereas the contours for an anisotropic function can also be elliptical (see Figure 2.11).

In general it is not easy to determine if a function is a valid covariance function. It is a common approach to take simple covariance functions and combine them to a more complex function.

Given valid covariance functions $k_1(\boldsymbol{x}, \boldsymbol{x}')$ and $k_2(\boldsymbol{x}, \boldsymbol{x}')$, the following will also be



(a) isotropic **(b)** anisotropic

Figure 2.11: Contours of iso-correlation. Panels (a) and (b) show the isotropic and anisotropic covariance functions respectively. The isotropic has (always) spherical correlation contours, whereas contours of the anisotropic in this example are elliptic.

1 Also sometimes called *homogeneous*

valid covariance functions:

$$k(\boldsymbol{x}, \boldsymbol{x}') = c \cdot k_1(\boldsymbol{x}, \boldsymbol{x}') \tag{2.13a}$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}') \tag{2.13b}$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') \cdot k_2(\boldsymbol{x}, \boldsymbol{x}') \tag{2.13c}$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x}) \cdot k_1(\boldsymbol{x}, \boldsymbol{x}') \cdot f(\boldsymbol{x}') \tag{2.13d}$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = q(k_1(\boldsymbol{x}, \boldsymbol{x}')) \tag{2.13e}$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(k_1(\boldsymbol{x}, \boldsymbol{x}')) \tag{2.13f}$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}' \tag{2.13g}$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = k_3(\psi(\boldsymbol{x}), \psi(\boldsymbol{x}')) \tag{2.13h}$$

where $c > 0$, $f(\cdot)$ can be any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{A}$ is a symmetric positive semidefinite matrix, $\psi(\cdot)$ maps $\boldsymbol{x}$ to a space $\mathbb{R}^M$ and $k_3(\cdot, \cdot)$ is a valid covariance function in $\mathbb{R}^M$. See Taylor and Cristianini [2004], Bishop [2006] and Rasmussen and Williams [2006] for a detailed discussion about covariance/kernel functions.

The next part of this section will give some examples of covariance functions. As mentioned before, stationary covariance functions only consider the distance between two points in the input space. Most often a function is used that causes the covariance to decay with distance. A parameter $l > 0$ called *characteristic length scale* is used to control this decay rate and will appear in the form

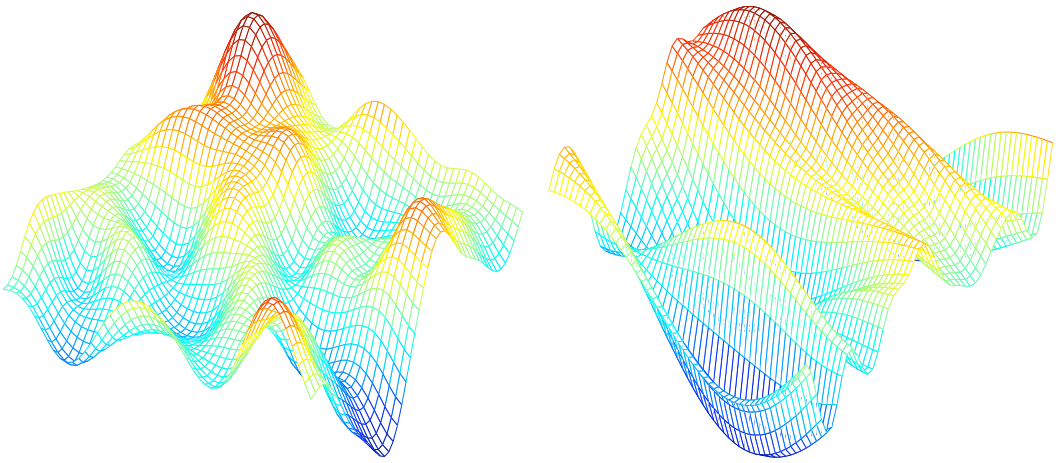
$$\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|}{l}. \tag{2.14}$$

This isotropic form can be modified to a more general anisotropic adaptation with

$$\|\boldsymbol{x} - \boldsymbol{x}\|_{\boldsymbol{M}}^2 := (\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{M}^{-1} (\boldsymbol{x} - \boldsymbol{x}'). \tag{2.15}$$

Setting $\boldsymbol{M} = l\boldsymbol{I}$ causes the characteristic length scale to be equal in all directions and is therefore equivalent to the isotropic formulation. Typically this assumption about the underlying function is too optimistic and not all input dimensions have the same local behavior. An individual length scale can be chosen for every dimension with $\boldsymbol{M} = \text{diag}(\boldsymbol{l})$ and $\boldsymbol{l} = [l_1, \ldots, l_n]^T$. If one length scale parameter becomes very large, the covariance will become nearly independent of that input. In other words, the length scale is negative proportional to the information gain of an input dimension. Therefore such a covariance function can be used to perform *automatic relevance determination* (ARD) as described in [Neal, 1996], evaluating the posterior distribution over the parameters in a Bayesian inference setting. A more complex parameterization is given by $\boldsymbol{M} = \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T + \text{diag}(\boldsymbol{l})$. This is called the *factor analysis distance*, where $\boldsymbol{\Lambda}$ is a matrix with low rank.

The next part of this section will introduce some covariance functions that have very



(a) equal length scale **(b)** individual length scale

Figure 2.12: The squared exponential covariance function for a two-dimensional input space. In panel (a) with the same length scale $l = 5$ for both input dimensions, in panel (b) with $\boldsymbol{l} = [4, 20]^T$.

useful properties for the usage in the context of machine learning. Most of them are based on [Rasmussen and Williams, 2006, chapter 4]. All of these can be combined by the rules of Equation 2.13a - 2.13h to generate more complex functions.

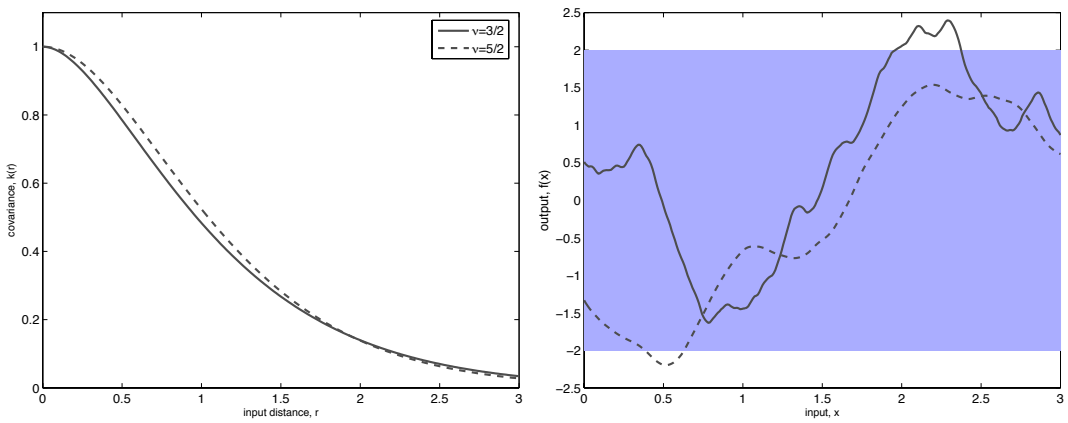
2.3.1 Examples of Covariance Functions

The Matérn Covariance Function

The *Matérn* covariance function is very popular in spatial geo-statistics and given by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \|\boldsymbol{x} - \boldsymbol{x}'\|_M \right)^{\nu} K_\nu \left(\sqrt{2\nu} \|\boldsymbol{x} - \boldsymbol{x}'\|_M \right), \tag{2.16}$$

where ν is a positive parameter, Γ is the gamma function and K_ν is a modified Bessel function. For $\nu \to \infty$ the Matérn converges to the squared exponential covariance function. Rasmussen and Williams [2006] mentioned that for most interesting cases in machine learning the both Matérn functions with $\nu = 3/2$ and $\nu = 5/2$ should be sufficient. As a desired property of these parameters, the Matérn covariance



(a) covariance function **(b)** random samples

Figure 2.13: The Matérn covariance function: panel (a): the shape of covariance functions with $\nu = 3/2$, $\nu = 5/2$ and $l = 1$. In panel (b): random functions drawn from Gaussian processes with the same parameters as in panel (a).

function becomes very simple:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \left(1 + \sqrt{3}\|\boldsymbol{x} - \boldsymbol{x}'\|_M\right) \exp\left(-\sqrt{3}\|\boldsymbol{x} - \boldsymbol{x}'\|_M\right) \quad ; \nu = \frac{3}{2}, \tag{2.17}$$

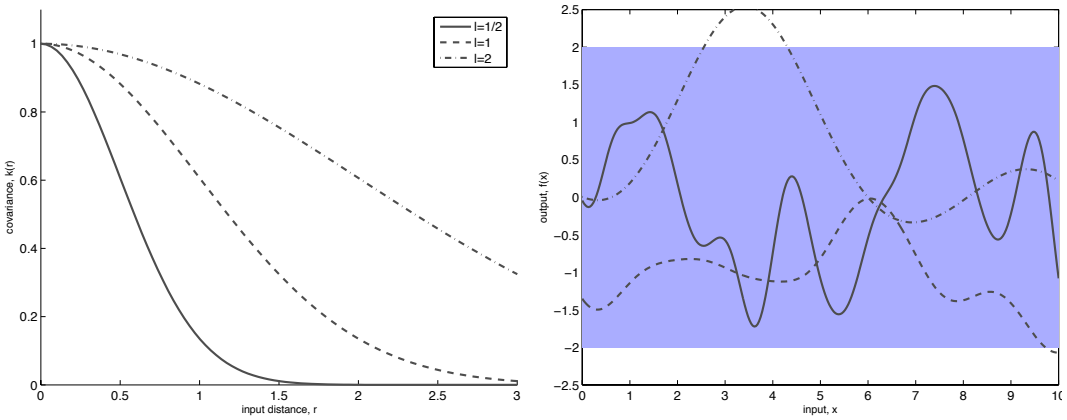
$$k(\boldsymbol{x}, \boldsymbol{x}') = \left(1 + \sqrt{5}\|\boldsymbol{x} - \boldsymbol{x}'\|_M + \frac{5}{3}\|\boldsymbol{x} - \boldsymbol{x}'\|_M^2\right) \exp\left(-\sqrt{5}\|\boldsymbol{x} - \boldsymbol{x}'\|_M\right) \quad ; \nu = \frac{5}{2}. \tag{2.18}$$

The Squared Exponential Covariance Function

The *squared exponential* (SE) covariance function had already been introduced in section 2.2 and is also very often referred to as the *Gaussian* covariance function given by

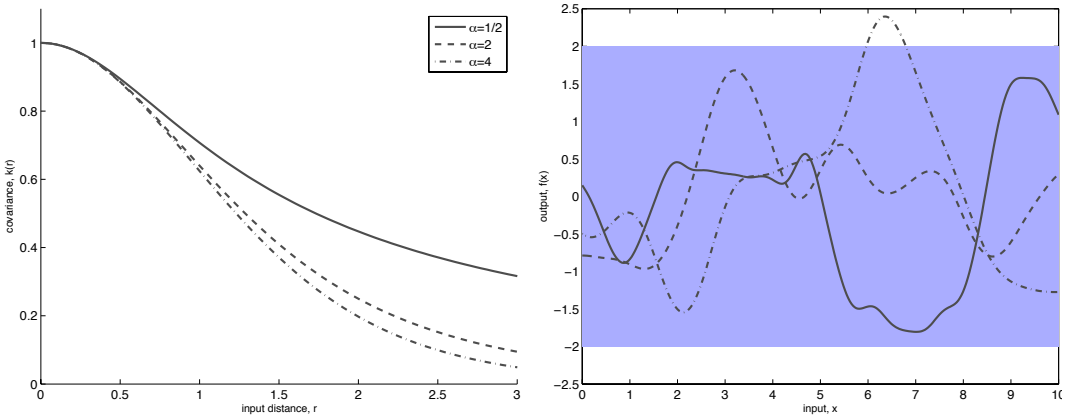
$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}'\|_M^2\right). \tag{2.19}$$

The corresponding Gaussian process will give higher priors to ”smooth” functions. This could be one reason, why the squared exponential is very popular in machine learning literature, even if this assumption about the latent function is very unrealistic.



(a) covariance function **(b)** random samples

Figure 2.14: The squared exponential covariance function: panel (a): the shape of covariance functions with different parameter settings and in panel (b): random functions drawn from Gaussian processes with the same parameters as in panel (a).



(a) covariance function **(b)** random samples

Figure 2.15: The rational quadratic covariance function: panel (a): the shape of covariance functions with different parameter settings for α and $l = 1$. In panel (b): random functions drawn from Gaussian processes with the same parameters as in panel (a).

The Rational Quadratic Covariance Function

The *rational quadratic* (RQ) covariance function

$$k(\boldsymbol{x}, \boldsymbol{x}') = \left(1 + \frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_M^2}{2\alpha}\right)^{-\alpha} \tag{2.20}$$

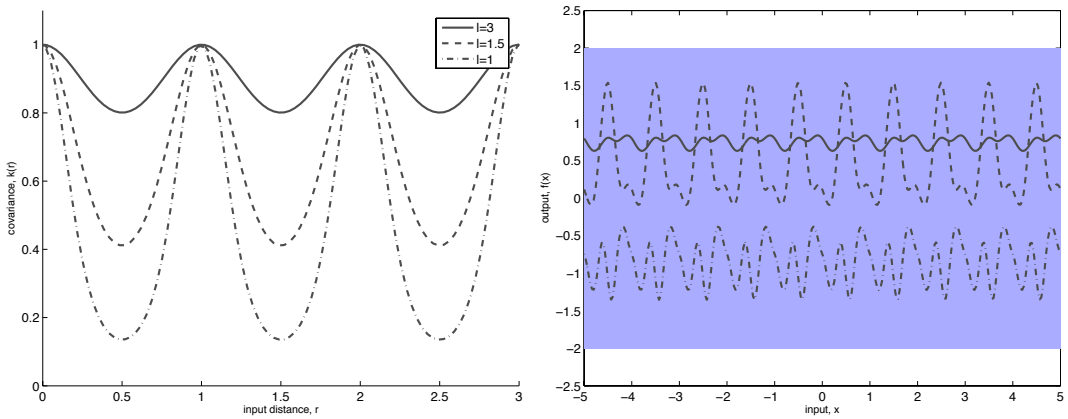
with $\alpha > 0$ is related to the SE. For $\alpha \to \infty$ the rational quadratic converges to the SE covariance function.

The Periodic Covariance Function

The isotropic form of a *periodic* covariance function is given by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{2\sin^2(\frac{\boldsymbol{x}-\boldsymbol{x}'}{2})}{l^2}\right), \tag{2.21}$$

where l is a scalar length scale parameter as described before. It is easy to verify that this function is non-stationary. Even far points of the input space are now able to "see" each other and have high covariance. It is not trivial to extend the isotropic to



(a) covariance function **(b)** random samples

Figure 2.16: The periodic covariance function: panel (a): covariance functions, and (b): random functions drawn from Gaussian processes with the same parameters as in panel (a).

the anisotropic form without the danger to produce an invalid covariance function. The following form was derived by Gibbs [1997]

$$k(\boldsymbol{x}, \boldsymbol{x}') = \prod_{d=1}^{D} \left(\frac{2\psi_d(\boldsymbol{x})\psi_d(\boldsymbol{x}')}{\psi_d^2(\boldsymbol{x}) + \psi_d^2(\boldsymbol{x}')} \right)^{1/2} \exp\left(- \sum_{d=1}^{D} \frac{(x_d - x_d')^2}{\psi_d^2(\boldsymbol{x}) + \psi_d^2(\boldsymbol{x}')} \right) \tag{2.22}$$

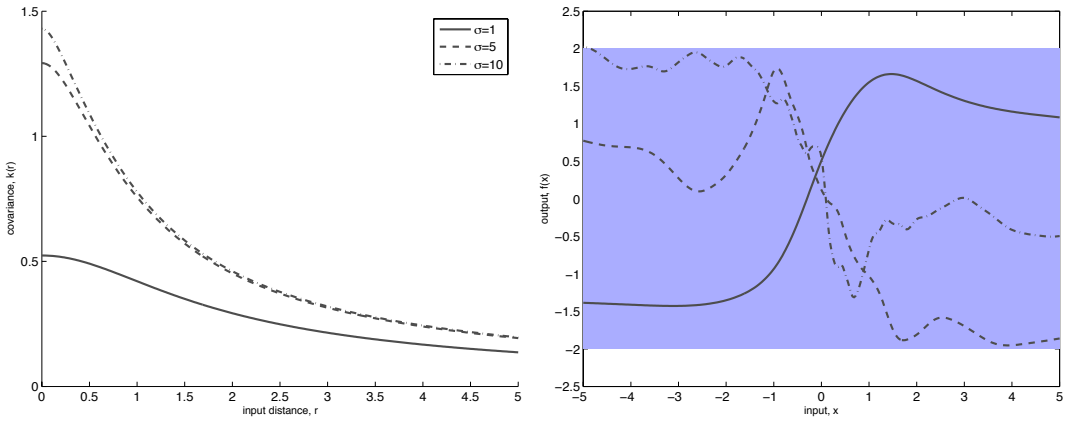
where ψ_d is an arbitrary positive "length-scale" function of $\boldsymbol{x}$.

The Neural Network Covariance Function

Another interesting non-stationary function is the *neural network* (NN) covariance function given by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \frac{2}{\pi} \sin^{-1}\left(\frac{2\tilde{\boldsymbol{x}}^T \boldsymbol{\Sigma} \tilde{\boldsymbol{x}}'}{\sqrt{(1 + 2\tilde{\boldsymbol{x}}^T \boldsymbol{\Sigma} \tilde{\boldsymbol{x}})(1 + 2\tilde{\boldsymbol{x}}'^T \boldsymbol{\Sigma} \tilde{\boldsymbol{x}}')}} \right), \tag{2.23}$$

where $\tilde{\boldsymbol{x}} = [1, x_1, \ldots, x_d]^T$ is the input vector with an additional interception term. The matrix $\boldsymbol{\Sigma}$ is a diagonal matrix with the parameter σ^2 for all d input dimensions plus an additional parameter σ_0^2 for the interception term. Figure 2.18 compares a stationary and a non-stationary covariance function. In both panels the resulting covariance matrix can be seen for the Matérn and the neural network respectively.



(a) covariance function **(b)** random samples

Figure 2.17: The neural network covariance function: panel (a): covariance functions, and (b): random functions drawn from Gaussian processes with the same parameters as in panel (a). σ_0 was set to the same value as σ

Other Covariance Functions

The following covariance functions are less interesting and most often only used in combination with the functions above.

The *exponential* covariance function

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}'\|_{\boldsymbol{M}}\right) \tag{2.24}$$

The *polynomial* covariance function

$$k(\boldsymbol{x}, \boldsymbol{x}') = \left(\boldsymbol{x}^T \boldsymbol{\Sigma} \boldsymbol{x}' + \sigma_0^2\right)^p \tag{2.25}$$

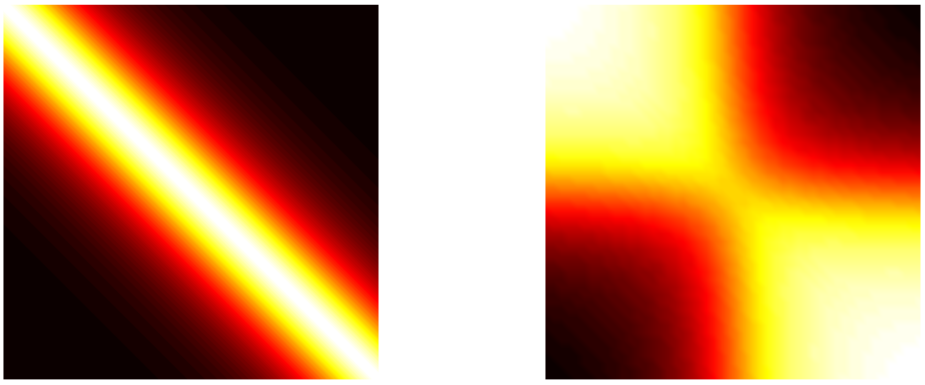
The *constant* covariance function

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_0^2; \tag{2.26}$$

The *white noise* covariance function

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \begin{cases} \sigma_n^2 & \text{if } \boldsymbol{x}_p = \boldsymbol{x}_q, \\ 0 & \text{else} \end{cases} \tag{2.27a}$$

$$= \sigma_n^2 \delta_{pq} \tag{2.27b}$$



(a) Matérn **(b)** neural network

Figure 2.18: Panel (a): a plot of the covariance matrix $\boldsymbol{K}$ calculated by the stationary Matérn. Panel (b): the result from a non-stationary neural network function.

2.4 Model Selection

It has been shown in section 2.2 how to do inference over the latent function f in a Gaussian process model with a given covariance function. The last section introduced some covariance functions that are common in the context of machine learning. It has been stated that these functions can be combined in several ways. For example one widely used function is given by the squared exponential together with a polynomial, a noise and a constant term as:

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \sigma_f^2 \exp\left(-\frac{1}{2}\|\boldsymbol{x}_p - \boldsymbol{x}_q\|_{\boldsymbol{M}}^2\right) + \sigma_p^2 \boldsymbol{x}_p^T \boldsymbol{\Sigma} \boldsymbol{x}_q + \sigma_n^2 \delta_{pq} + \sigma_c^2. \tag{2.28}$$

The open parameters $\boldsymbol{\theta} = [\{\boldsymbol{M}\}, \{\boldsymbol{\Sigma}\}, \sigma_f^2, \sigma_p^2, \sigma_n^2, \sigma_c^2]^T$ are called hyperparameters. It is easy to see that there are many different possibilities and algorithms to estimate the correct values. The expression hyperparameter is used because these parameters do not absorb the information from the training data. Also the statement that Gaussian processes are non-parametric models corresponds to this fact. The learning rate in a sequential least squares setting or the length scale of a radial basis function are examples of hyperparameters. Typically, the correct values for the hyperparameters are not known a priori and have to be estimated from the data. Both the selection of the hyperparameters and the choice of a covariance function itself are denoted as the model selection problem.

The ordinary Bayesian inference works in two stages:

1. Compute the posterior distribution
2. Evaluate all statements under the posterior distribution

The posterior distribution is calculated as

$$posterior = \frac{likelihood \times prior}{marginal\ likelihood}, \tag{2.29}$$

where the *prior* encodes the beliefs about the model/hyperparameters in advance. The *likelihood* tells how plausible the observed data under the parameter settings are. The normalization constant in the denominator is called *marginal likelihood* or *evidence* and plays a central role in model selection. It can be interpreted as an a priory predictive distribution for the model without having seen any data. It is similar to the likelihood, but the parameters had been integrated out. Once a data set of observations is given, the marginal likelihood can measure the probability of generating exactly this data set under the model with parameters randomly sampled from prior. It is important to note that the marginal likelihood is also a probability distribution and therefore it must also integrate up to one. It follows, that if a model can explain a lot of different datasets (and therefore it will be more complex) it will have only a small marginal likelihood on each individual set. In contrast, a very

simple model that can only describe a very small range of data sets will have a high marginal likelihood if one of these is actually observed. Figure 2.19 shows a sketch of the marginal likelihood given the model complexity for all possible datasets. This clearly avoids the problem of over-fitting because if two models are explaining the observed data, then the simple one will be chosen. This effect is often referred to as *Occam's razor* principle after William of Ockham. MacKay [1991, chapter 2] was showing that Bayesian model selection automatically implements Occam's razor.

In a fully Bayesian treatment with parameters $\boldsymbol{w}$ and hyperparameters $\boldsymbol{\theta}$ priors should be assigned to these as $p(\boldsymbol{w}|\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$ respectively. The posterior over the parameters $\boldsymbol{w}$ is given by

$$p(\boldsymbol{w}|\boldsymbol{y},\boldsymbol{X},\boldsymbol{\theta}) = \frac{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w},\boldsymbol{\theta})p(\boldsymbol{w}|\boldsymbol{\theta})}{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})} \tag{2.30}$$

with marginal likelihood

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = \int p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w},\boldsymbol{\theta})p(\boldsymbol{w},\boldsymbol{\theta})\,d\boldsymbol{w}. \tag{2.31}$$

In the second step the posterior over hyperparameters is calculated, where the marginal likelihood from the first step is now in the place of the likelihood giving

$$p(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{X}) = \frac{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y}|\boldsymbol{X})} \tag{2.32}$$

with normalizing constant in the denominator

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})p(\boldsymbol{\theta})\,d\boldsymbol{\theta}. \tag{2.33}$$

In the Gaussian process model there are no parameters $\boldsymbol{w}$ to fit, but the function itself. Therefore the marginal likelihood from Equation 2.31 is

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = \int p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{X},\boldsymbol{\theta})p(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{\theta})\,d\boldsymbol{f}, \tag{2.34}$$

where $\boldsymbol{f}|\boldsymbol{X},\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}))$ and $\boldsymbol{y}|\boldsymbol{f},\boldsymbol{X},\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{f},\sigma_n^2\boldsymbol{I})$. Because all quantities are Gaussian the integral is analytically tractable and can be computed using the standard form for multivariate Gaussian distributions (see Appendix A for details), giving

$$\log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T\boldsymbol{K}_y^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K}_y| - \frac{n}{2}\log 2\pi, \tag{2.35}$$

where $\boldsymbol{K}_y$ substitutes $\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I}$ and $|\cdot|$ is the determinant. The marginal likelihood plays an important role in model selection and model comparison. Occam's razor can also be derived directly from Equation 2.35. The first term $\boldsymbol{y}^T\boldsymbol{K}_y^{-1}\boldsymbol{y}/2$ is called *data-fit* and the (always positive) term $\log|\boldsymbol{K}_y|/2$ subtracted from it is the

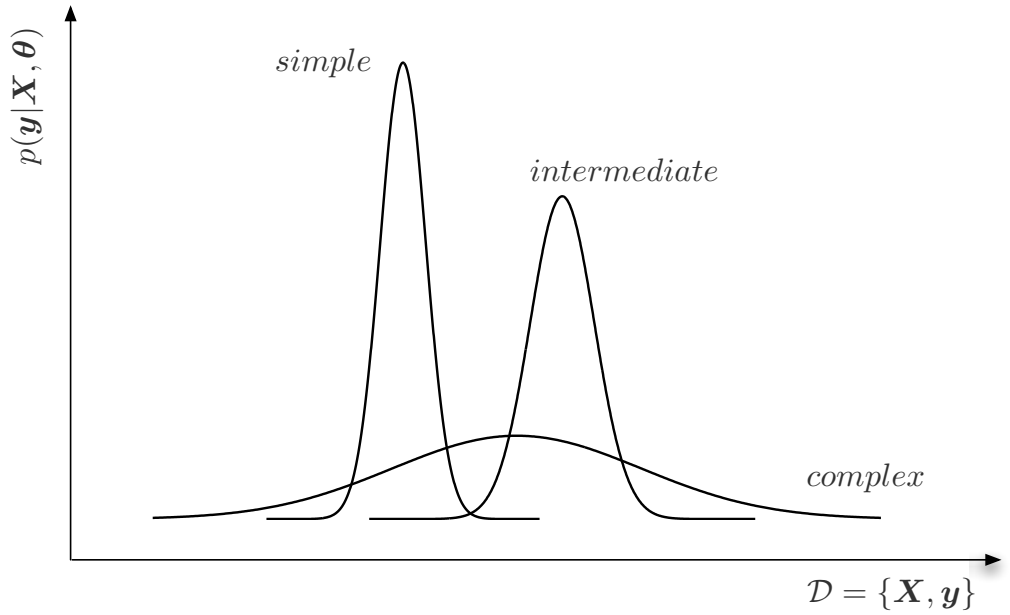


Figure 2.19: The marginal likelihood of the model given the data. The complex model covers a wide range of the input set, but has a low probability for all. The simple model is only good for a very few datasets. If two models would explain the data, then the simple one would be selected by marginal likelihood. (Illustration after [Rasmussen and Williams, 2006, Figure 5.2] and [MacKay, 1991, Figure 2.2])

complexity penalty. The last term is just a normalization constant.

Figure 2.20 shows the (log) marginal likelihood as a function of the length scale and the noise level. The data had been generated from a random function with length scale 1.1 and noise sampled from $\mathcal{N}(0, 0.1)$. The ML-II estimation is approximately at these values. It can also be seen that the marginal likelihood drops dramatically for a long length scale with little noise. In this situation the Gaussian process is not able to explain the data. If the noise level is increased, the $\mathcal{GP}$ is still unable to match the data points, but this is expected since the model assumes there is a lot of noise.

In general the integral in Equation 2.33 is analytically intractable and exact marginalization is impossible. One common approach is to set the hyperparameter to a specific value determined by maximizing Equation 2.34. This is known as *empirical Bayes*, *evidence approximation* or *type II maximum likelihood* (ML-II). It is also possible to interpret the ML-II as *maximum a posteriori* (MAP) of $p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{X}) \propto p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})$. This runs again the risk of over-fitting because it is a single point estimate. The hope is that the ML-II estimate may not differ much from the result of integrating over

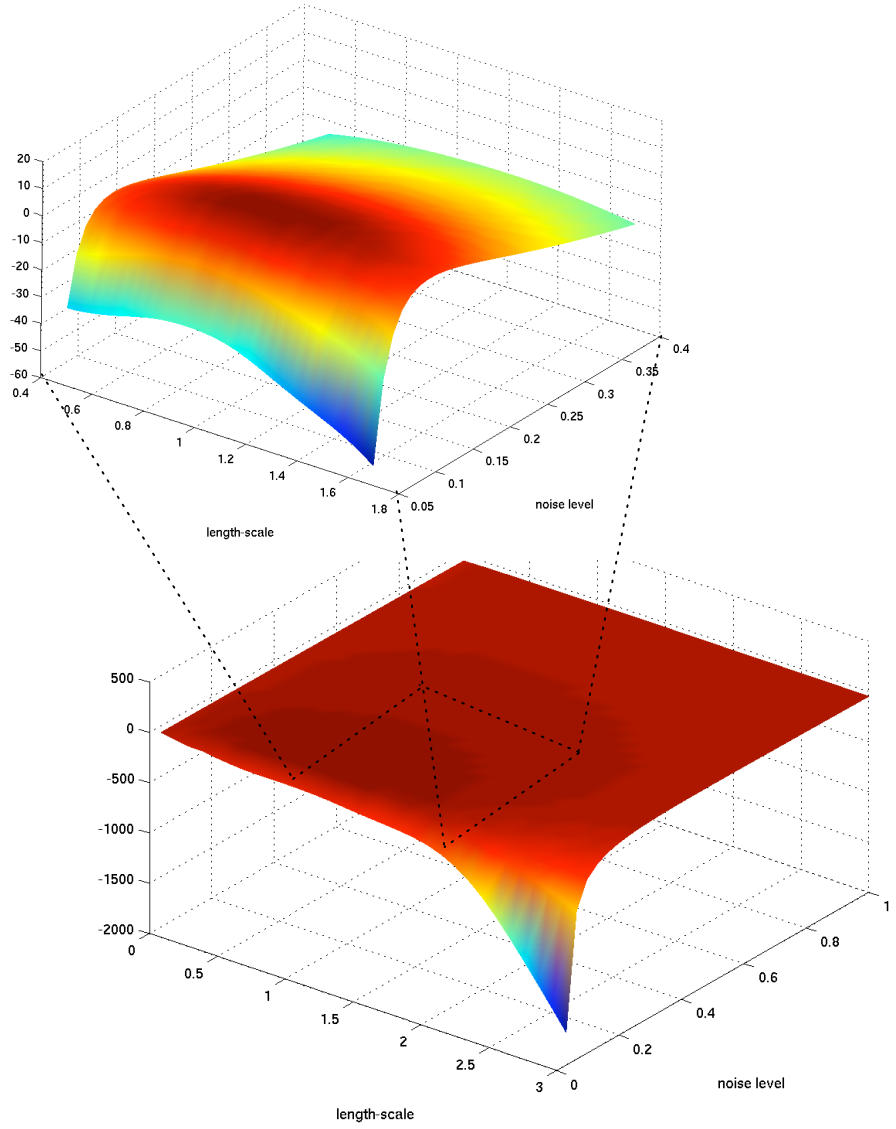
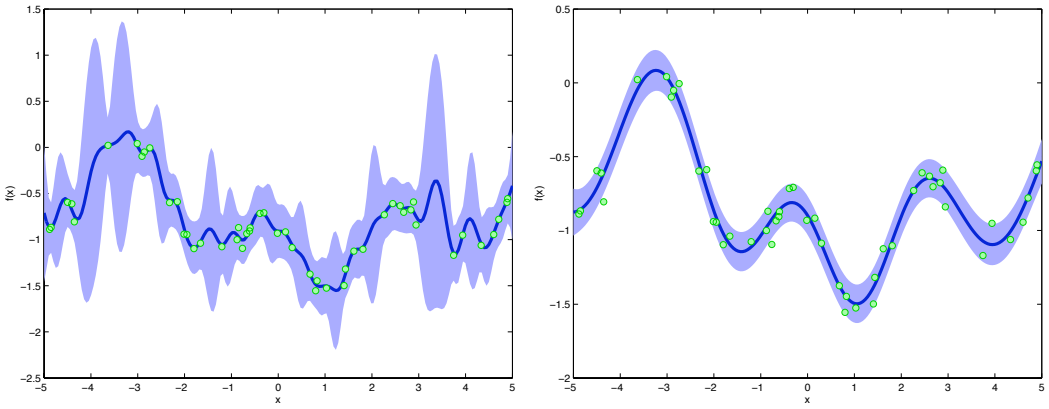


Figure 2.20: The log marginal likelihood as a function of two hyperparameters. A randomly created function with length scale $l^2 = 1.1^2$ and noise level $\sigma_n^2 = 0.1^2$ had been used to generate samples. Then the log marginal likelihood as a function of l^2 and σ_n^2 had been evaluated on these samples. The marginal likelihood has its maximum at (approximately) the actually used values. Note that there is not necessarily one global maximum!



(a) random **(b)** optimized

Figure 2.21: The same configuration as in section 2.3, Figure 2.10: A $\mathcal{GP}$ with covariance function $k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \sigma_f^2 \exp(-(\|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2)/(2l^2)) + \sigma_n^2 \delta_{pq}$. The hyperparameters $\boldsymbol{\theta} = [\sigma_f^2, l^2, \sigma_n^2]^R$ are the signal variance, length-scale and noise term respectively. Panel (a) shows a random configuration of the hyperparameters, whereas in panel (b) the result after the conjugate gradient optimization is shown.

hyperparameters. In order to perform the maximization of the marginal likelihood, the partial derivatives with respect to the hyperparameters are needed:

$$\frac{\partial}{\partial \theta_j} \log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}^{-1} \frac{\partial \boldsymbol{K}}{\partial \theta_j} \boldsymbol{K}^{-1}\boldsymbol{y} - \frac{1}{2} \operatorname{tr}\left(\boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta_j}\right) \tag{2.36}$$

$$= \frac{1}{2} \operatorname{tr}\left((\boldsymbol{K}^{-1}\boldsymbol{y})(\boldsymbol{K}^{-1}\boldsymbol{y})^T - \boldsymbol{K}^{-1})\frac{\partial \boldsymbol{K}}{\partial \theta_j}\right) \tag{2.37}$$

Gradient descent can be performed using a multivariate optimization algorithm. Pseudo-code for a *Polak-Ribière Conjugate Gradient* (CG) algorithm can be found in section 2.5. Figure 2.21 shows an example where the ML-II estimate had been used.

There are other methods that try to directly approximate Equation 2.33 like *Expectations Propagation* (EP), *Laplace Approximation* or *Markov Chain Monte Carlo* (MCMC) methods (see also the work of Sundararajan and Keerthi [2001]). As a drawback, these methods are much slower than the maximization approach.

Approximate Bayesian Inference is not part of this thesis and therefore the interested reader is referred to the literature. A description of the Expectation Propagation algorithm can be found in [Minka, 2001] and [Bishop, 2006]. An introduction to Laplace's approximation is described in the book of MacKay [2003] and an application to Gaussian processes is given by Williams [1998]. Markov chain Monte

Carlo methods with in reference to Bayesian inference can be found in [MacKay, 1992, 2003, Neal, 1993] and [Neal, 1997]

2.5 Nonlinear Conjugate Gradient Methods

Conjugate Gradient (CG) methods can be used to minimize any continuous function $f(\boldsymbol{x})$ whose gradient $f'(\boldsymbol{x})$ can be computed. At each step of iteration, a search direction is computed and then a *line search* is performed to find a (local) minimum along this direction. This is just a short outline of the used algorithms. For a detailed explanation of the methods sketched here see the book of Nocedal and Wright [1999].

There are two popular methods to compute the search directions: *Fletcher-Reeve* and *Polak–Ribière*. They have the from

$$\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k) + \beta_k \boldsymbol{p}_{k-1}, \tag{2.38}$$

where β_k is a scalar that ensures that $\boldsymbol{p}_k$ and $\boldsymbol{p}_{k-1}$ are *conjugate*. See Algorithm 2.2 for pseudo code of the Polak–Ribière conjugate gradient version used in this thesis.

Algorithm 2.2 Polak–Ribière Conjugate Gradient (similar to [Nocedal and Wright, 1999])

1: **input:** f (the objective function), $\boldsymbol{x}_0$ (initial guess for $\boldsymbol{x}$)
2: Evaluate $\boldsymbol{f}_0 = f(\boldsymbol{x}_0)$
3: Evaluate $\nabla \boldsymbol{f}_0 = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0)$
4: Set $\boldsymbol{p}_0 = -\nabla \boldsymbol{f}_0$
5: $k = 0$
6: **while** $\nabla \boldsymbol{f}_k \neq 0$ **do**
7: $\quad \alpha_k = \text{LINESEARCH}(f, \boldsymbol{x}_k, \boldsymbol{p}_k)$
8: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
9: $\quad$ Evaluate $\nabla \boldsymbol{f}_{k+1} = \nabla f(\boldsymbol{x}_{k+1})$
10: $\quad \beta_{k+1} = \frac{\nabla \boldsymbol{f}_{k+1}^T (\nabla \boldsymbol{f}_{k+1} - \nabla \boldsymbol{f}_k)}{||\nabla \boldsymbol{f}_k||^2}$
11: $\quad \beta_{k+1} = \max\{\beta_{k+1}, 0\}$
12: $\quad \boldsymbol{p}_{k+1} = -\nabla \boldsymbol{f}_{k+1} + \beta_{k+1} \boldsymbol{p}_k$
13: $\quad k = k + 1$
14: **end while**

In order to use the Fletcher-Reeve version simply change line 10 in Algorithm 2.2 to

$$\beta_{k+1} \leftarrow \frac{\nabla \boldsymbol{f}_{k+1}^T \nabla \boldsymbol{f}_{k+1}}{\nabla \boldsymbol{f}_k^T \nabla \boldsymbol{f}_k}; \tag{2.39}$$

2.5.1 Line Search

After a search direction p_k has been determined the task of the line search algorithm is to find a proper *step length* α along this direction. This can be done by minimizing the one-dimensional problem

$$\min_{\alpha>0} f(\boldsymbol{x} + \alpha p_k). \tag{2.40}$$

Line search is not going to solve the problem exactly, but instead it tries to do the best in a limited number of function evaluations. Typically, there are some conditions that a step length candidate has to hold:

$$f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) \leq f(\boldsymbol{x}_k) + c_1 \alpha_k \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) \tag{2.41a}$$

$$\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) \geq c_2 \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) \tag{2.41b}$$

with positive constants $c_1, c_2 \in (0, 1)$ and $c_1 < c_2$. A typical value of c_2 for conjugate gradient methods is 0.1. These are called the *Wolfe conditions* and they ensure the estimation of an acceptable step length. For the so-called *strong Wolfe conditions* Equation 2.41b is changed to

$$|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)| \leq c_2 |\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)|. \tag{2.41c}$$

Typically Equation 2.41a is referred to as the *sufficient decrease condition* and Equation 2.41c is called the *curvature condition*.

For pseudocode of the line search algorithm from [Nocedal and Wright, 1999], see Algorithm 2.3 and Algorithm 2.4. The LINESEARCH function starts with an initial guess $\alpha_1 > 0$ for the step length. In each iteration it extrapolates the step length until it has an acceptable value or an interval around the desired step length can be found. In the second case, the function REDUCE is used to successively shrink the interval. Interpolation is used to choose a step length α_j between the left (α_{lo}) and right (α_{hi}) interval border. Then α_j replaces one of these such that

- the interval still contains the desired step length satisfying the strong Wolfe conditions;
- $\phi(\alpha_{lo})$ has the smallest function value of all step lengths generated so far; and
- α_{hi} holds $\phi'(\alpha_{lo})(\alpha_{hi} - \alpha_{lo}) < 0$.

See also the work of More et al. [1992] for additional information.

Algorithm 2.3 Line Search Algorithm (from [Nocedal and Wright, 1999])

Require: $\alpha_1 > 0$
Require: $0 < c_1 < c_2 < 1$
function LINESEARCH($\alpha_1, \alpha_{max}, c_1, c_2$)
  define $\phi(\alpha) := f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k)$
  define $\phi'(\alpha) := \nabla_\alpha f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) := \nabla_{\boldsymbol{x}_k} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k)^T \boldsymbol{p}_k$
  $\alpha_0 = 0$
  $i = 1$
  loop
    Evaluate $\phi(\alpha_i)$
    if $\phi(\alpha_i) > \phi(0) + c_1 \alpha_i \phi'(0)$ or $[\phi(\alpha_i) \geq \phi(\alpha_{i-1})$ and $i > 1]$ **then**
      return REDUCE(α_{i-1}, α_i)
    end if
    Evaluate $\phi'(\alpha_i)$
    if $|\phi'(\alpha_i)| \leq -c_2 \phi'(0)$ **then**
      return α_i
    end if
    if $\phi'(\alpha_i) \geq 0$ **then**
      return REDUCE(α_i, α_{i-1})
    end if
    EXTRAPOLATE(α_i, α_{max})
    $i = i + 1$
  end loop
end function

Algorithm 2.4 Interpolation Algorithm (from [Nocedal and Wright, 1999])

function REDUCE(α_{lo}, α_{hi})
  loop
    Choose $\alpha_j \in [\alpha_{lo}, \alpha_{hi}]$ using interpolation (cubic, quadratic or bisection)
    Evaluate $\phi(\alpha_j)$
    if $\phi(\alpha_j) > \phi(0) + c_1\alpha_j\phi'(0)$ or $\phi(\alpha_j) \geq \phi(\alpha_{lo})$ **then**
      $\alpha_{hi} = \alpha_j$
    else
      Evaluate $\phi'(\alpha_j)$
      if $|\phi'(\alpha_j)| \leq -c_2\phi'(0)$ **then**
        return α_j
      end if
      if $\phi'(\alpha_j)(\alpha_{hi} - \alpha_{lo}) \geq 0$ **then**
        $\alpha_{hi} = \alpha_{lo}$
      end if
      $\alpha_{lo} = \alpha_j$
    end if
  end loop
end function

CHAPTER 3

The Learning from Demonstration Framework

This chapter presents a probabilistic framework for the Learning from Demonstration task. We introduce the theoretical aspects of Gaussian processes when used to encode robot movements. We also present methods which allow the robot to acquire more than the simple movement itself, they allow it to learn *what* is important to *reproduce* or to *imitate*.

Section 3.1 introduces the terminology and how variability in demonstration can be used to extract additional informations about the task to learn.

Section 3.2 extends the standard Gaussian process to a model with variable noise using *heteroscedastic* Gaussian processes with an expectation-maximization algorithm.

Section 3.3 explains how several controllers for reproduction can be merged into a single controller.

Section 3.4 is about inverse kinematics and how it can be used to transform the policy for reproduction from task space into joint space.

Section 3.5 gives an introduction to *Dynamic Time Warping*, a useful preprocessing step.

Section 3.6 is a summary of all modules described in this thesis.

3.1 Policy Representation & Constraints

The core of this framework uses Gaussian processes, as described in the previous chapter, to encode a set of demonstrations. This encoding is called a *policy* denoted with π. During a demonstration, sensor information is collected together with a timestamp and stored as a state/observation $\boldsymbol{\xi} = (\xi_t^{(i)}, \boldsymbol{\xi}_s^{(i)}) \in \mathbb{R}^D$. Here, $\xi_t^{(i)}$ denoting the i-th *temporal value* $\in \mathbb{R}$ and $\boldsymbol{\xi}_s^{(i)} \in \mathbb{R}^{(D-1)}$ is the i-th vector of *spatial values*. The time stamp is the temporal part of the state vector, whereas the spatial part is represented through coordinates in *joint space* ($\theta_1, \ldots, \theta_k$ for the k encoders) or *task space* (3D coordinates and orientation). Suppose we have n demonstrations and each demonstration is resampled to a fixed size of T, then the data set $\mathcal{D}$ of all observations will be of length $N = nT$, formally $\mathcal{D} = \{(\xi_t^{(i)}, \boldsymbol{\xi}_s^{(i)}) | i = 1, \ldots, N\}$. The policy used here maps from ξ_t to $\boldsymbol{\xi}_s$ using

$$p(\boldsymbol{\xi}_s | \xi_t, \mathcal{D}) = \mathcal{GP}(\cdot, \cdot) \quad \forall\, \xi_t \in \mathbb{R} \tag{3.1}$$

in terms of a Gaussian process. This approach fits perfectly the requirements of an approximation in the context of robots:

1. continuous and smooth paths
2. generalization over multiple demonstrations

The Gaussian process covariance function controls the function shape and ensures a smooth path[1] and the fundamental $\mathcal{GP}$ algebra calculates the mean over demonstrations. This reduces noise (introduced by sensors and human jitter during the demonstration) and recovers the underlying trajectory.

The variation and similarity between demonstrations play a key role for the reproduction part because they are already defining *what* is important to imitate. Assume a robot arm scenario where the goal is to reach a specific end state (e.g. given 3D coordinates) and the start state is chosen randomly. The trajectories will differ significantly at the beginning, whereas the position should be more or less the same at the end. Such a part of a trajectory with low variation is defined as a *constraint* because no discrepancy is desired (see Figure 3.1). We use *heteroscedastic* Gaussian processes, as discussed in the next section, to model these constraints.

1 The covariance function should be chosen appropriately depending on the task. A good choice for smooth trajectories is the squared exponential function.

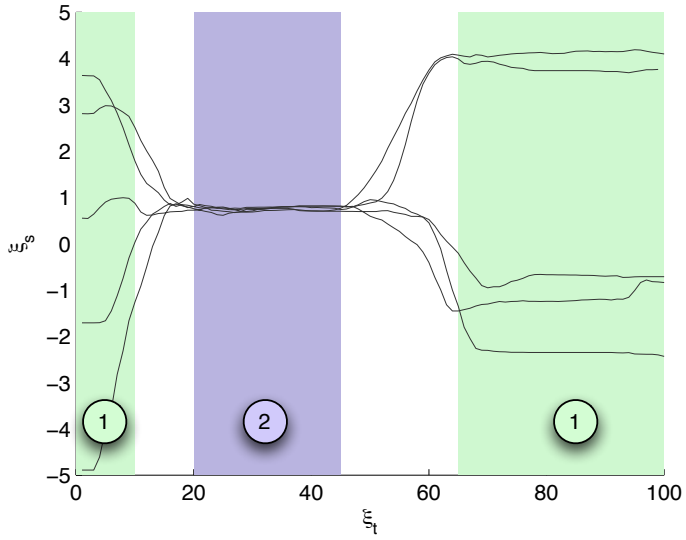


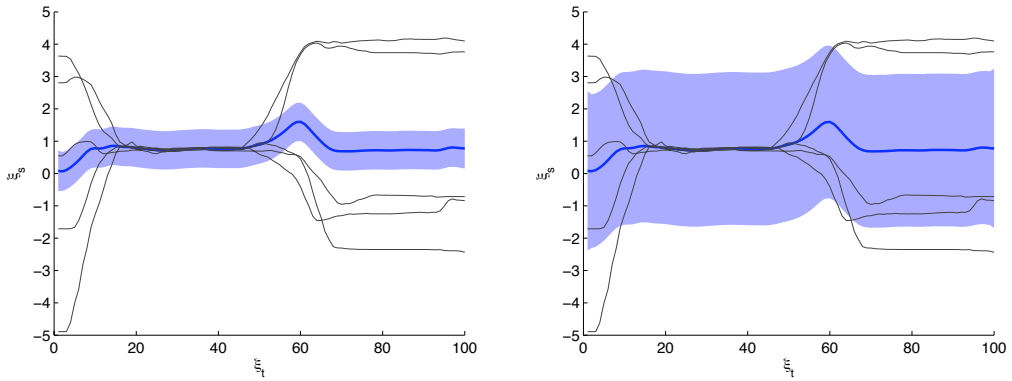
Figure 3.1: An Illustration of policy constraints. The demonstrations (lines) vary significantly at time frames marked with (1), hence this position seems to be uncritical. In contrast, all demonstrations meet at position ≈ 0.9 in the frame marked with (2). It can be concluded that the robot should also follow this in the reproduction step.

3.2 Constraint Encoding with *Heteroscedastic GP Models*

The standard Gaussian process model for regression as discussed in chapter 2 has one important limitation: it assumes a constant noise level. The problem is sketched in Figure 3.2 for one dimension. While the standard Gaussian process model is still able to correctly estimate the mean, it totally fails to approximate the variance. If the global noise level is too small, the variance is underestimated at the beginning, if the noise level is too big, the variance is overestimated at the end. Taking *input-dependnent noise* into account can solve this problem. Such models are said to be *heteroscedastic*.

Goldberg et al. [1997] was one of the first who developed an input-dependent noise model for Gaussian processes. Also [Snelson and Ghahramani, 2006b] and [Snelson, 2007] used a variable noise model for their sparse pseudo-input Gaussian processes (SPGP). The rest of this section reviews the approach suggested in [Kersting et al., 2007] and [Plagemann, 2008] and discusses the implementation details for this Learning from Demonstration framework.

We keep the notation from section 2.2 for convenience and recall the definitions: There are observed target values y, given by the true underlying function value f and some i. i. d. noise $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ as $y = f + \epsilon$. The data sets of n training inputs



(a) underestimated variance **(b)** overestimated variance

Figure 3.2: An example of a standard Gaussian process applied to a data set that requires a variable noise model. Figure (a) clearly underestimates the variance from 0 to 60, whereas the GP in Figure (b) overestimates the variance from 40 to 100

$(\{x^{(i)}\}_{i=1}^{n})$ and n_* test inputs $(\{x_*^{(i)}\}_{i=1}^{n_*})$ are collected in the design matrices $\boldsymbol{X}$ and $\boldsymbol{X}_*$ respectively. Given the training and test inputs, the Gaussian process predictive distribution is given by

$$\boldsymbol{f}_*|\boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}(\bar{\boldsymbol{f}}_*, \text{cov}[\boldsymbol{f}_*]), \quad \text{with} \tag{3.2a}$$

$$\bar{\boldsymbol{f}}_* = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{y} \tag{3.2b}$$

$$\text{cov}[\boldsymbol{f}_*] = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*). \tag{3.2c}$$

From these equations it can be seen that the noise rate is input-independent with value σ_n^2. This can be extended to a heteroscedastic Gaussian process model by introducing a noise function that depends on the input: $r(\boldsymbol{x})$. This function is no more required to be constant. Equation 3.2b and Equation 3.2c of the predictive distribution are changed to

$$\bar{\boldsymbol{f}}_* = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{R}(\boldsymbol{X}))^{-1}\boldsymbol{y} \tag{3.3a}$$

$$\text{cov}[\boldsymbol{f}_*] = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) + \boldsymbol{R}(\boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{R}(\boldsymbol{X}))^{-1}\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \tag{3.3b}$$

respectively. In the equations above, $\boldsymbol{R}(\boldsymbol{X})$ is the noise covariance matrix of training inputs, defined as $\text{diag}(\boldsymbol{r})$, with $\boldsymbol{r} = (r(\boldsymbol{x}^{(1)}), r(\boldsymbol{x}^{(2)}), \ldots, r(\boldsymbol{x}^{(n)}))^T$, and $\boldsymbol{R}(\boldsymbol{X}_*) = \text{diag}(\boldsymbol{r}_*)$, with $\boldsymbol{r}_* = (r(\boldsymbol{x}_*^{(1)}), r(\boldsymbol{x}_*^{(2)}), \ldots, r(\boldsymbol{x}_*^{(n_*)}))^T$ is the noise covariance matrix of the test inputs.

We follow Snelson and Ghahramani [2006b] using a second independent Gaussian process to model $z(\boldsymbol{x}) = \log(r(\boldsymbol{x}))$ (the logarithms of the noise function outputs).

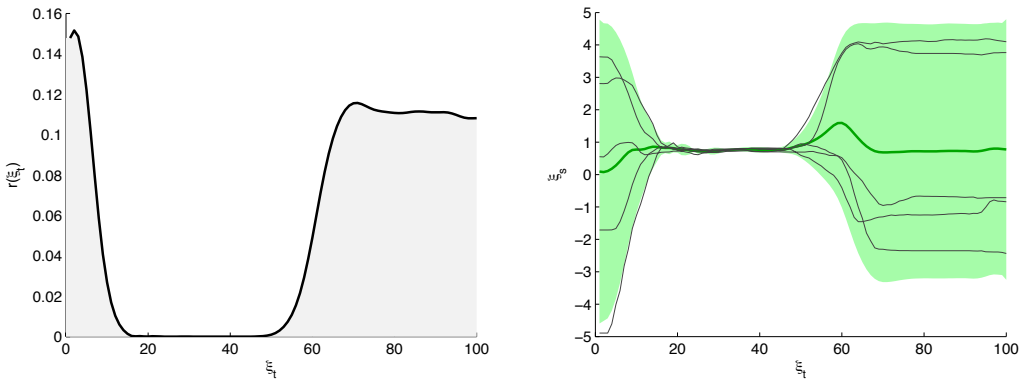
This process maintains its own covariance function and will be referred to as $\mathcal{GP}_z$. We use the same training inputs, $\boldsymbol{x}$, as in the original Gaussian process for the noise process, but other choices are also possible. With $\boldsymbol{z}_*$ as the $\mathcal{GP}_z$ posterior prediction, the predictive distribution for $\boldsymbol{f}_*$ changes to

$$p(\boldsymbol{f}_*|\boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}) = \int\int p(\boldsymbol{f}_*|\boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{z}_*) p(\boldsymbol{z}, \boldsymbol{z}_*|\boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y})\, d\boldsymbol{z}\, d\boldsymbol{z}_*, \tag{3.4}$$

where the last term prevents an analytical solution of the integral. A common method is to approximate $p(\boldsymbol{f}_*|\boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}) \approx p(\boldsymbol{f}_*|\boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}, \tilde{\boldsymbol{z}}, \tilde{\boldsymbol{z}}_*)$, where $\tilde{\boldsymbol{z}}, \tilde{\boldsymbol{z}}_* = \arg\max_{\boldsymbol{z}, \boldsymbol{z}_*} p(\boldsymbol{z}, \boldsymbol{z}_*|\boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y})$.

Kersting et al. [2007] introduced an expectation-maximization (EM) algorithm, that iteratively estimates $\mathcal{GP}_z$ and then combines it with a noise free $\mathcal{GP}$ to estimate a heteroscedastic Gaussian process. A description is given in Algorithm 3.1. First, an initial standard Gaussian process is trained on the data using maximum likelihood for parameter estimation. The predictions of this process are used to estimate the (log) noise levels empirically by calculating the arithmetic mean between samples from the posterior distribution and the observed data (see function EstimateLogNoise).

The result after applying this technique to the problem set shown above can be seen in Figure 3.3. The left figure illustrates the noise function approximated by $\mathcal{GP}_z$. The other is the resulting heteroscedastic Gaussian process, now able to correctly estimate the noise variances for all inputs (see Figure 3.2 for comparison).



(a) latent local noise rate **(b)** heteroscedastic GP

Figure 3.3: A heteroscedastic Gaussian process is applied to the same data set as in Figure 3.2. Panel (a) shows the noise level estimated by $\mathcal{GP}_z$. The plot in panel (b) is the resulting heteroscedastic Gaussian process after the EM run.

Algorithm 3.1 EM algorithm for the heteroscedastic Gaussian process model as described in [Kersting et al., 2007]

input: $\boldsymbol{X}$ (inputs), $\boldsymbol{y}$ (observations)
$\mathcal{GP}_0$ = STANDARDGP($\boldsymbol{X}$, $\boldsymbol{y}$) as in Algorithm 2.1
for $i = 0 \ldots \infty$ **do**
    $\boldsymbol{z} =$ ESTIMATELOGNOISE($\mathcal{GP}_i$, $\boldsymbol{X}$, $\boldsymbol{y}$)
    $\mathcal{GP}_z$ = STANDARDGP($\boldsymbol{X}$, $\boldsymbol{z}$) as in Algorithm 2.1
    define $r(\boldsymbol{x}) = \exp(\mathcal{GP}_z(\boldsymbol{x}))$
    $\mathcal{GP}_{i+1}$ =HETEROSCEDASTICGP($\mathcal{GP}_i$, $r(\boldsymbol{x})$, $\boldsymbol{X}$, $\boldsymbol{y}$) using Equation 3.3
    if EM converged **then**
        return $\mathcal{GP}_{i+1}$
    end if
end for

function ESTIMATELOGNOISE($\mathcal{GP}$, $\boldsymbol{X}$, $\boldsymbol{y}$)
    define S as number of samples
    for all $y_i \in \boldsymbol{y}$ **do**
        for $k = 1 \ldots S$ **do**
            $\tilde{y}_{i,k} \sim \mathcal{GP}(y_i)$ (sample from predictive distribution)
        end for
        $\mathbb{V}[y_i, \mathcal{GP}(\boldsymbol{x}_i)] \approx S^{-1} \sum_{k=1}^{S} \frac{1}{2}(y_i - \tilde{y}_{i,k})^2$
        $z_i = \log(\mathbb{V}[y_i, \mathcal{GP}(\boldsymbol{x}_i)])$
    end for
    return $\boldsymbol{z}$ (log noise estimate)
end function

3.3 Extracting multiple Constraints

The previous section discussed how heteroscedastic Gaussian processes can be used to extract policy constraints from multiple demonstrations. However, in many cases the task requires that the robot considers more than a single constraint. This happens typically if the robot acts on several objects. In addition to the constraints, the new policy still has to fulfill the requirements from section 3.1 (to generate *continuous* and *smooth* paths).

Assuming M different constraints, encoded as a set of Gaussian processes, we also get M predictive distributions $\{p(\boldsymbol{\xi}_s|\xi_t)^{(h)} = \mathcal{N}(\bar{\boldsymbol{f}}_*, \mathrm{cov}[\boldsymbol{f}_*])^{(h)} | h = 1, \ldots, M\}$. By considering the constraints as independent, we can use the product of Gaussian distributions to generate a new predictive distribution $p(\hat{\boldsymbol{\xi}}_s|\hat{\xi}_t)$ as

$$p(\hat{\boldsymbol{\xi}}_s|\hat{\xi}_t) = \prod_{h=1}^{M} p(\boldsymbol{\xi}_s|\xi_t)^{(h)}, \tag{3.5}$$

where the product of Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ is given by

$$\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \cdot \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) = \boldsymbol{Z}^{-1}\mathcal{N}(\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3) \tag{3.6a}$$

$$\text{with} \quad \boldsymbol{\mu}_3 = \boldsymbol{\Sigma}_3(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2) \tag{3.6b}$$

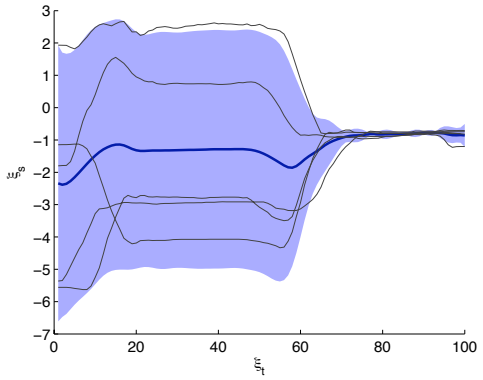
$$\text{and} \quad \boldsymbol{\Sigma}_3 = (\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1})^{-1}. \tag{3.6c}$$

The normalization constant $\boldsymbol{Z}^{-1}$ in Equation 3.6 is defined as

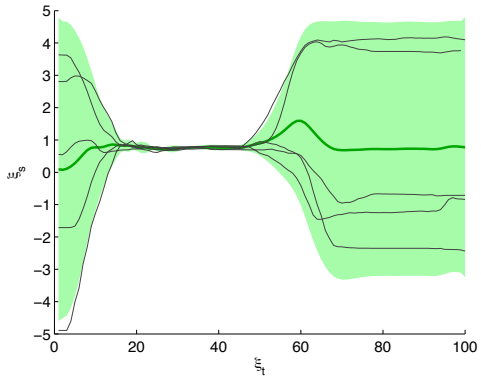
$$\boldsymbol{Z}^{-1} = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\right) \tag{3.7}$$

and looks itself like a Gaussian distribution in $\boldsymbol{\mu}_1$ or $\boldsymbol{\mu}_2$.

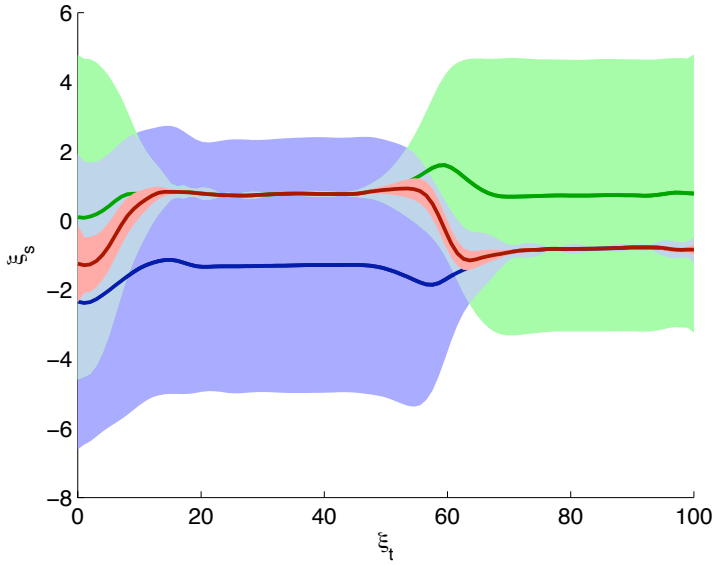
Note that we do not have to perform the Gaussian product at each time step as in other methods since the Gaussian process already provides the covariance for the entire function. The reproduced trajectory $\hat{\boldsymbol{\xi}}$ is a compromise between the M demonstrated trajectories $\boldsymbol{\xi}^{(h)}$, each weighted inverse to their variances. An example is illustrated in Figure 3.4. It can be seen that the trajectory with the lowest variance has the most influence. It changes over time since the variance also changes.



(a) Constraint 1



(b) Constraint 2



(c) Reproduced trajectory

Figure 3.4: In Panel (a) and (b) are two independent constraints (bold line and shaded regions) extracted from the demonstrations (thin lines). Panel (c) show the resulting trajectory (red) after the two constraints are combined. As it can be seen easily, the new "controller" considers both constraints weighted by their variances. (This figure is inspired by [Calinon, 2007, Figure 2.14].)

3.4 A unified View of Joint Space and Task Space

The previous section showed how several constraints can be combined into one using the Gaussian product property. This requires the constraints to be in the same space (either joint space $\boldsymbol{\theta}$ or task space $\boldsymbol{x}$), but sometimes it can be useful to consider constraints in different scopes.

To achieve this, a simple Jacobian-based *inverse kinematics* (IK) solution is used as in [Calinon and Billard, 2008]. The Jacobian matrix $\boldsymbol{J}$ is a function of $\boldsymbol{\theta}$ and is defined by

$$\boldsymbol{J}(\boldsymbol{\theta}) = \left(\frac{\partial x_i}{\partial \theta_j} \right)_{i,j} \tag{3.8}$$

for task space coordintes x_i and joints θ_j. Then, the forward dynamics

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{3.9}$$

are describing the velocities of the end effector in terms of the Jacobian. We use the *pseudoinverse*[1] $\boldsymbol{J}^\dagger$ of $\boldsymbol{J}$ given by

$$\boldsymbol{J}^\dagger = (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T \tag{3.10}$$

to iteratively calculate an update value $\Delta\boldsymbol{\theta}$ that leads to $\Delta\boldsymbol{x} = \boldsymbol{J}\Delta\boldsymbol{\theta}$, where $\Delta\boldsymbol{x}$ is the desired change in position. Using the pseudoinverse has the advantage to obtain the best solution in a least square sense and we are able to define further optimization criteria ϕ using

$$\Delta\boldsymbol{\theta} = \boldsymbol{J}^\dagger\Delta\boldsymbol{x} + (\boldsymbol{I} - \boldsymbol{J}^\dagger\boldsymbol{J})\phi, \tag{3.11}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{I} - \boldsymbol{J}^\dagger\boldsymbol{J}$ is a projection in the null space of the Jacobian matrix. The nullspace method was introduced by Liegois [1977] and can be exploited during the inverse kinematics calculation by keeping the joint angles as close as possible to the demonstrated joint space values for all valid solutions in task space. In addition, we can also project the covariance matrix from task space to joint space if we assume that the true nonlinear projection is locally linearizable through the Jacobian.

Formally, let $\boldsymbol{\theta}_t$, $\boldsymbol{x}_t$ be the actual joint angles and pose at time t and let $\hat{\boldsymbol{\theta}}_t$, $\hat{\boldsymbol{x}}_t$ the desired joint angles and pose calculated by the Gaussian process policy together

1 $\boldsymbol{J}^\dagger$ is also called the *Moore-Penrose inverse* of $\boldsymbol{J}$

with their covariance matrices $\boldsymbol{\Sigma}_t^{\hat{\theta}}$, $\boldsymbol{\Sigma}_t^{\hat{x}}$, then we set

$$\boldsymbol{\theta}_{t+1} = (\boldsymbol{x}_{t+1} - \hat{\boldsymbol{x}}_t) + \alpha \left(\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \boldsymbol{J}(\boldsymbol{\theta}_t) \right) (\hat{\boldsymbol{\theta}}_{t+1} - \boldsymbol{\theta}_t) \tag{3.12a}$$

$$\boldsymbol{\Sigma}_{t+1}^{\theta} = \boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \boldsymbol{\Sigma}_t^{\hat{x}} \left(\boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \right)^T. \tag{3.12b}$$

Here α is a weight factor for the secondary goal in the Jacobian null space. The result is a joint space representation of the constraints in task space. Using this together with $\hat{\boldsymbol{\theta}}$, $\boldsymbol{\Sigma}^{\hat{\theta}}$ and Equation 3.5, a new set of constraints can be obtained which now considers both spaces (see also Algorithm 3.2 in section 3.6).

3.5 Preprocessing

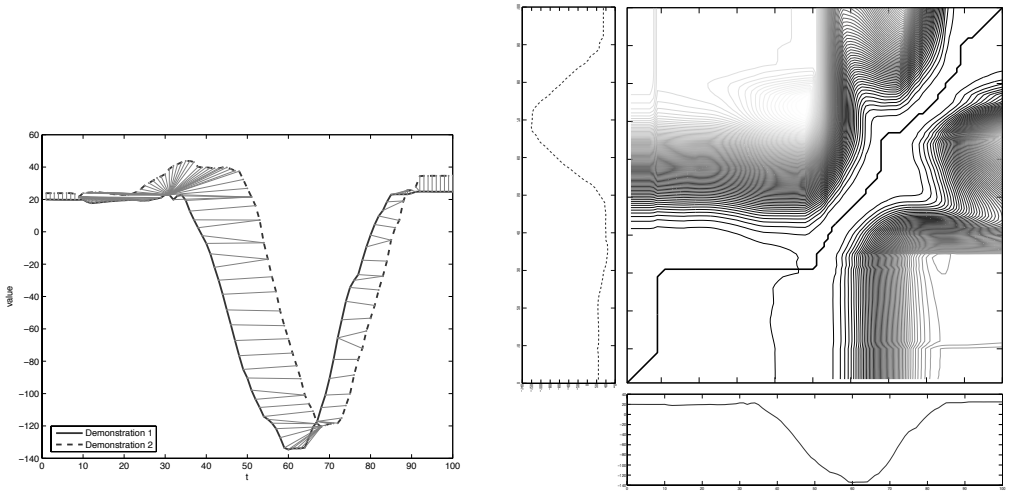
3.5.1 Correct Shiftings in Time of demonstrated Trajectories with *Dynamic Time Warping* (DTW)

As introduced earlier in this chapter, Gaussian processes can be used to encode trajectories and to extract constraints using the variability between demonstrations. The quality of the resulting policy depends very much on the demonstration skills of the teacher. In general it is very difficult for a human to repeat the demonstrations with the same velocity and accelerations. As a result there are distortions and shifts in time between trajectories (see Figure 3.5a). Therefore it can be useful to realign the trajectories in time in a preprocessing step.

The *Dynamic Time Warping* algorithm (DTW) is a well-known algorithm in the context of speech recognitions and signal processing. Given two sequences in time (e.g. two trajectories) $\boldsymbol{\xi}_s^A = \{\boldsymbol{\xi}_{s,1}^A, \cdots, \boldsymbol{\xi}_{s,n}^A\}$, $\boldsymbol{\xi}_s^B = \{\boldsymbol{\xi}_{s,1}^B, \cdots, \boldsymbol{\xi}_{s,m}^B\}$, DTW finds a non-linear mapping in time from $\boldsymbol{\xi}_s^B$ to $\boldsymbol{\xi}_s^A$ at the costs of $O(nm)$. This can be easily improved with path constraints as explained later in this chapter. DTW requires the data to be at equal distance in time. If this is not the case, one can define a mapping from the original distance to a new one which fulfills the requirements.[1] In order to align the two sequences, a $n \times m$ matrix $\boldsymbol{C}$, called the *local cost matrix*, is constructed, where $\boldsymbol{C}_{ij}$ contains the distance $d(\boldsymbol{\xi}_{s,i}^A, \boldsymbol{\xi}_{s,j}^B)$ between the points $\boldsymbol{\xi}_{s,i}^A$ and $\boldsymbol{\xi}_{s,j}^B$. In order to simplify notation we define $d(i,j) := d(\boldsymbol{\xi}_{s,i}^A, \boldsymbol{\xi}_{s,j}^B)$. The distance function[2] $d(\cdot,\cdot) \geq 0$ can vary with the application, but most often the Euclidean distance is used. DTW uses the local cost matrix to find the *warping path* $\boldsymbol{W} = \{w_1, w_2, \cdots, w_L\}$ with $w_l = \{u, v\}$ minimizing the overall costs. The warping path must satisfy the following criteria:

1 A straightforward approach would be resampling.

2 Due to the relation between DTW and *Dynamic Programming*, this function is also very often called the *cost function*.



(a) Two demonstrations with time shift **(b)** The cumulative distance function

Figure 3.5: Panel (a): Two demonstrations with a time delay at the beginning and small distortions. The thin gray lines denoting the matches found by DTW. Panel (b): The cumulative distance function with the warping path (black line from lower left to upper right corner) can be seen in the middle of the figure. The sequence on the bottom is matched against the sequence on the left side.

Boundary conditions: $w_1 = (1,1)$ and $w_L = (m,n)$. The start and end point of the warped sequences must be equal to the original.

Monotonicity condition: Given $w_l = (a,b)$, $w_{l-1} = (a',b')$ with $a \geq a'$ and $b \geq b'$. This keeps the time ordering of points.

Continuity condition: Given $w_l = (a,b)$, $w_{l-1} = (a',b')$ with $a - a' \leq 1$ and $b - b' \leq 1$. This prevents the warping path from jumps in time.

The *Dynamic Programming* (DP) algorithm is used to minimize $\sum_{l=1}^{L} w_l$ in order to find an optimal warping path. The so called *cumulative distance function*[1] $\gamma(i,j)$ is defined as the distance of the current cell $d(i,j)$ and the minimum of the cumulative

1 The cumulative distance function is also known as the *value function* in the context of Dynamic Programming.

distances of the neighbor cells:

$$\gamma(1,j) = \sum_{k=1}^{j} d(1,k),\ j \in [1,m] \qquad \text{(first row)} \tag{3.13}$$

$$\gamma(i,1) = \sum_{k=1}^{i} d(k,1),\ i \in [1,n] \qquad \text{(first column)} \tag{3.14}$$

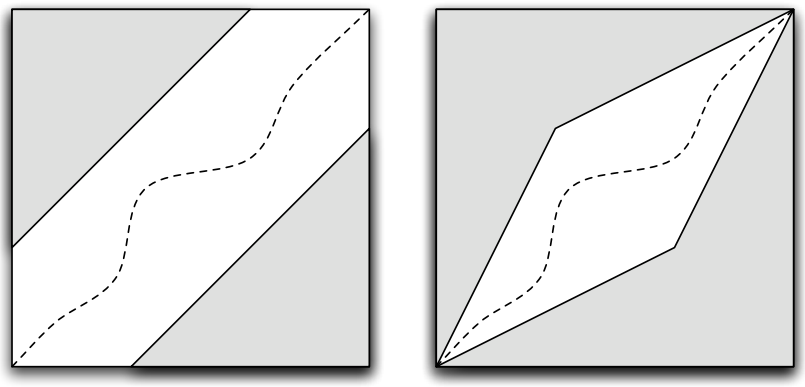
$$\gamma(i,j) = d(i,j) + \min\{\gamma(i-1,j-1), \gamma(i-1,j), \gamma(i,j-1)\} \tag{3.15}$$

An illustration of the cumulative distance function together with a warping path can be seen in Figure 3.5b. The pseudocode for this algorithm is included in section B.2.

There are several improvements that can be made in order to avoid singularities and enhance the computational speed:

Global Path Constraints

The warping path can be restricted to a region near to the diagonal. This is also called *windowing* and had been first introduced by Sakoe and Chiba [Feb., 1978] and Itakura [1975]. An illustration of the two approaches can be seen in Figure 3.6. This restricts the maximum size of a singularity and saves a lot of computations because only the non-shaded regions have to be considered.



(a) Sakoe-Chiba band **(b)** Itakura parallelogram

Figure 3.6: Global path constraints. The warping path is not allowed to enter the shaded regions.

Slope Weighting

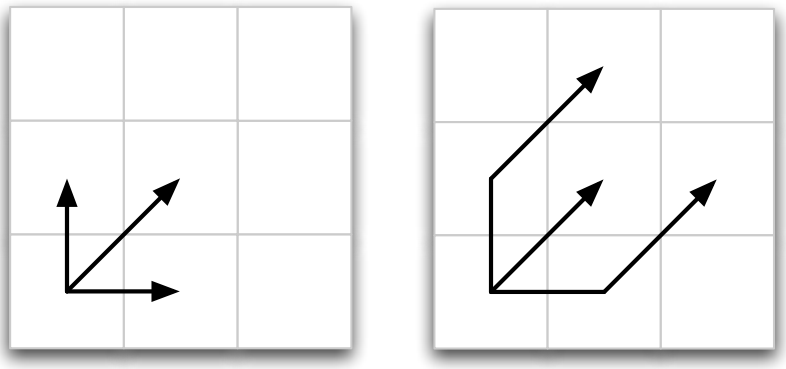
Sometimes it is desirable to favor one step direction over an other. If Equation 3.15 is replaced by

$$\gamma(i,j) = d(i,j) + \min\{\gamma(i-1,j-1), \eta\gamma(i-1,j), \eta\gamma(i,j-1)\} \tag{3.16}$$

then paths along the diagonal will be preferred (for $\eta > 1$) or avoided (for $\eta < 1$). Whereas this is normally sufficient it is also possible to weight each of the three terms in the $\min\{\ldots\}$ expression individually to create an asymmetric preference as described in [Sakoe and Chiba, Feb., 1978].

Slope Constraints

Sometimes DTW finds a warping path with very unnatural correspondences between short segments in one sequence and long segments in the other sequence. This can be avoided with the help of slope constraints. Moving r steps in one direction, the path is not allowed to go further in the same direction before stepping at least q times in the diagonal direction. A straightforward realization can be achieved with so called *step patterns*. Two pattern examples can be seen in Figure 3.7.



(a) Original pattern **(b)** A constraint slope pattern

Figure 3.7: In panel (a): The original unconstrained pattern. In panel (b): For each step parallel to the axis, the path is forced to make one diagonal step.

3.6 System Overview

Figure 3.8 illustrates the interaction between the individual components considering constraints in joint space and constraints with respect to objects at the same time. The *demonstration* module generates the data set for the Gaussian process training in the *policy encoding* module. The *policy encoding* module uses Algorithm 2.1 and Algorithm 3.1. The pseudo code for *optimal controller derivation* as described in the previous sections can be seen in Algorithm 3.2 on the next page. The controller is then used to generate motor commands that are executed by the robot. The *visual object detection* is still in a testing phase and we use also kinesthetic teaching to get the initial positions of the M objects at the moment.

Algorithm 3.2 Constraint Extraction and Reproduction

input: $\hat{\boldsymbol{\theta}}$ ($\mathcal{GP}$ estimate joint space), $\hat{\boldsymbol{\Sigma}}^{\theta}$ (joint space covariances), $\boldsymbol{o}^{(m)}$ (M objects), $\hat{\boldsymbol{x}}^{(m)}$ ($\mathcal{GP}$ estimate relative to objects), $\hat{\boldsymbol{\Sigma}}^{x(m)}$ (covariances relative to objects)
$\alpha = 0.5$ (weight factor for Jacobian null space optimization)
Set initial values by direct kinematics
$\boldsymbol{\theta}_0 = \hat{\boldsymbol{\theta}}_0, \quad \boldsymbol{x}_0 = \mathrm{DK}(\hat{\boldsymbol{\theta}}_0)$
for $t = 0 \rightarrow T$ **do**
    Compute Jacobian matrix, where k is the number of joints and p is the number of elements in the pose vector
    $\boldsymbol{J}(\boldsymbol{\theta}_t) = \left(\frac{\partial x_i}{\partial \theta_{t,j}} \right)_{i,j} \Big| \; i \in [1, \ldots, p], j \in [1, \ldots, k]$
    Compute the pseudoinverse of the Jacobian matrix
    $\boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) = (\boldsymbol{J}(\boldsymbol{\theta}_t)^T \boldsymbol{J}(\boldsymbol{\theta}_t))^{-1} \boldsymbol{J}(\boldsymbol{\theta}_t)^T$
    Approximate dynamics with Δ-values for the M objects
    for $m = 1 \rightarrow M$ **do**
        $\Delta \boldsymbol{x}_{t+1}^{(m)} = \boldsymbol{o}^{(m)} + \hat{\boldsymbol{x}}_{t+1}^{(m)} - \boldsymbol{x}_t$
        $\Delta \boldsymbol{\theta}_{t+1}^{(m)} = \boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \Delta \boldsymbol{x}_{t+1}^{(m)} + \alpha \left(\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \boldsymbol{J}(\boldsymbol{\theta}_t) \right) (\hat{\boldsymbol{\theta}}_{t+1} - \boldsymbol{\theta}_t)$
        $\boldsymbol{\Sigma}_{t+1}^{(m)} = \boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \hat{\boldsymbol{\Sigma}}_{t+1}^{x(m)} \left(\boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \right)^T$
    end for
    Compute posture by gaussian product for
    Joint space only: $\boldsymbol{\theta}_{t+1} = \hat{\boldsymbol{\theta}}_{t+1}$
    Task space only: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \prod_{m=1}^{M} \mathcal{N} \left(\Delta \boldsymbol{\theta}_{t+1}^{(m)}, \boldsymbol{\Sigma}_{t+1}^{(m)} \right)$
    Joint & task space: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \prod_{m=1}^{M} \mathcal{N} \left(\Delta \boldsymbol{\theta}_{t+1}^{(m)}, \boldsymbol{\Sigma}_{t+1}^{(m)} \right) \mathcal{N} \left(\hat{\boldsymbol{\theta}}_{t+1} - \boldsymbol{\theta}_t, \boldsymbol{\Sigma}_{t+1}^{\theta} \right)$
    Update new position by direct kinematic
    $\boldsymbol{x}_{t+1} = \mathrm{DK}(\boldsymbol{\theta}_{t+1})$
end for
return: $\boldsymbol{\theta}$ (trajectory to reproduce)

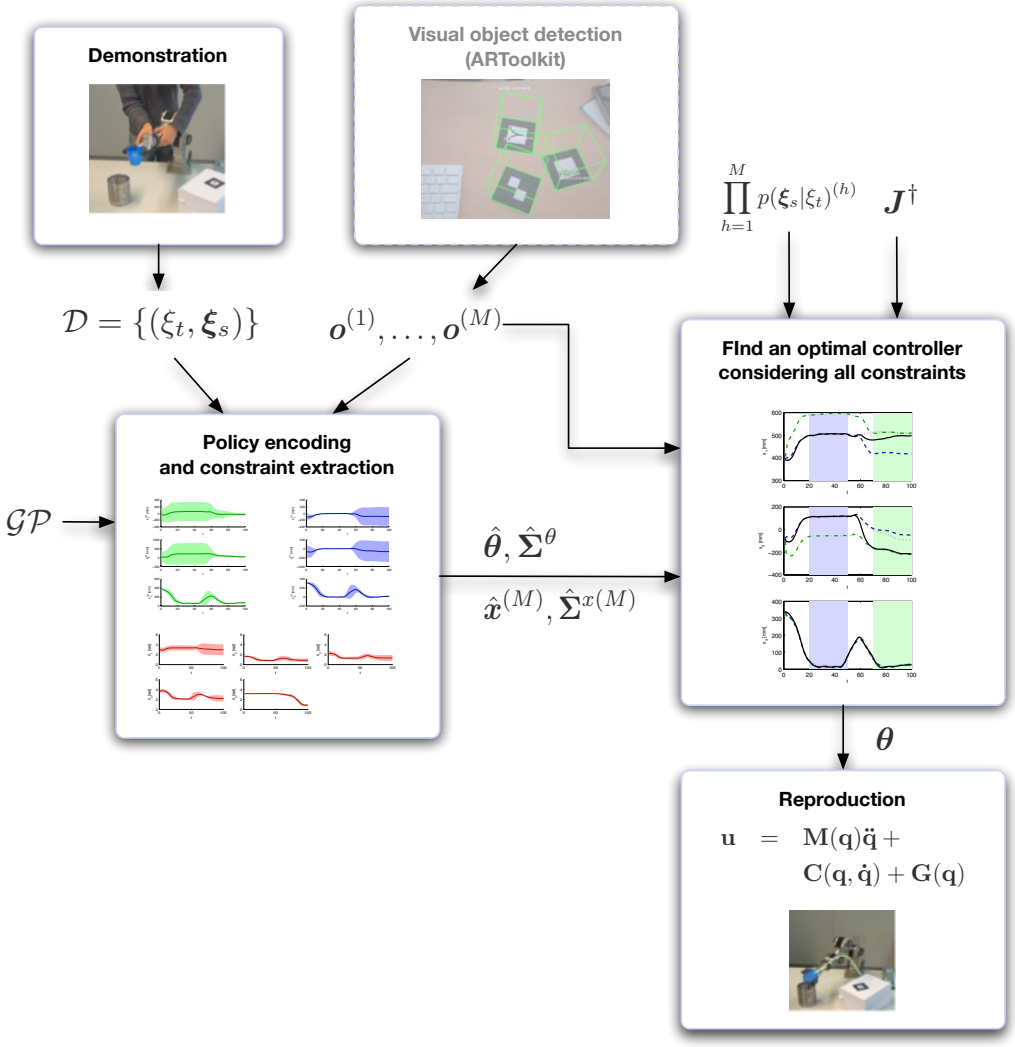


Figure 3.8: An overview of the Learning from Demonstration Framework

CHAPTER 4

Experiments

This chapter presents the experimental setup and the robot used to evaluate the theoretical results from the previous chapters. In the first experiment we proof that Gaussian processes can be used to derive a controller with the necessary generalization properties. In the second experiment the robot has to extract several task constraints from a few demonstrations and must be able to reproduce it in a new situation.

An independent heteroscedastic Gaussian process is used for every dimension in $\boldsymbol{\xi}_s$ with ξ_t as input. We use a *squared exponential* (SE) covariance function with a single length scale parameter multiplied with an additional parameter for the overall process variance. This function is used for both the standard Gaussian process and the noise level process (but with individual parameterization). Even this simple choice of a covariance function is sufficient for the Gaussian process model to achieve good results as shown later. We do not distinguish between joint space, task space and gripper data. All of them are controlled by the same model with automatic parameter optimization. The information extracted from individual models is then merged to an overall policy/controller for the final reproduction step as explained earlier.

4.1 The Katana robot

The experiments were performed on a six degrees of freedom (DOFs) Katana robot from Neuronics[1]. The Katana is a lightweight robot arm with a repeatability of ± 0.1

1 Neuronics Homepage: http://www.neuronics.ch

mm and a maximum speed of 90 degrees per second. All axes are equipped with harmonic drive gears.

The Katana native C++ interface (KNI) was used to program applications for recording and replaying of trajectories. The current implementation uses lists of waypoints to exchange data. Each waypoint is a composition of encoder values. For the *kinesthetic teaching process* all motors are set to *passive mode* which allows the user to freely move the robot arm. During the movement all motor encoder values are recorded at a sampling rate of 100 Hz. We use $\theta_1, \ldots, \theta_5$ to label the five encoders and θ_6 for the gripper. $x_1, \ldots, x_3$ and $x_4, \ldots, x_6$ denote the position and orientation of the end effector. The x, y, z notation is used as well to represent the position in Cartesian coordinates. At present, calculations for the *policy encoding* as well as *constraint extraction* are done in an offline step. The result is transferred back to the Katana afterwards.

Figure 4.1: The Katana robot arm from Neuronics

4.2 Experiment I: The Figure Eight Curve

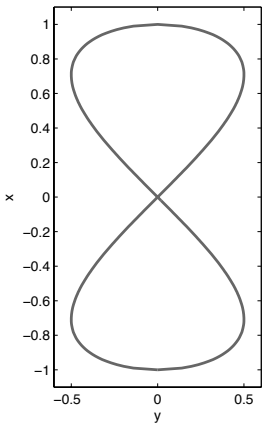
In this experiment the robot arm has to follow the figure eight curve[1], a fourth degree algebraic curve with equation:

$$x^4 - x^2 + y^2 = 0 \tag{4.1}$$

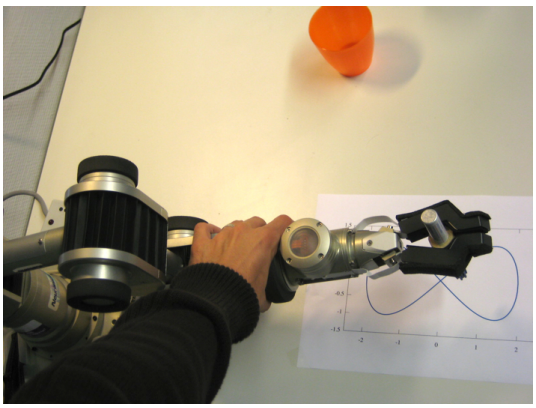
as drawn in Figure 4.2a. In addition to the encoder values, the task space coordinates (the position and orientation of robot's end effector) are calculated through Katana's forward kinematics. In this experiment the task was demonstrated only four times. Even if the robot arm is moved directly, it is very difficult to follow the pattern exactly and with smooth movements. This can also be seen in Figure 4.3 where the demonstrations are plotted in the x-y plane.

The raw data set has been uniformly resampled to $T = 100$ points per demonstration. A separate Gaussian process was used to encode each dimension in joint space as shown in Figure 4.5. This is absolutely sufficient to fulfill the exercise and the encoding in task space is only illustrated for completeness (as in Figure 4.6).

The Gaussian process encoding was able to generalize over multiple demonstrations, to filter out noise introduced by a human teacher and to indicate the confidence



(a) The figure eight curve



(b) kinesthetic teaching

Figure 4.2: Left: The figure eight curve. In this experiment the figure was rotated by 90°. Right: A picture of the kinesthetic teaching process.

1 The figure eight curve is also known as the Lemniscate of Gerono (Camille-Christophe Gerono 1799-1891).

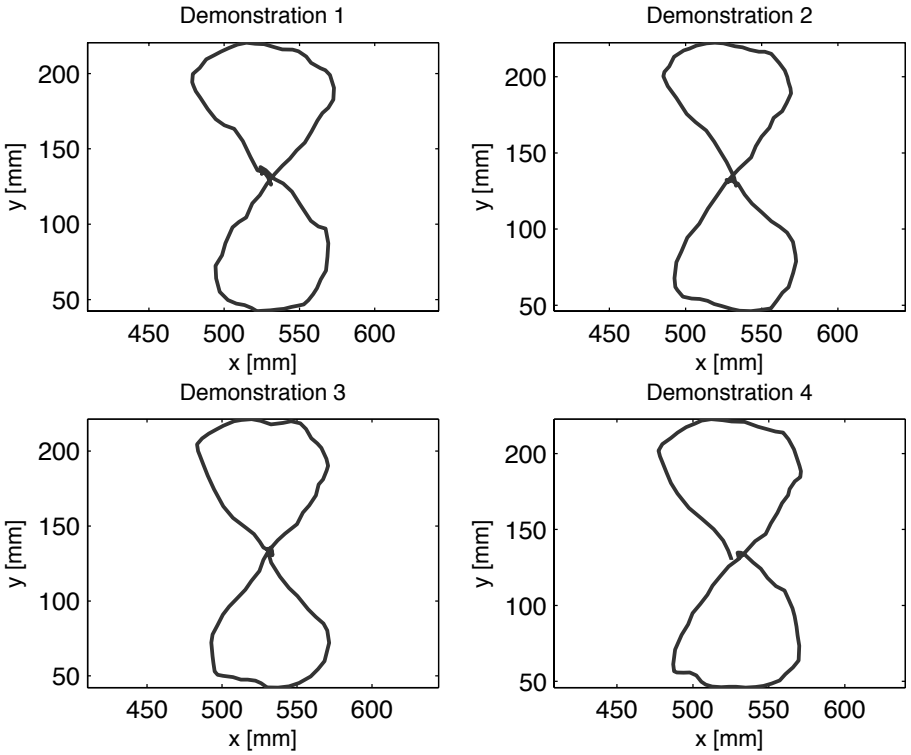


Figure 4.3: The four trajectories demonstrated by kinesthetic teaching.

at each step of the trajectory. Where the last feature mentioned is more important for constrained tasks, it could also be used to give feedback to the teacher which sub-trajectories may need additional demonstrations.

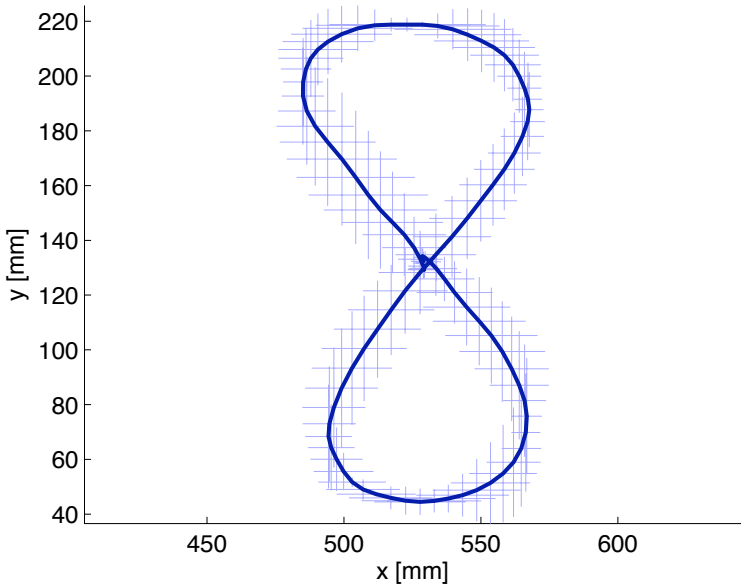


Figure 4.4: The output of the heteroscedastic Gaussian Process. The dark line is the resulting trajectory (the posterior of the GP). The crosses indicating the 95% confidence interval in x and y direction. Note that the result is much closer to the original pattern (especially the corners) and the entire movement is smoother (compared toFigure 4.3).

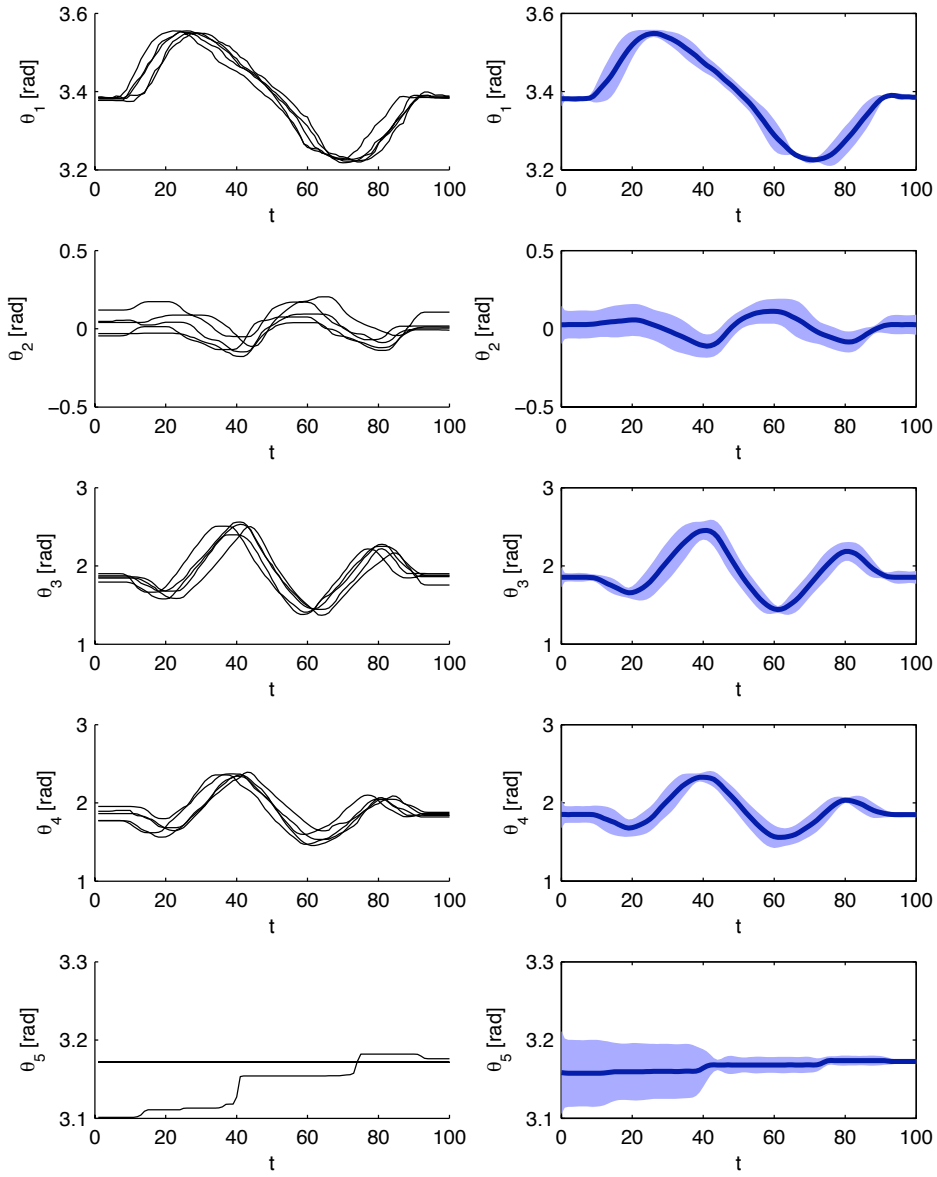


Figure 4.5: The left column shows the input data set for the five joints of the Katana. The GP encoding is plotted in the right column.

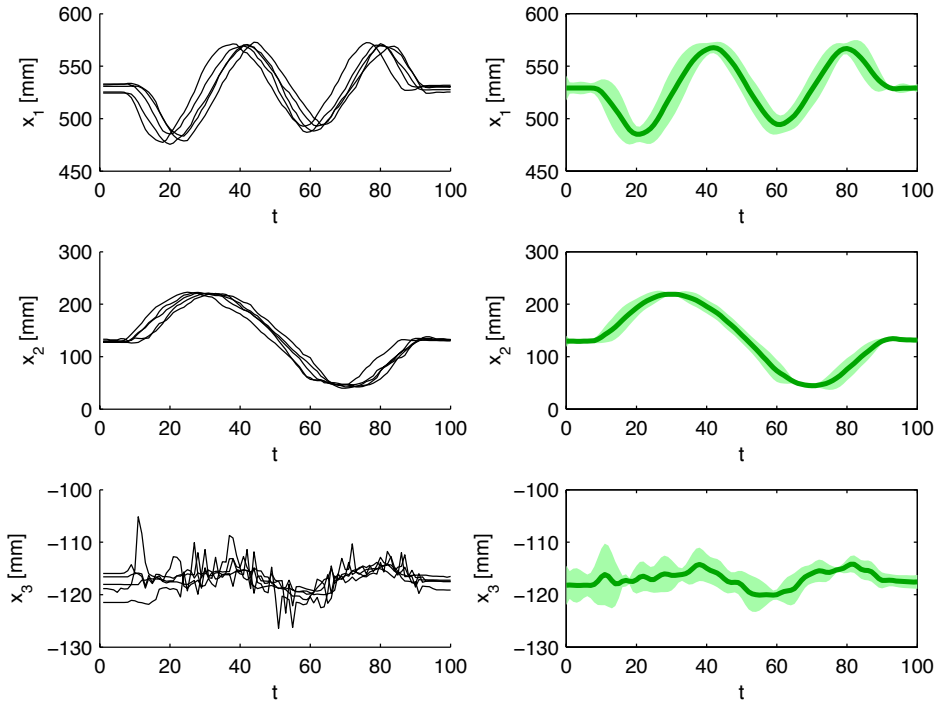


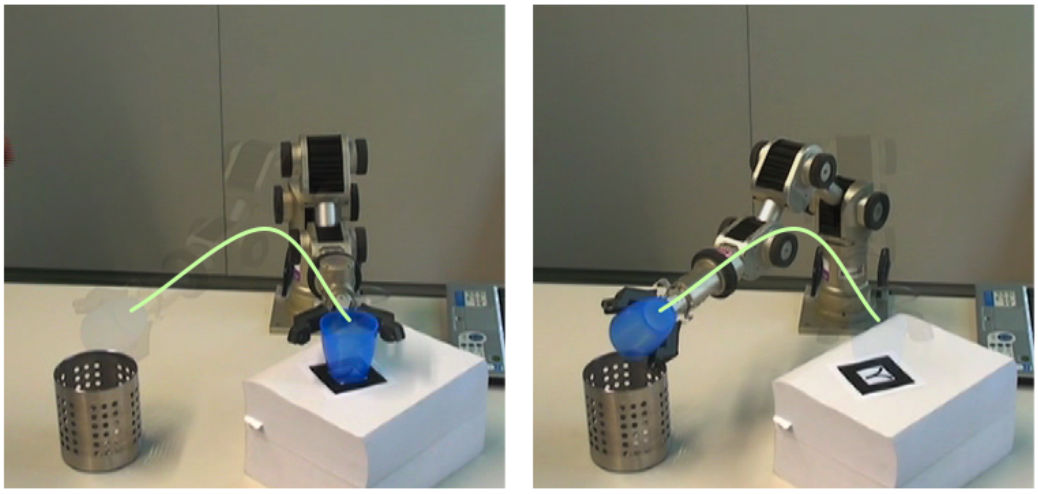
Figure 4.6: The left column shows the input data set in task space. The GP encoding is plotted in the right column. Movements in the z-direction (last row, x_3) are less than 5 mm and a result of imperfect demonstration, not sensor noise.

4.3 Experiment II: Grasping and Emptying

This experiment is composed of two subtasks. First, the robot has to grasp a cup and then unload it into a small trash (see Figure 4.7). This task requires to follow constraints in both, task space and joint space. The constraints in task space are mainly involved in the movements to the cup and trash, whereas the controller has to rotate the fifth joint to empty the cup. Kinesthetic demonstrations are used to gather the trajectory data and the initial positions of the objects.

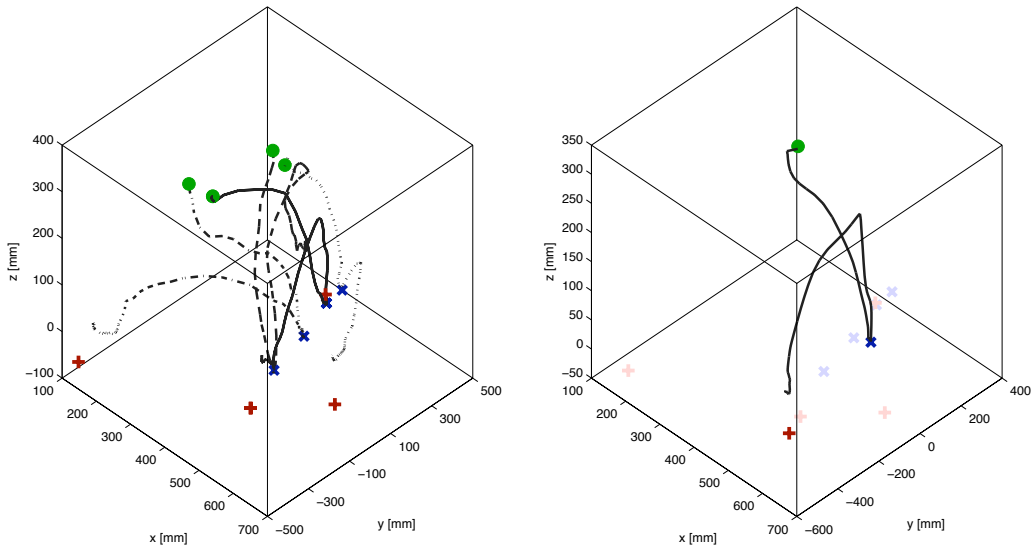
The data set for training consists of four trajectories with nine variables describing the five joint angles of the arm, the gripper and three Cartesian coordinates representing the position of the end effector (calculated by the katana direct kinematics). We resample each trajectory to $T = 100$ data points in a preprocessing step. The initial positions of the two objects are added to the data set at the beginning of each trajectory. We record these positions with kinesthetic teaching at the beginning of the demonstration. The demonstration process in Cartesian coordinates can be seen in Figure 4.8a. The initial position of the cup, the trash and the robot arm end effector are marked with "x", "+" and "o" respectively.

We calculate the relative position of the end effector with respect to each object. As a result, we use 12 Gaussian processes to encode the task (5 for the joints, 1 for the gripper and 2×3 for the movements relative to objects). The policies are illustrated in Figure 4.9 and Figure 4.10.



(a) Subtasks 1 **(b)** Subtasks 2

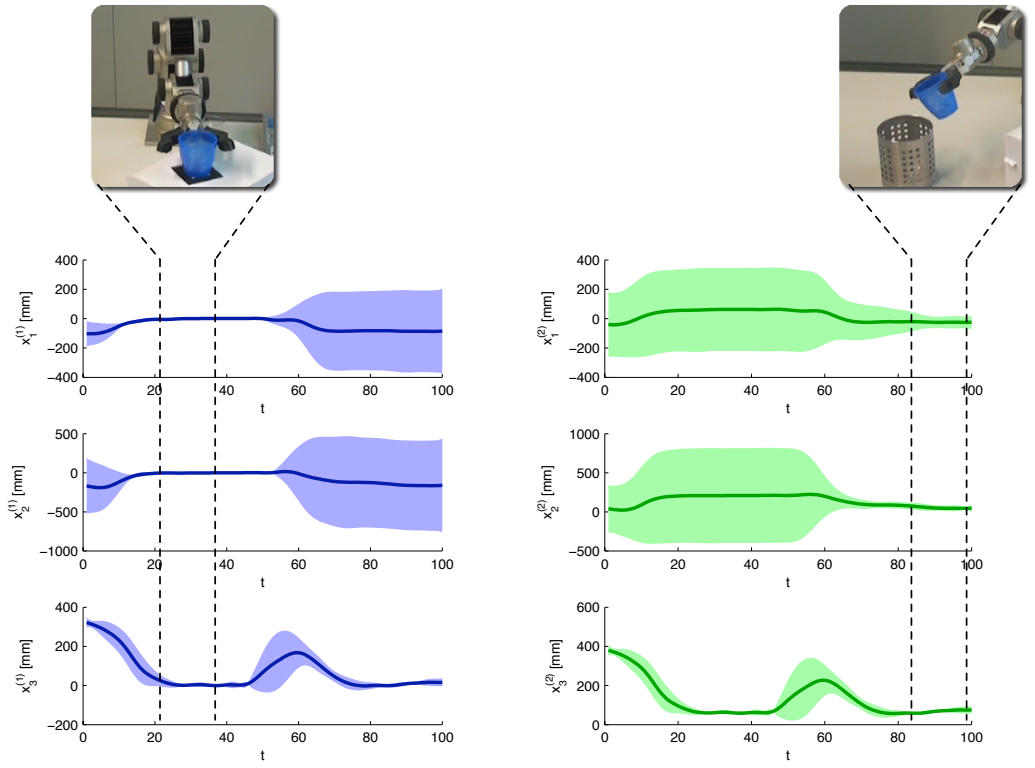
Figure 4.7: The *grasping and emptying* task.



(a) Demonstration **(b)** Reproduction

Figure 4.8: In the left panel: Four demonstrations in 3D Cartesian space. The start states are represented with green circles, the initial positions of the cup and trash are marked with a ”x” and ”+” respectively. In the right panel: Reproduction of the task in a new situation (new position of the objects).

After the learning phase we place the cup and the trash to new positions not included in the training data set as in Figure 4.8b. We first calculate a controller using the policies with respect to the two objects. Then we use the inverse kinematics with optimization in the Jacobian null space (as described in section 3.4) for projection to joint space. Here, the optimization criteria is to stay as close as possible to the joint space policy. The final controller is calculated considering all constraints weighted by their variances. The reproduction process is illustrated in Figure 4.11, decomposed in the steps described above. The results presented here are similar to those obtained by [Calinon and Billard, 2008].



(a) Relative to cup **(b)** Relative to trash

Figure 4.9: The illustration shows the learned policies relative to the initial positions of the cup (a) and the trash (b) in Cartesian coordinates. We can see that the movement relative to the cup is highly constrained between time step 20 and 40. At this time, the end effector reaches the cup and grasp it. In contrast, the movement relative to the trash is constrainted at the end of the trajectory (time step 85 to 100), when the arm is able to unload the cup.

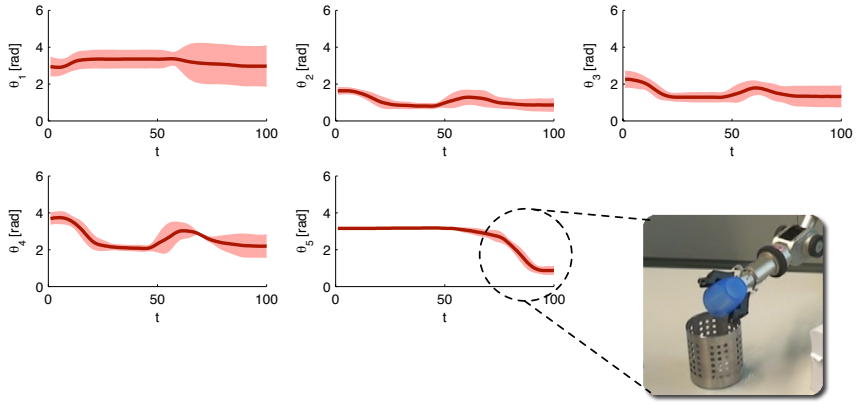
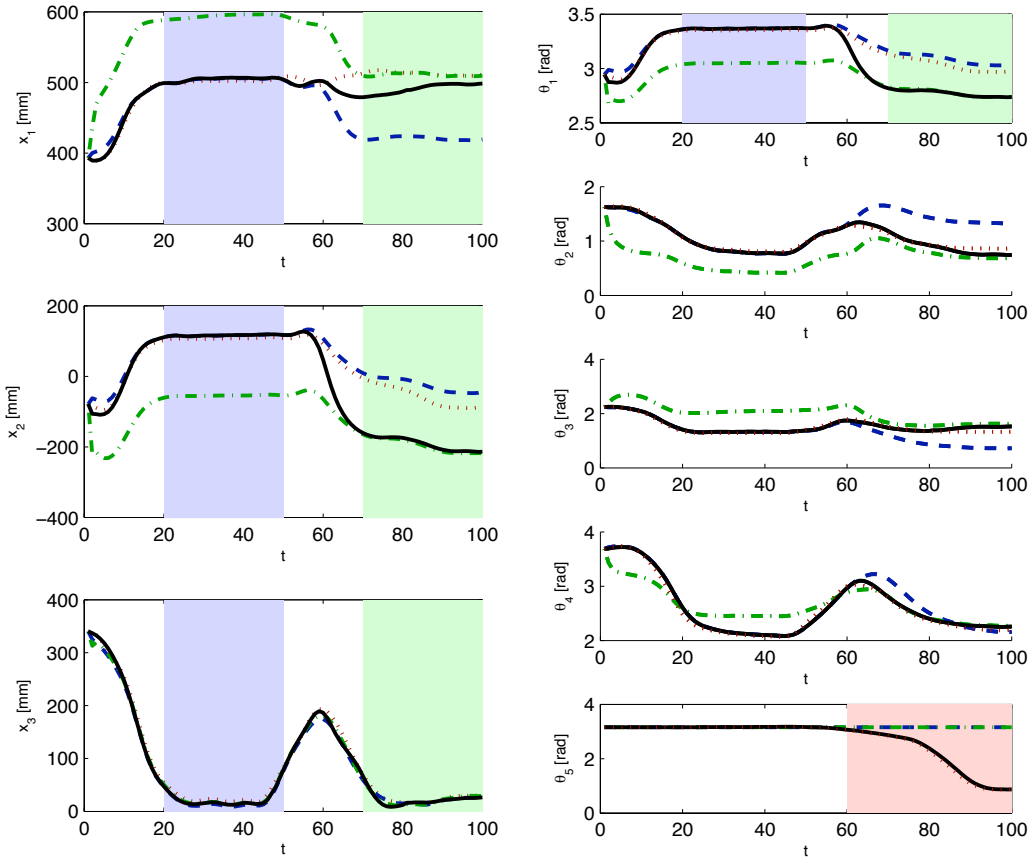


Figure 4.10: The figure shows the learned policy in joint space. It can be seen that the information about the rotation to empty the cup is encoded in θ_5 (which is the last motor on the arm). The cup is horizontal until the end of the trajectory, when the arm reaches the trash.



(a) Reproduction in task space **(b)** Reproduction in joint space

Figure 4.11: This figure shows the reproduction attempts for a new constellation of cup and trash positions. The *dashed blue* line is the policy relative to the cup, the *dash dotted green* line is the policy relative to the trash and the *dotted red* line marks the policy in joint space. The *solid black* line is the overall policy considering all constraints together. The left panel shows the reproduction in task space. We can see that the final controller first follows the constraint for grasping the cup (shaded region between 20 and 50) and then follows the constraint for moving to the trash (shaded region between 70 and 100). The reproduction in joint space is drawn in the right panel. The rotation to empty the cup can be seen in the plot for the last joint θ_5. We can see that the policies relative to the object do not deliver any information about the unload action and the controller has to follow the policy in joint space. The first joint θ_1 has a tremendous influence on the position and therefore is mainly controlled by the policies relative to the objects.

CHAPTER 5

Conclusion & Future Work

In conclusion, we have accomplished the following in this thesis:

1. We showed how *Gaussian processes* (GP) can be used in a *Learning from Demonstration* (LfD) framework to encode movements of a robot.
2. We demonstrated that the GP encoding is able to generalize over multiple demonstrations to create smooth and continuous trajectories.
3. We showed how the probabilistic properties of Gaussian processes can be used to extract so called *constraints* of a task and how multiple constraints can be combined.
4. We successfully evaluated the theoretical results on two experiments with a six DOF *Katana* robot arm.

There are several interesting projects we want to investigate after this thesis:

1. Using Reinforcement Learning (RL) to equip the robot with self-improvement capabilities. A very promising approach seems to be the *Natural Actor Critic* (NAC) algorithm introduced by Peters and Schaal [2008]. The idea is to calculate the gradient in the Gaussian process function space view.
2. Using *Principal Component Analysis* (PCA), *Independent Component Analysis* (ICA) or similar techniques to reduce the dimensionality of the input space.
3. Using a sparse representation of Gaussian processes. One possible implementation could be the *sparse pseudo-input Gaussian process* (SPGP) introduced by Snelson and Ghahramani [2006a].
4. Using time implicitly instead of having it represented as an explicit variable. This could be achieved by taking the system dynamics into account as, for example, in [Hersch et al., 2008].

APPENDIX A

Mathematical Background

A.1 Matrix Analysis

A.1.1 Matrix Identities

For a square invertible matrix $\boldsymbol{\Sigma}$ let

$$\Sigma = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{A}} & \tilde{\boldsymbol{B}} \\ \tilde{\boldsymbol{C}} & \tilde{\boldsymbol{D}} \end{bmatrix}^{-1} \tag{A.1}$$

the partition of $\boldsymbol{\Sigma}$. Then

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} = \begin{bmatrix} (\tilde{\boldsymbol{A}} - \tilde{\boldsymbol{B}}\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{C}})^{-1} & -(\tilde{\boldsymbol{A}} - \tilde{\boldsymbol{B}}\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{C}}^{-1})\tilde{\boldsymbol{B}}\tilde{\boldsymbol{D}}^{-1} \\ -\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{C}}(\tilde{\boldsymbol{A}} - \tilde{\boldsymbol{B}}\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{C}})^{-1} & (\tilde{\boldsymbol{D}} - \tilde{\boldsymbol{C}}\tilde{\boldsymbol{A}}^{-1}\tilde{\boldsymbol{B}})^{-1} \end{bmatrix} \tag{A.2}$$

and

$$\begin{bmatrix} \tilde{\boldsymbol{A}} & \tilde{\boldsymbol{B}} \\ \tilde{\boldsymbol{C}} & \tilde{\boldsymbol{D}} \end{bmatrix} = \begin{bmatrix} (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1} & -(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1}\boldsymbol{B}\boldsymbol{D}^{-1} \\ -\boldsymbol{D}^{-1}\boldsymbol{C}(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1} & (\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1} \end{bmatrix} \tag{A.3}$$

is called the *matrix inversion lemma*. See [Woodbury, 1950] for proof and further details.

A.1.2 The Trace

The trace of a squared matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is denoted by $\operatorname{tr}(\boldsymbol{A})$ (or $\operatorname{tr} \boldsymbol{A}$ if it is obvious form the context) and is the sum of the diagonal elements in the matrix:

$$\operatorname{tr}(\boldsymbol{A}) = \sum_{i=1}^{n} \boldsymbol{A}_{ii} \tag{A.4}$$

For matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ such that their products are square, we also have

$$\operatorname{tr}(\boldsymbol{ABC}) = \operatorname{tr}(\boldsymbol{CAB}) = \operatorname{tr}(\boldsymbol{BCA}). \tag{A.5}$$

Also the following properties of the trace operator are very useful:

$$\operatorname{tr}(\boldsymbol{A}) = \operatorname{tr}(\boldsymbol{A}^T) \tag{A.6}$$
$$\operatorname{tr}(\boldsymbol{A} + \boldsymbol{B}) = \operatorname{tr}(\boldsymbol{A}) + \operatorname{tr}(\boldsymbol{B}) \tag{A.7}$$
$$\operatorname{tr}(a\boldsymbol{A}) = a \operatorname{tr}(\boldsymbol{A}) \tag{A.8}$$

For vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ the relationship between the trace operator and the dot product is

$$\boldsymbol{x}^T \boldsymbol{y} = \operatorname{tr}(\boldsymbol{x}\boldsymbol{y}^T) \tag{A.9}$$

A.1.3 Matrix Derivatives

The derivative of a matrix $\boldsymbol{A}$ with respect to a scalar is defined element-wise as

$$\left(\frac{\partial \boldsymbol{A}}{\partial x}\right)_{ij} = \frac{\partial \boldsymbol{A}_{ij}}{\partial x}. \tag{A.10}$$

Some further facts of matrix derivatives with respect to scalar values:

$$\frac{\partial \operatorname{tr}(\boldsymbol{A})}{\partial x} = \operatorname{tr}\left(\frac{\partial \boldsymbol{A}}{\partial x}\right) \tag{A.11}$$
$$\frac{\partial \boldsymbol{A}^{-1}}{\partial x} = -\boldsymbol{A}^{-1} \frac{\partial \boldsymbol{A}}{\partial x} \boldsymbol{A}^{-1} \tag{A.12}$$
$$\frac{\partial |\boldsymbol{A}|}{\partial x} = |\boldsymbol{A}| \operatorname{tr}\left(\boldsymbol{A}^{-1} \frac{\partial \boldsymbol{A}}{\partial x}\right) \tag{A.13}$$
$$\frac{\partial \ln |\boldsymbol{A}|}{\partial x} = \operatorname{tr}\left(\boldsymbol{A}^{-1} \frac{\partial \boldsymbol{A}}{\partial x}\right) \tag{A.14}$$

and matrices:

$$\frac{\partial \operatorname{tr}(\boldsymbol{AB})}{\partial \boldsymbol{A}} = \boldsymbol{B}^T \tag{A.15}$$

$$\frac{\partial \operatorname{tr}(\boldsymbol{ABA}^T\boldsymbol{C})}{\partial \boldsymbol{A}} = \boldsymbol{CAB} + \boldsymbol{C}^T\boldsymbol{AB}^T \tag{A.16}$$

$$\frac{\partial \ln|\boldsymbol{A}|}{\partial \boldsymbol{A}} = \left(\boldsymbol{A}^T\right)^{-1} \tag{A.17}$$

A.1.4 Symmetric Positive Definite Matrices

Definition A.1.1. *A quadratic matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is called a symmetric matrix if $\boldsymbol{A} = \boldsymbol{A}^T$.*

Definition A.1.2. *A symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is*

- *positive definite, if $\boldsymbol{x}^T\boldsymbol{Ax} > 0$,*
- *negative definite, if $\boldsymbol{x}^T\boldsymbol{Ax} < 0$,*
- *positive semidefinite, if $\boldsymbol{x}^T\boldsymbol{Ax} \geq 0$,*
- *positive semidefinite, if $\boldsymbol{x}^T\boldsymbol{Ax} \leq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n \setminus \{0\}$.*
- *$\boldsymbol{A}$ is indefinit, if there is an $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ with $\boldsymbol{x}^T\boldsymbol{Ax} > 0$ and $\boldsymbol{y}^T\boldsymbol{Ay} < 0$.*

The space of symmetric positive definite $(n \times n)$ matrices will be denoted by $\mathbf{S}^n_{++}$.

A.1.5 Cholesky Decomposition

The Cholesky decomposition of a symmetric positive definite matrix $\boldsymbol{A}$ decomposes $\boldsymbol{A}$ into

$$\boldsymbol{A} = \boldsymbol{LL}^T, \tag{A.18}$$

where the so called Cholesky factor $\boldsymbol{L}$ is a lower triangular matrix. This can easily be verified:

Proof. Assuming a symmetric positive definite matrix $\boldsymbol{A} \in \mathbf{S}^n_{++}$. This matrix can be written as

$$\boldsymbol{A} = \boldsymbol{U}^T\boldsymbol{DU} \tag{A.19}$$

where $\boldsymbol{D}$ is a diagonal matrix with all positive eigenvalues and $\boldsymbol{U}$ is an upper triangular matrix. Since the matrix $\boldsymbol{A}$ is symmetric positive definite, we can therefore write

$$\boldsymbol{A} = \boldsymbol{U}^T \boldsymbol{D} \boldsymbol{U} \tag{A.20}$$

$$= (\boldsymbol{U}^T \sqrt{\boldsymbol{D}})(\sqrt{\boldsymbol{D}} \boldsymbol{U}) \tag{A.21}$$

$$= (\sqrt{\boldsymbol{D}} \boldsymbol{U})^T (\sqrt{\boldsymbol{D}} \boldsymbol{U}) \tag{A.22}$$

The matrix $\boldsymbol{L} = (\sqrt{\boldsymbol{D}} \boldsymbol{U})$ therefore satisfies $\boldsymbol{L}^T \boldsymbol{L} = \boldsymbol{A}$. □

The Cholesky decomposition can be useful to compute the inverse $\boldsymbol{A}^{-1}$ and its determinant $|\boldsymbol{A}|$ as

$$\boldsymbol{A}^{-1} = (\boldsymbol{L}^{-1})^T \boldsymbol{L}^{-1}, \tag{A.23}$$

$$|\boldsymbol{A}| = \prod_{i=1}^{n} \boldsymbol{L}_{ii}^2 \quad or \quad \log|\boldsymbol{A}| = 2 \sum_{i=1}^{n} \log \boldsymbol{L}_{ii}. \tag{A.24}$$

A.2 Probability Theory Review

In the following the probability of the vector of the n joint random variables $x_1, \ldots, x_n$ will be denoted as $p(\boldsymbol{x})$.

A.2.1 Rules of Probability

The *marginal* probability function is given by

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{y} \tag{A.25}$$

The *conditional* probability function is

$$p(\boldsymbol{x}|\boldsymbol{y}) = \frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{y})} \tag{A.26}$$

From these two functions *Bayes's theorem* can be obtained

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y})}{p(\boldsymbol{x})} \tag{A.27}$$

$$= \frac{p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y})}{\int_{\boldsymbol{y}} p(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{y}} \tag{A.28}$$

A.2.2 Expectations and covariances

The *expectation* of some function $f(\boldsymbol{x})$ under a probability distribution $p(\boldsymbol{x})$ is given by

$$\mathbb{E}[f] = \int p(\boldsymbol{x}) f(\boldsymbol{x})\, d\boldsymbol{x} \tag{A.29}$$

and the *variance* of $f(\boldsymbol{x})$ is defined by

$$\mathbb{V}[f] = \mathbb{E}[(f(\boldsymbol{x}) - \mathbb{E}[f(\boldsymbol{x})])^2] \tag{A.30}$$

$$= \mathbb{E}[f(\boldsymbol{x})^2] - \mathbb{E}[f(\boldsymbol{x})]^2. \tag{A.31}$$

For two random variables $\boldsymbol{x}$ and $\boldsymbol{y}$, the *covaricance* is defined by

$$\operatorname{cov}[\boldsymbol{x}, \boldsymbol{y}] = \mathbb{E}[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{y} - \mathbb{E}[\boldsymbol{y}])^T] \tag{A.32}$$

$$= \mathbb{E}[\boldsymbol{x}\boldsymbol{y}^T] - \mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{y}^T]. \tag{A.33}$$

A.3 Multivariate Gaussians

A multivariate Gaussian distribution has the probability density given by

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{A.34}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is a vector-valued random variable with mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and a covariance matrix $\boldsymbol{\Sigma} \in \mathbf{S}^n_{++}$ (a symmetric, positive definite $n \times n$ matrix). This is written as $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

A.3.1 The "completion of squares"

The following mathematical trick is called the "completion of squares", where $\boldsymbol{A}$ is a symmetric matrix, $\boldsymbol{z}, \boldsymbol{b}$ are vectors and c is a constant.

$$-\frac{1}{2}\boldsymbol{z}^T\boldsymbol{A}\boldsymbol{z} - \boldsymbol{z}^T\boldsymbol{b} - c = -\frac{1}{2}\boldsymbol{z}^T\boldsymbol{A}\boldsymbol{z} - \boldsymbol{z}^T\boldsymbol{A}\boldsymbol{A}^{-1}\boldsymbol{b} - c \tag{A.35}$$

$$= -\frac{1}{2}\boldsymbol{z}^T\boldsymbol{A}\boldsymbol{z} - \frac{1}{2}\boldsymbol{z}^T\boldsymbol{A}\boldsymbol{A}^{-1}\boldsymbol{b} - \frac{1}{2}(\boldsymbol{A}^{-1}\boldsymbol{b})^T\boldsymbol{A}^T\boldsymbol{z}^T - c \tag{A.36}$$

$$= -\frac{1}{2}(\boldsymbol{z}-\boldsymbol{A}^{-1}\boldsymbol{b})^T\boldsymbol{A}(\boldsymbol{z}-\boldsymbol{A}^{-1}\boldsymbol{b}) + \frac{1}{2}\boldsymbol{b}^T\boldsymbol{A}^{-1}\boldsymbol{b} - c. \tag{A.37}$$

It is very useful in the context of Gaussian distributions. Given the quadratic exponent term of a Gaussian the mean and covariance function can be calculated easily.

A.3.2 Conditional of a joint Gaussian

The *conditional* of a joint Gaussian distribution is again a Gaussian distribution. Let $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ be jointly Gaussian random vectors:

$$\begin{bmatrix}\boldsymbol{x}_a\\ \boldsymbol{x}_b\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_a\\ \boldsymbol{\mu}_b\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{bb}\\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_a\\ \boldsymbol{\mu}_b\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{bb}\\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb}\end{bmatrix}^{-1}\right) \tag{A.38}$$

where $\boldsymbol{x}_a \in \mathbb{R}^m$, $\boldsymbol{x}_b \in \mathbb{R}^n$ and the dimensions of the mean vectors and covariance submatrizes are chosen respectively. Then the mean and covariance of the conditional Gaussian distribution $p(\boldsymbol{x}_a|\boldsymbol{x}_b)$ is denoted by $\boldsymbol{\mu}_{a|b}$ and $\boldsymbol{\Sigma}_{a|b}$ respectively and

$$\boldsymbol{x}_a|\boldsymbol{x}_b \sim \mathcal{N}(\boldsymbol{\mu}_{a|b}, \boldsymbol{\Sigma}_{a|b}) \tag{A.39}$$

with

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\boldsymbol{x}_b - \boldsymbol{\mu}_b) \tag{A.40}$$

$$\boldsymbol{\Sigma}_{a|b} = \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba}. \tag{A.41}$$

Proof. The quadratic exponent from Equation A.38 is given by

$$\begin{aligned}-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) = \\ -\frac{1}{2}(\boldsymbol{x}_a-\boldsymbol{\mu}_a)^T\boldsymbol{\Lambda}_{aa}(\boldsymbol{x}_b-\boldsymbol{\mu}_a) - \frac{1}{2}(\boldsymbol{x}_a-\boldsymbol{\mu}_a)^T\boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b-\boldsymbol{\mu}_b) \\ -\frac{1}{2}(\boldsymbol{x}_b-\boldsymbol{\mu}_b)^T\boldsymbol{\Lambda}_{ba}(\boldsymbol{x}_a-\boldsymbol{\mu}_a) - \frac{1}{2}(\boldsymbol{x}_b-\boldsymbol{\mu}_b)^T\boldsymbol{\Lambda}_{bb}(\boldsymbol{x}_b-\boldsymbol{\mu}_b)\end{aligned} \tag{A.42}$$

Now, recalling the results from Equation A.35 with $\boldsymbol{z} = \boldsymbol{x}$, $\boldsymbol{A} = \boldsymbol{\Sigma}^{-1}$, $\boldsymbol{\mu} = \boldsymbol{A}^{-1}\boldsymbol{b}$ and $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{\mu}$, we get

$$-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) = -\frac{1}{2}\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}. \tag{A.43}$$

Applying this to Equation A.42 and picking all terms depending on $\boldsymbol{x}_a$, we get

$$= -\frac{1}{2}\boldsymbol{x}_a^T\boldsymbol{\Lambda}_{aa}\boldsymbol{x}_a + \boldsymbol{x}_a^T\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a - \boldsymbol{x}_a^T\boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b-\boldsymbol{\mu}_b) + const \tag{A.44}$$

$$= -\frac{1}{2}\boldsymbol{x}_a^T\boldsymbol{\Lambda}_{aa}\boldsymbol{x}_a + \boldsymbol{x}_a^T(\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b-\boldsymbol{\mu}_b)) + const \tag{A.45}$$

where *const* is a collection of all terms not depending on $\boldsymbol{x}_a$; exploiting the fact that $\boldsymbol{\Lambda}_{ba}^T = \boldsymbol{\Lambda}_{ab}$. The left hand side of Equation A.43 tells us that $\boldsymbol{\Sigma}_{a|b}$ must be equal to

$\boldsymbol{\Lambda}_{aa}^{-1}$ and $\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b - \boldsymbol{\mu}_b)$ must be equal to $\boldsymbol{\Sigma}_{a|b}^{-1}\boldsymbol{\mu}_{a|b}$ and hence

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\Sigma}_{a|b}(\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b - \boldsymbol{\mu}_b)) \tag{A.46}$$

$$= \boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{aa}^{-1}\boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b - \boldsymbol{\mu}_b). \tag{A.47}$$

Using the matrix inversion lemma from Equation A.3, we end up with

$$\boldsymbol{\Lambda}_{aa} = (\boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba})^{-1} \tag{A.48}$$

$$\boldsymbol{\Lambda}_{ab} = -(\boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba})^{-1}\boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1} \tag{A.49}$$

and therefore

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\boldsymbol{x}_b - \boldsymbol{\mu}_b) \tag{A.50}$$

$$\boldsymbol{\Sigma}_{a|b} = \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba}. \tag{A.51}$$

□

A.3.3 Marginal of a joint Gaussian

The *marginal* of a joint Gaussian distribution is again a Gaussian distribution. Let $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ be jointly Gaussian random vectors:

$$\begin{bmatrix}\boldsymbol{x}_a\\ \boldsymbol{x}_b\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_a\\ \boldsymbol{\mu}_b\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{bb}\\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_a\\ \boldsymbol{\mu}_b\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{bb}\\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb}\end{bmatrix}^{-1}\right) \tag{A.52}$$

where $\boldsymbol{x}_a \in \mathbb{R}^m$, $\boldsymbol{x}_b \in \mathbb{R}^n$ and the dimensions of the mean vectors and covariance sub matrices are chosen respectively. Then the marginal distribution of $\boldsymbol{\mu}_a$ is

$$\boldsymbol{x}_a \sim \mathcal{N}(\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_{aa}). \tag{A.53}$$

Proof. The marginal distribution $p(\boldsymbol{x}_a)$ over $p(\boldsymbol{x}_a, \boldsymbol{x}_b)$ is defined as

$$p(\boldsymbol{x}_a) = \int p(\boldsymbol{x}_a, \boldsymbol{x}_b)\, d\boldsymbol{x}_b, \tag{A.54}$$

which is a multidimensional integral. Again the ”completion of squares” will be used to avoid the direct integration. The results from Equation A.45 are used, but now considering only the terms depending on $\boldsymbol{x}_b$ (simply switch $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$). Then

$$-\frac{1}{2}\boldsymbol{x}_b^T\boldsymbol{\Lambda}_{bb}\boldsymbol{x}_b + \boldsymbol{x}_b^T(\boldsymbol{\Lambda}_{bb}\boldsymbol{\mu}_b - \boldsymbol{\Lambda}_{ba}(\boldsymbol{x}_a - \boldsymbol{\mu}_a)) + \boldsymbol{c} \tag{A.55}$$

where

$$\boldsymbol{c} = -\frac{1}{2}\boldsymbol{x}_a^T\boldsymbol{\Lambda}_{aa}\boldsymbol{x}_a + \boldsymbol{x}_a^T(\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a + \boldsymbol{\Lambda}_{ab}\boldsymbol{\mu}_b) + const, \tag{A.56}$$

with *const* denoting quantities independent of $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$. Using Equation A.35 with $\boldsymbol{A} = \boldsymbol{\Lambda}_{bb}$ and $\boldsymbol{b} = \boldsymbol{\Lambda}_{bb}\boldsymbol{\mu}_b - \boldsymbol{\Lambda}_{ba}(\boldsymbol{x}_a - \boldsymbol{\mu}_a)$ we can obtain

$$-\frac{1}{2}\boldsymbol{x}_b^T\boldsymbol{\Lambda}_{bb}\boldsymbol{x}_b + \boldsymbol{x}_b^T\boldsymbol{b} + \boldsymbol{c} = -\frac{1}{2}(\boldsymbol{x}_b - \boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{b})^T\boldsymbol{\Lambda}_{bb}(\boldsymbol{x}_b - \boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{b}) + \frac{1}{2}\boldsymbol{b}^T\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{b} + \boldsymbol{c}. \tag{A.57}$$

In order to take the integral in Equation A.54 of the exponential of Equation A.57, the last term on the right hand side does not depend on $\boldsymbol{x}_b$ and therefore

$$p(\boldsymbol{x}_a) = z\exp\left(\frac{1}{2}\boldsymbol{b}^T\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{b} + \boldsymbol{c}\right)\int\exp\left(-\frac{1}{2}(\boldsymbol{x}_b - \boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{b})^T\boldsymbol{\Lambda}_{bb}(\boldsymbol{x}_b - \boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{b})\right)\, d\boldsymbol{x}_b, \tag{A.58}$$

where z is the normalization constant of the joint Gaussian $p(\boldsymbol{x}_a, \boldsymbol{x}_b)$. It can easily be seen that the integral is over an unnormalized Gaussian and so the result of this will

be just another normalization constant.[1] So the only term depending on $\boldsymbol{x}$ remaining is the one outside the integral. Substituting $\boldsymbol{b}$ and $\boldsymbol{c}$ again, we have

$$\begin{aligned}
&\frac{1}{2}(\boldsymbol{\Lambda}_{bb}\boldsymbol{\mu}_b - \boldsymbol{\Lambda}_{ba}(\boldsymbol{x}_a - \boldsymbol{\mu}_a))^T \boldsymbol{\Lambda}_{bb}^{-1}(\boldsymbol{\Lambda}_{bb}\boldsymbol{\mu}_b - \boldsymbol{\Lambda}_{ba}(\boldsymbol{x}_a - \boldsymbol{\mu}_a)) \\
&\quad - \frac{1}{2}\boldsymbol{x}_a^T\boldsymbol{\Lambda}_{aa}\boldsymbol{x}_a + \boldsymbol{x}_a^T(\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a + \boldsymbol{\Lambda}_{ab}\boldsymbol{\mu}_b) + const \qquad \text{(A.59)} \\
&\quad = -\frac{1}{2}\boldsymbol{x}_a^T(\boldsymbol{\Lambda}_{aa} - \boldsymbol{\Lambda}_{ab}\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{\Lambda}_{ba})\boldsymbol{x}_a + \boldsymbol{x}_a^T(\boldsymbol{\Lambda}_{aa} - \boldsymbol{\Lambda}_{ab}\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{\Lambda}_{ba})\boldsymbol{\mu}_a + const.
\end{aligned}$$

Comparing this to Equation A.43 and using the results from the matrix inversion lemma from Equation A.3 we see, that the covariance matrix is $(\boldsymbol{\Lambda}_{aa} - \boldsymbol{\Lambda}_{ab}\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{\Lambda}_{ba})^{-1}$ and the mean is $\boldsymbol{\mu}_a$. □

1 Because the Gaussian is a probability distribution, it must integrate up to 1.

A.3.4 Generating Samples from Multivariate Gaussians

The Problem is to generate samples $\boldsymbol{x} = (x_1, \ldots, x_2)^T$ where $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is Gaussian distributed with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.

Suppose $u_i \sim \mathcal{N}(0,1)$[1] and independent, identically distributed (i.i.d.) for $i = 1, \ldots, n$. Any linear combination of the u_i's is again normal distributed with

$$c_1 u_1 + \cdots + c_n u_n \sim \mathcal{N}(0, c_1^2 + \cdots + c_n^2) \tag{A.60}$$

Defining $\boldsymbol{u} = (u_1, \ldots, u_n)^T$ and $\boldsymbol{L}$ be a $(n \times m)$ matrix containing all constants

$$\boldsymbol{L} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix} \tag{A.61}$$

then

$$\boldsymbol{L}^T \boldsymbol{u} \sim \mathcal{N}(0, \boldsymbol{L}^T \boldsymbol{L}). \tag{A.62}$$

That means we only have to find a Matrix $\boldsymbol{L}$ such that

$$\boldsymbol{L}^T \boldsymbol{L} = \boldsymbol{\Sigma}. \tag{A.63}$$

Fortunately any covariance matrix is symmetric positive definite and we can use cholesky decomposition (see subsection A.1.5) to do this.

Algorithm A.1 Pseudocode for generating samples from multivariate Gaussians

1: **input:** $\boldsymbol{\mu}$ (mean vector), $\boldsymbol{\Sigma}$ (covariance matrix)
2: $\boldsymbol{L}$ = CHOLESKY($\boldsymbol{\Sigma}$)
3: $\boldsymbol{u} = (\mathcal{N}(0,1)_1, \ldots, \mathcal{N}(0,1)_n)^T$
4: $\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{L}^T \boldsymbol{u}$
5: **return:** $\boldsymbol{x}$ (multivariate Gaussian sample)

1 Assuming a scalar Gaussian random number generator, which is available in nearly every programming language

APPENDIX B

Additional Code and Algorithms

B.1 Katana Kinematics

```
% This function creates a symbolic representation of the
% pose function (direkt kinematics) and the Jacobian matrix
% for a Katana6M180

syms t t1 t2 t3 t4 t5
syms pose J
t = [t1;t2;t3;t4;t5];

%% Katana 6M180 configuration
L1=190.0;
L2=139.0;
L3=147.3;
L4=166.0;

%% correct encoder values
t2 = t2 - pi/2;
t3 = t3 - pi;
t4 = pi - t4;
t5 = -t5;
t3 = t2+t3;
t4 = t3+t4;

%% calculation
factor = L1.*sin(t2) + L2.*sin(t3) + (L3+L4).*sin(t4);

% compare homogenous transformation matrix
pose(1,:) = cos(t1).*factor;
pose(2,:) = sin(t1).*factor;
pose(3,:) = L1.*cos(t2) + L2.*cos(t3) + (L3+L4).*cos(t4);

```

```
31 % phi = atan2(R13/-R23)
32 R13 = cos(t1).*sin(t4);
33 R23 = sin(t1).*sin(t4);
34 pose(4,:) = 2.*atan(R13./(sqrt(R23.^2+R13.^2)-R23));
35
36 % theta = acos(R33)
37 pose(5,:) = acos(cos(t4));
38
39 % psi = atan2(R31/R32)
40 R31 = sin(t4).*sin(t5);
41 R32 = sin(t4).*cos(t5);
42 pose(6,:) = 2.* atan(R31./(sqrt(R32.^2+R31.^2)+R32));
43
44 %% Computation of the Jacobian matrix
45 for i=1:6
46   for j=1:5
47     J(i,j) = diff(pose(i),t(j));
48   end
49 end
```

B.2 Dynamic Time Warping Algorithm

Algorithm B.1 DTW Cumulative Distance Matrix

input: $\boldsymbol{\xi}^{(1)}$ (first sequence), $\boldsymbol{\xi}^{(2)}$ (second sequence)
$n = |\boldsymbol{\xi}^{(1)}|$
$m = |\boldsymbol{\xi}^{(2)}|$

// Local cost matrix
for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, m$ **do**
        $\boldsymbol{C}_{i,j} = \textsc{Distance}(\boldsymbol{\xi}_i^{(1)}, \boldsymbol{\xi}_j^{(2)})$
    end for
end for

// Accumulated cost matrix
$\boldsymbol{\Gamma}_{0,0} = 0$
for $i = 1, \ldots, n$ **do**
    $\boldsymbol{\Gamma}_{i,1} = \boldsymbol{\Gamma}_{i-1,1} + \boldsymbol{C}_{i,1}$
end for
for $j = 1, \ldots, m$ **do**
    $\boldsymbol{\Gamma}_{1,j} = \boldsymbol{\Gamma}_{1,j-1} + \boldsymbol{C}_{1,j}$
end for
for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, m$ **do**
        $\boldsymbol{\Gamma}_{i,j} = \boldsymbol{C}_{i,j} + \min\{\boldsymbol{\Gamma}_{i-1,j}, \boldsymbol{\Gamma}_{i,j-1}, \boldsymbol{\Gamma}_{i-1,j-1}\}$
    end for
end for
return: $\boldsymbol{\Gamma}$ (cumulative distance matrix)

Algorithm B.2 DTW Optimal Warping Path

input: $\boldsymbol{\Gamma}$ (Cumulative Distance Matrix)
$\boldsymbol{W} = \{\}$
$i = \text{ROWS}(\boldsymbol{\Gamma})$, $j = \text{COLUMNS}(\boldsymbol{\Gamma})$
while $i > 1$ and $j > 1$ **do**
    if $i == 1$ **then**
        $j = j - 1$
    else if $j == 1$ **then**
        $i = i - 1$
    else
        $q = \min\{\boldsymbol{\Gamma}_{i-1,j}, \boldsymbol{\Gamma}_{i,j-1}, \boldsymbol{\Gamma}_{i-1,j-1}\}$
        if $q == \boldsymbol{\Gamma}_{i-1,j-1}$ **then**
            $i = i - 1$
            $j = j - 1$
        else if $q == \boldsymbol{\Gamma}_{i-1,j}$ **then**
            $i = i - 1$
        else
            $j = j - 1$
        end if
    end if
    $\boldsymbol{W} = \{\boldsymbol{W}, (i, j)\}$
end while
return: $\boldsymbol{W}$ (optimal warping path)

Bibliography

Aris Alissandrakis, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. Imitation with ALICE: learning to imitate corresponding actions across dissimilar embodiments. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 32(4):482–496, 2002. (page 5)

Aris Alissandrakis, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. Correspondence mapping induced state and action metrics for robotic imitation. *Systems, Man, and Cybernetics, Part B, IEEE Transactions on*, 37(2):299–307, 2007. (page 6)

Brenna Argall, Sonia Chernova, Manuela M. Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5): 469–483, 2009. (page 4)

Christopher G. Atkeson and Stefan Schaal. Robot learning from demonstration. In *Proc. 14th International Conference on Machine Learning*, pages 12–20. Morgan Kaufmann, 1997. (page 3)

Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics*. Springer, 2008. In press. (page 4)

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. (pages 10, 14, 22, 25, and 36)

Sylvain Calinon. *Continuous Extraction of Task Constraints in a Robot Programming by Demonstration Framework*. PhD thesis, Learning Algorithms and Systems Laboratory (LASA), Ecole Polytechnique Federale de Lausanne (EPFL), 2007. (pages 6, 9, and 48)

Sylvain Calinon and Aude Billard. What is the teacher's role in robot programming by demonstration? - Toward benchmarks for improved learning. *Interaction Studies. Special Issue on Psychological Benchmarks in Human-Robot Interaction*, 8(3): 441–464, 2007a. (page 6)

Sylvain Calinon and Aude Billard. Active teaching in robot programming by demonstration. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 702–707, August 2007b. (page 6)

Sylvain Calinon and Aude Billard. A probabilistic programming by demonstration framework handling skill constraints in joint space and task space. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, September 2008. (pages 49 and 64)

Sylvain Calinon, Florent Guenter, and Aude Billard. On learning the statistical representation of a task and generalizing it to various contexts. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2978–2983, May 2006. (page 6)

Staffan Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 358–363, 2006. (page 6)

H. Friedrich, S. Münch, R. Dillman, S. Bocionek, and M. Sassin. Robot programming by demonstration (RPD): Supporting the induction by human interaction. *Machine Learning*, 23:163–189, 1996. (page 6)

M. N. Gibbs. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, 1997. (pages 10 and 30)

Paul W. Goldberg, Christopher K. I. Williams, and Christopher M. Bishop. Regression with input-dependent noise: A gaussian process treatment. In *Neural Information Processing Systems*. The MIT Press, 1997. (page 43)

Florent Guenter, Micha Hersch, Sylvain Calinon, and Aude Billard. Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics, Special Issue on Imitative Robots*, 2007. In press. (page 3)

M. Hersch, F. Guenter, S. Calinon, and A. Billard. Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Trans. on Robotics*, 24(6): 1463–1467, 2008. (page 68)

Kazuo Hirai, Masato Hirose, Yuji Haikawa, and Toru Takenaka. The development of honda humanoid robot. In *ICRA*, pages 1321–1326, 1998. (page 2)

G. E. Hovl, P. Sikka, and B. J. Mccarragher. Skill acquisition from human demonstration using a hidden markov model, October 22 1996. (page 6)

Yoon-Kwon Hwang, Kook Jin Choi, and Dae Sun Hong. Self-learning control of cooperative motion for a humanoid robot. In *ICRA*, pages 475–480. IEEE, 2006. (page 7)

Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *ICRA*, pages 1398–1403. IEEE, 2002a. ISBN 0-7803-7273-5. (page 7)

Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems 15*, pages 1547–1554. MIT Press, 2002b. (page 7)

F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, February 1975. (page 52)

Kristian Kersting, Christian Plagemann, Patrick Pfaff, and Wolfram Burgard. Most likely heteroscedastic gaussian process regression. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvalis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 393–400, 2007. ISBN 978-1-59593-793-3. (pages 43, 45, and 46)

Alain Liegois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-7(12): 868–871, 1977. (page 49)

David J. C. MacKay. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, 1991. (pages 33 and 34)

David J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992. (page 37)

David J. C. MacKay. Gaussian processes - a replacement for supervised neural networks?, 1997. (pages 10 and 11)

David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. (pages 36 and 37)

Thomas P. Minka. Expectation propagation for approximate bayesian inference. In Jack Breese and Daphne Koller, editors, *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 362–369, San Francisco, CA, August 2–5 2001. Morgan Kaufmann Publishers. (page 36)

Jorge J. More, David J. Thuente, and Preprint Mcs-p. Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Math. Software*, 20:286–307, 1992. (page 38)

S. Münch, J. Kreuziger, M. Kaiser, and R. Dillmann. Robot programming by demonstration (rpd) - using machine learning and user interaction methods for the development of easy and comfortable robot programming systems. In *In Proceedings of the 24th International Symposium on Industrial Robots*, pages 685–693, 1994. (pages 5 and 6)

R. M. Neal. *Bayesian learning for neural networks*. Springer-Verlag, New York, 1996. (pages 10, 13, and 25)

Radford M. Neal. Probabilistic inference using markov chain monte carlo methods. Technical report, University of Toronto, 1993. (page 37)

Radford M. Neal. Monte carlo implementation of gaussian process models for bayesian regression and classification, April 28 1997. (page 37)

Monica N. Nicolescu and Maja J. Mataric. Natural methods for robot task learning: instructive demonstrations, generalization and practice. In *AAMAS*, pages 241–248. ACM, 2003. ISBN 1-58113-683-8. (page 6)

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, 1999. ISBN 0-387-98793-2. (pages 37, 38, 39, and 40)

Michael Pardowitz, Steffen Knoop, Rüdiger Dillmann, and R. D. Zollner. Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(2):322–332, 2007. (page 6)

Jan Peters. *Machine Learning of Motor Skills for Robotics*. PhD thesis, University of Southern California, 2007. (page 3)

Jan Peters and Stefan Schaal. Policy learning for motor skills. In Masumi Ishikawa, Kenji Doya, Hiroyuki Miyamoto, and Takeshi Yamakawa, editors, *ICONIP (2)*, volume 4985 of *Lecture Notes in Computer Science*, pages 233–242. Springer, 2007. ISBN 978-3-540-69159-4. (page 3)

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomput.*, 71(7-9):1180–1190, 2008. ISSN 0925-2312. (page 68)

Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Conference on Humanoid Robots*, September 2003. (page 7)

Jan Peters, Stefan Schaal, and Bernhard Schölkopf. Towards machine learning of motor skills. In Karsten Berns and Tobias Luksch, editors, *AMS*, Informatik Aktuell, pages 138–144. Springer, 2007. ISBN 978-3-540-74763-5. (page 3)

C. Plagemann. *Gaussian Processes for Flexible Robot Learning*. PhD thesis, University of Freiburg, Department of Computer Science, December 2008. (page 43)

Carl Edward Rasmussen. *Evaluation Of Gaussian Processes And Other Methods For Non-Linear Regression*. PhD thesis, University of Toronto, 1996. (page 10)

Carl Edward Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. (pages 10, 11, 14, 15, 21, 22, 25, 26, and 34)

H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *ieeeassp*, 26(1):43–49, Feb., 1978. (pages 52 and 53)

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999. (page 2)

Stefan Schaal, Christopher G. Atkeson, and Sethu Vijayakumar. Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17 (1):49–60, July 2002. (page 6)

Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philos Trans R Soc Lond B Biol Sci*, 358(1431):537–547, March 2003. (page 4)

Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1259–1266, 2006a. (page 68)

Edward Snelson and Zoubin Ghahramani. Variable noise and dimensionality reduction for sparse gaussian processes. In *UAI*. AUAI Press, 2006b. ISBN 0-9749039-2-2. (pages 43 and 44)

Edward Lloyd Snelson. *Flexible and efficient Gaussian process models for machine learning*. PhD thesis, University of Cambridge, 2007. (page 43)

S. Sundararajan and S. S. Keerthi. Predictive approaches for choosing hyperparameters in gaussian processes. *Neural Computation*, 13(5):1103–1118, 2001. (page 36)

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998. ISBN 0262193981. (page 2)

J. Shawe Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004. (page 25)

Ales Ude. Trajectory generation from noisy positions of object features for teaching robot paths. *Robotics and Autonomous Systems*, 11(2):113–127, 1993. (page 6)

Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000*, pages 1079–1086, 2000. (page 6)

Christopher K. I. Williams. Prediction with gaussian processes: From linear regression to linear prediction and beyond. In *Learning and Inference in Graphical Models*, pages 599–621. Kluwer, 1997. (page 10)

Christopher K. I. Williams. Computation with infinite neural networks. *Neural Computation*, 10(5):1203–1216, 1998. (pages 10 and 36)

Christopher K. I. Williams and David Barber. Bayesian classification with gaussian processes. *IEEE Trans. Pattern Anal. Mach. Intell*, 20(12):1342–1351, 1998. (page 11)

Max A. Woodbury. Inverting modified matrices. Memo. Report 42, Statistical Research Group, Princeton Univ., 1950. (page 69)