Dissertation zur Erlangung des Doktorgrades
der Technischen Fakultät
der Albert-Ludwigs-Universität Freiburg im Breisgau

Model Learning in Robot Control

Duy Nguyen-Tuong

May, 2011

Albert-Ludwigs-Universität Freiburg im Breisgau
Technische Fakultät
Institut für Informatik

Dekan der Technischen Fakultät
Prof. Dr. Bernd Becker

1. Gutachter
Prof. Dr. Martin Riedmiller, Universität Freiburg
2. Gutachter
Prof. Dr. Bernhard Schölkopf, MPI für Biologische Kybernetik

Tag der Disputation
26.05.2011

To my family and Moni

Contents

Abstract

Creating autonomous robots that can safely interact with humans in daily life has been a long-standing dream of robotics and artificial intelligence. This goal cannot be achieved without compliant and fully adaptive robot control, which requires accurate models of the robot and its environment. However, analytical models obtained from physics-based modeling techniques have shown to be insufficient for many modern robots systems. It appears that compliant and fully adaptive robot control can only be achieved with model learning. In this thesis, we explore how statistical kernel-based learning techniques can be employed in real-time approximate model based robot control. The presented work includes contributions to robot control, as well as machine learning. This thesis advances a kernel-based statistical approach to robot control.

We show that kernel-based learning can be used to obtain both compliance and accurate control performance. In this case, learning the inverse dynamics models is necessary to predict the required torques for the robot to perform a desired task. In order to adapt the models to changes in the robot dynamics and environment, real-time online learning of such models is essential. We additionally present machine learning solutions to model learning problems in robotics, such as learning models from multi-valued mappings. Learning from such mappings is necessary for approximating torque prediction models for task-space robot tracking control. The proposed kernel-based approach for learning models from multi-valued mappings is based on the insight that although it is globally ill-posed, the learning problem is locally well-defined. Additionally, in many real-world situations, we also face the problems of sparse and potentially poor data. In such cases, nonparametric learning methods can fail to provide a good prediction model. We investigate how prior model knowledge can help to improve the model learning process in the presence of sparse and poor data. The developed semiparametric learning approaches are effective for this data.

As a contribution to machine learning, we develop kernel-based learning methods which enable online model learning in real-time. We present two novel real-time kernel learning techniques for online model approximation. The first approach employs the local learning principle to speed up the nonparametric Gaussian process regression. The core idea behind this approach is to partition the data space into local regions, for which independent local Gaussian models are learned. Using these local models, learning and prediction can be accelerated significantly. The second approach relies on the concept of sparsification. Here, the idea is to select informative points from the stream of online arriving data and use them for learning the models. We present a framework for online, incremental sparsification designed for fast real-time model learning. The proposed framework can be used to speed up incremental learning methods appropriate for online model learning in real-time. The approaches are implemented and evaluated on a Barrett whole arm manipulator for real-time learning control.

Zusammenfassung

Die Erschaffung autonomer Roboter, die mit Menschen im täglichen Leben interagieren, ist ein erklärtes Ziel der Robotik und der künstlichen Intelligenz. Dieses Ziel kann jedoch nicht ohne adaptive und nachgiebige Roboterregelung realisiert werden. Eine weiche und nachgiebige Regelung erfordert jedoch ein genaues Modell des Roboters und seiner Umgebung. Es zeigt sich, dass traditionelle analytische Modelle für viele moderne Roboter-Systeme unzureichend sind. In solchen Fällen kann eine nachgiebige und adaptive Roboterregelung nur mit Hilfe von Modelllernen erreicht werden. In dieser Arbeit soll gezeigt werden, wie statistische Lerntechniken in der modellbasierten Roboterregelung eingesetzt werden können.

Es soll dargestellt werden, wie kernbasiertes Lernen angewendet werden kann, um eine genaue Regelung zu erlangen und gleichzeitig den Roboter nachgiebig zu halten. Um die erforderlichen Drehmomente für die Roboterbewegung zu berechnen, ist es notwendig, die inverse Dynamik des Roboters zu lernen. Darüber hinaus ist Online-Lernen in Echtzeit für solche Modelle unerlässlich, um den Veränderungen in der Dynamik und Roboter-Umgebung Rechnung zu tragen. Zusätzlich wird dargestellt, wie maschinelles Lernen dazu beiträgt, das Lernen von nicht eindeutigen Abbildungen zu ermöglichen. Das Lernen dieser Abbildungen ist notwendig für eine Folgeregelung im Roboter-Handraum. Der hierzu vorgeschlagene kernbasierte Ansatz beruht auf der Erkenntnis, dass - obwohl solche Abbildungen global nicht wohl-definiert sind - das Lernproblem jedoch lokal wohl-definiert ist. Darüber hinaus stellt sich zudem in vielen realen Situationen das Problem der nicht-informativen Daten. In solchen Fällen können nicht-parametrische Lernmethoden keine guten Modelle für die Regelung bieten. Die vorliegende Untersuchung zeigt, wie zusätzliche analytische Modellierungsinformationen nicht-parametrischem Modelllernen helfen kann. Es wird gezeigt, dass semi-parametrische Lernansätze den Lernprozess bei spärlicher und nicht-informativer Datengrundlage entscheidend verbessern können.

Als Beitrag zum maschinellen Lernen werden kernbasierte Lernmethoden entwickelt, die Online-Modelllernen in Echtzeit ermöglichen. Hierzu werden zwei neue kernbasierte Lerntechniken für das Online-Lernen in Echtzeit vorgestellt. Der erste Ansatz nutzt das Prinzip des lokalen Lernens zur Beschleunigung der nicht-parametrischen Gauß'schen Prozesse. Die zentrale Idee dieser Methode ist die Einteilung des Datenraums in lokale Regionen, für die unabhängige lokale Gauß'sche Modelle gelernt werden. Mit Hilfe dieser lokalen Modelle kann die Geschwindigkeit des Lernens deutlich erhöht werden. Der zweite Ansatz beruht auf dem Konzept der Sparsifikation. Das entscheidende Vorgehen bei dieser Methode besteht darin, informative Punkte aus der Menge der ankommenden Daten online zu wählen und diese für das Lernen der Modelle zu verwenden. Hierzu wird eine Methode für die inkrementelle Online-Sparsifikation vorgestellt. Die vorgeschlagene Methode kann dazu verwendet werden, inkrementelle Online-Lernverfahren echtzeitfähig zu machen. Die vorgestellten Ansätze wurden auf einem Barrett-Manipulator für die Echtzeit-Lernregelung implementiert und evaluiert.

Acknowledgment

I would like to thank all the wonderful people who have contributed to this work and who have made my PhD such a pleasant experience.

My thanks go first to my supervisor Jan Peters who guided me through my research, and whose comments often sparked my deeper interest in directions which I would have ignored otherwise. I have also benefitted from his wide overview of relevant literature in many areas of robotics I have been interested in.

I would like to thank Bernhard Schölkopf for many inspiring discussions and interesting talks covering a wide range of topics in machine learning and kernel methods. I would also like to thank Martin Riedmiller for reviewing this thesis and for being my Doktorvater. I am very grateful to Matthias Seeger for sharing some of his knowledge about Gaussian Processes and inference.

It was a pleasure for me to work with all the wonderful people in our RoboLab here in Tübingen. In particular, I want to thank my colleagues and friends: Jens Kober, Katharina Muelling, Oli Kroemer, Botond Bocsi, Abdeslam Boularias, Zhikun Wang, Yevgeny Seldin, Justus Piater. I will remember the lab-parties and joyful Carcassonne-evenings we had. My postgraduate studies led me into terrain which was quite unfamiliar to me, and it would have been a much harder journey without the many helps I got from the people in the department Empirical Inference at the MPI for Biological Cybernetics. I would like to thank all the people from the department for the friendly atmosphere and for stimulating discussions during the coffee breaks.

I am also grateful to Stefan Schaal for making my stay at the University of Southern California possible. My thanks also go to the people I met there at the computational learning and motor control lab for many interesting discussions. In particular, I want to thank Mrinal Kalakrishnan for accommodating me during my USC stay.

I am thankful to Oli and Lawrence for helping me polish the language and style of this thesis. I would also like to thank the "soccer people" for many great matches which help me to recover my balance.

Finally and most importantly, I want to thank Monika and my family for the support and love they gave me in every period of my life.

1 Introduction

To date, several million complex robots with many degrees of freedom populate factories world-wide. These robots are engineered with powerful motors using heavy, rigid gear boxes. This design approach is necessary in order to use traditional control methods. However, this approach results in an unyielding stiffness, therefore, these robots can only be operated efficiently in a fully modeled environment. Keeping unmodeled disturbances out of the environments is essential for these setups. In particular, human beings also need to stay out of the reach of these robots both for their own safety, as well as to keep the environment in a fully modeled state.

In order to integrate high-dimensional robots (e.g., humanoids and service robots) into daily life, we need compliant, fully adaptive robots that are guaranteed to be safe when interacting with humans. A key component towards this goal is the precise but compliant robot control, which requires accurate models. It seems that this aim can only be achieved using model learning, as we will detail later. In this thesis, we investigate how robot control can benefit from model learning using statistical learning approaches. The principle objective of this thesis is to move closer to fully autonomous robots, which can interact safely with humans in their uncertain environments.

1.1 Motivation

Accurate models of the robot system allow the design of significantly more precise, energy-efficient and compliant controls for robots. Model based control uses a system model to predict the required torques for executing a kinematically specified task (Craig, 2004; Slotine and Li, 1991). It offers a large variety of advantages over model-free methods, e.g., potentially higher tracking accuracy, lower feedback gains and higher suitability for compliant control. Despite the potential advantages, model based control is currently used in only few robot setups. The reason for this lack of application is the requirement of precise models of the robot system. Traditionally, such models can be obtained analytically using physics-based modeling techniques. However, in many cases the accuracy of such analytical models does not suffice for proper control performance due to unmodeled nonlinearities, such as hydraulic cables, complex friction, or actuator dynamics. In such cases, the automatic acquisition of models from data using estimation techniques poses an interesting alternative, and approximate model based control is a promising approach (Farrell and Polycarpou, 2006). Using modern regression methods, we can achieve a better model than with the physics-based formulation. It further allows us to take the

time-dependency into account by adapting the model over time, i.e., online model learning.

Approximate model based control using learning techniques has attracted much attention since the late 1970s. Starting with pioneering work in adaptive self-tuning control (Aström and Wittenmark, 1995), model based learning control has been developed in many aspects ranging from neural network controllers (Patino *et al.*, 2002) to more recent control paradigms using nonparametric methods (Nakanishi and Schaal, 2004). However, approximate model based control poses a tremendous technical challenge for modern statistical kernel-based learning approaches, such as Gaussian process regression (Rasmussen and Williams, 2006) and support vector regression (Schölkopf and Smola, 2002). While these nonparametric regression approaches provide a powerful tool to accurately approximate models from data, the high computational complexity of such approaches prevents a straightforward application to real-time control. Especially in robot tracking control, real-time online model learning is additionally required to cope with unknown state space regions, and to adapt the model to changes in the robot dynamics. Real-time online model learning poses three major challenges for kernel-based model learning: first, the learning and prediction processes need to be sufficiently fast. Second, the learning system needs to deal with large amounts of data. Third, since the data arrives as a continuous stream, the model needs to be able to continuously adapt to new examples. Thus, it is essential to develop efficient kernel-based learning approaches appropriate for online model learning in real-time. This step is absolutely necessary to establish statistical kernel learning approaches in the field of robot control.

1.2 Contributions

Online model learning in real-time has not been thoroughly addressed by statistical kernel-based learning approaches, such as kernel methods or Bayesian learning techniques. The goal of this thesis is to investigate how kernel learning can be used to learn models for robot control, and how kernel-based regression approaches can be sped up to cope with the real-time requirements in robotics. Thus, the contribution of this thesis is twofold:

1. We contribute to the field of statistical machine learning by developing fast online kernel-based regression approaches that are appropriate for real-time model learning.
2. We contribute to the field of robot control by showing that statistical kernel-based learning can be used to achieve both compliant and accurate control performance. We further investigate how kernel learning can help to solve real-world model learning problems, such as learning from multi-valued mappings and learning from sparse data.

In the next sections, we outline the major contributions of this thesis. The contributions result in novel kernel-based algorithms for real-time online model learning, and applications of kernel learning in model based robot control.

1.2.1 Novel Kernel-based Learning Algorithms

Model learning using kernel-based approaches can be expensive, as the computational complexity scales with the number of training data points. In the field of machine learning, several attempts have been made to reduce the computational cost of kernel-based model learning (for an overview see Schölkopf and Smola (2002)). These approaches are mainly based on two different, but effective approximation strategies. The first approach relies on the localization and partitioning of data (Vijayakumar and Schaal, 2000; Treps, 2001). The basic idea is to divide the original data space into many local regions, where each of the local region contains a limited number of data points. Learning and prediction can be sped up by using these local models trained on the associated data points. The main problem of this approach is determining how to partition the data space properly, and how to train the local models for an accurate prediction afterwards. The second possibility to reduce the computational complexity is to employ data sparsification (Candela and Rasmussen, 2005). Here, the original data is approximated by a smaller set of so-called inducing data points. In this approach, the difficulty lies in the selection of informative data points that approximately summarize the full data. However, current approximation methods are not applicable for online model learning in real-time. They are too expensive for real-time learning which must take place at a speed of roughly 100 Hz. Furthermore, most of them do not consider the online learning scenario, where the data arrives incrementally over time. Inspired by the concepts behind these two strategies for complexity reduction, we propose novel learning algorithms appropriate for real-time online model learning. In particular, we propose the local Gaussian process regression and an online sparsification technique to speed up the incremental model learning.

Local Gaussian Process Regression. Gaussian process regression (GPR) is well-known to be both accurate and straightforward to use (Rasmussen and Williams, 2006). However, the major drawback of GPR is the extensive computational complexity, which scales cubically in the number of training data points. To reduce the computational cost, local Gaussian process regression (Nguyen-Tuong and Peters, 2009) combines the local learning principle with nonparametric Gaussian process regression. Local GPR employs an online partitioning step for splitting the data space into local regions. In particular, each data point is assigned to the nearest local models based on a kernel distance measure, as it arrives. A Gaussian model is incrementally trained for each local region. Prediction can be made for a query point by using models in the vicinity. The approach has competitive learning performance for high-dimensional data while being sufficiently fast for real-time learning. This approach is among the fastest Bayesian regression techniques to date, and has been used for learning robot's inverse dynamics in real-time for model based control (Nguyen-Tuong *et al.*, 2008b).

Incremental Online Sparsification. In the machine learning literature, several online incremental learning algorithms have been proposed over the last decade, such as incremental support vector machines (Cauwenberghs and Poggio, 2000) and

incremental Gaussian processes (Candela and Winther, 2003). However, these incremental learning approaches are too slow for real-time online learning. To speed up these incremental learning approaches, we incrementally train the model using only informative inducing data points while ignoring the others. Selecting informative points from a data stream and occasionally removing old inducing data points is related to the problem of sparsification. We present a framework for online, incremental sparsification (Nguyen-Tuong and Peters, 2010b) while employing a kernel independence measure for the sparsification process. This approach can be used to speed up existing incremental learning algorithms for real-time applications. To demonstrate the applicability of the method, we combine the approach with incremental Gaussian process regression and incremental support vector regression, obtaining model approximation methods that are applicable to real-time online learning (Nguyen-Tuong and Peters, 2010b; Nguyen-Tuong *et al.*, 2009).

1.2.2 Robot Control Applications

Control applications require accurate models of robots, such as inverse dynamics models for joint space tracking control and torque prediction models for task-space control. Learning these models from data ameliorates the lack of accuracy due to unknown nonlinearities. Additionally, prior knowledge can be used to improve the quality of learned models, which is especially important when data is noisy or sparse.

Learning Inverse Dynamics Models. Inverse dynamics models are mappings from joint space to torque space. In joint space tracking control, inverse dynamics models are employed to predict the required torques for the robot to follow a given joint space trajectory (Craig, 2004). If accurate inverse dynamics models are available, low tracking gains can be used. Thus, the robot will achieve both precise tracking performance, as well as compliant control (Nguyen-Tuong *et al.*, 2008a). When learning inverse dynamics models online, the robot can further adapt to changes in the dynamics. It additionally ensures a good generalization of the learned models, when the robot moves to unknown state space regions. We show that learning inverse dynamics models is a key component for compliant and accurate robot control. Furthermore, we show that kernel-based learning approaches are promising for learning such models for real-time control (Nguyen-Tuong and Peters, 2010b, 2009).

Learning Models for Task-Space Tracking Control. Task-space tracking control is a general framework for directing a robot to follow certain trajectories in task-space. Formulation of task-space control laws requires model knowledge about the robot's kinematics and dynamics. However, task-space control approaches are known to be susceptible to modeling errors (Nakanishi *et al.*, 2008). Learning models for task-space tracking control may help to overcome this problem. A task-space control model predicts the joint torques that yield a correct task-space trajectory. However, direct learning of such models is an ill-posed problem. Recent work (Peters and Schaal, 2008; D'Souza *et al.*, 2001) shows that though the global problem is ill-posed, the learning is *locally* well-defined. Inspired by this insight, we employ a

kernel-based local learning method to learn a model for task-space tracking control.

Using Prior Knowledge for Model Learning. In many real-world situations, it is not possible to have large and informative data sets for model learning. In such cases, nonparametric models will not be able to generalize well to unknown data. Therefore, it is essential to investigate how the learning performance can be improved in such cases. In particular, we combine nonparametric model learning with analytical models, obtaining a semiparametric regression framework (Nguyen-Tuong and Peters, 2010c). We identify two approaches to incorporate the parametric model from analytical robotics into the nonparametric Gaussian process model. In the first approach, we insert the parametric model directly into the Gaussian process model as the mean function. In the second approach, we embed the parametric model in a kernel used for learning. We show that semiparametric models provide a higher model accuracy and better generalization in the presence of poor and sparse data (Nguyen-Tuong and Peters, 2010c).

1.3 Outline of the Thesis

This thesis is organized in six chapters. Each chapter is self-contained and, thus, the chapters can be read independently of each other. Figure 1.1 illustrates the outline and relation of the thesis chapters.

In Chapter 2, we survey the research on model learning within the field of robotics, with a strong focus on recent developments. In particular, we consider three aspects of model learning. First, we study the different possible model learning control architectures for robotics. Second, we discuss what kind of problems these control architectures imply for the applicable learning methods. Furthermore, we show in several case studies where these scenarios have been successfully applied. In addition to giving an overview of the field of model learning for robot control, this chapter further motivates the need of model learning for modern robot systems, and discusses how model learning can be obtained using statistical learning approaches.

Learning models from data in real-time poses a technical challenge for statistical regression methods, such as Gaussian process regression (GPR). In Chapter 3, we present a local approximation to the well-established GPR in order to accelerate the method sufficient for real-time online learning. Here, we combine the high accuracy of GPR with the fast speed of local learning methods, such as locally weighted projection regression (LWPR). First, we briefly describe the nonparametric learning approaches, i.e., standard GPR and LWPR. We then describe our local Gaussian process models (LGP) approach. We show that LGP inherits both the precision of GPR and the speed similar to LWPR. The learning accuracy and performance of the presented LGP approach is compared with several relevant regression methods. The applicability of the LGP for low-gain model based tracking control and real-time learning is demonstrated on a Barrett whole arm manipulator.

In Chapter 4, we present a framework for online, incremental sparsification designed for fast real-time model learning. In contrast to the LGP algorithm in Section

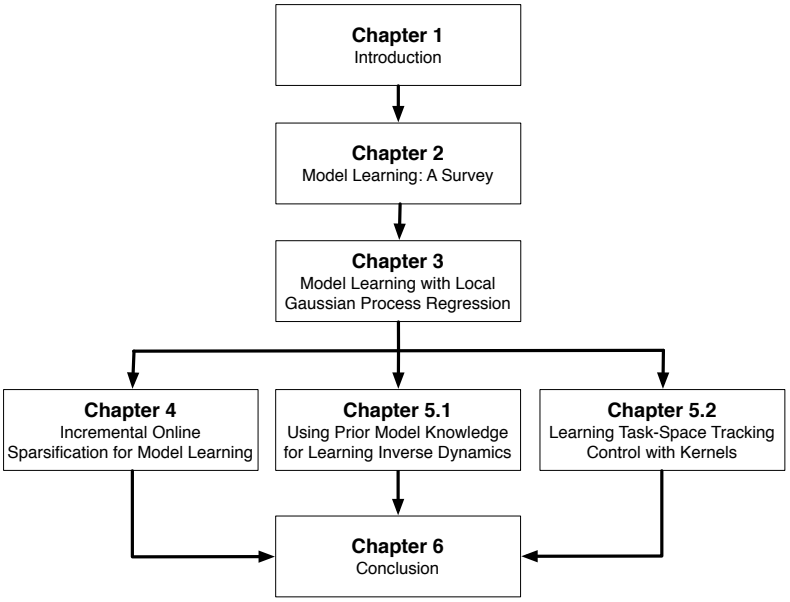


Figure 1.1: The structure of this thesis. Chapter 2 provides a review of the progress in model learning. Chapters 3, 4 and 5 show how kernel-based learning can be used to learn models for real-time robot control. Chapter 1 gives an introduction, while Chapter 6 concludes the thesis and provides an outlook to future work.

3, this sparsification framework can be used to speed up incremental learning approaches for real-time online model learning. The proposed sparsification approach selects and removes data points for a sparse set. In combination with an incremental learning approach, such as incremental GPR, we obtain a model approximation method which is applicable in real-time online learning. The efficiency of the proposed approach is demonstrated with an offline comparison of our method with well-established regression methods. The approach is subsequently used for an online approximation of inverse dynamics models for real-time tracking control on a Barrett whole arm manipulator.

In Chapter 5, we explore solutions to two model learning problems: learning models from small and potentially poor data set; and learning from multi-valued mappings for task-space tracking control. When only sparse and poor data is available, the learning performance can be improved by incorporating additional prior knowledge into the model learning process. In Section 5.1, we present two possible semiparametric regression approaches, where the knowledge of the physical model can either become part of the mean function or of the kernel in a nonparametric Gaussian process regression. In Section 5.2, we investigate the problem of learning models from multi-valued mappings for task-space tracking control. In particular, we employ a local learning approach formulated in the kernel framework. This approach is motivated by the insight that the model learning problem is locally well-defined in such cases, while it is globally ill-posed. In Chapter 6, we give a conclusion and discuss possible future work.

2 Model Learning: A Survey

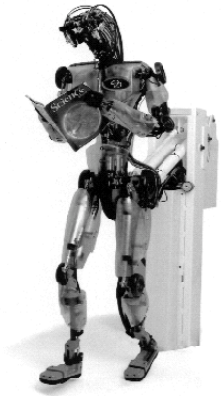
Models are among the most essential tools in robotics, and it is widely believed that intelligent mammals also rely on internal models in order to generate their actions. A model describes essential information about the behavior of the environment and the influence of an agent on this environment. While classical robotics relies on manually generated models, future autonomous, cognitive robots need to be able to automatically learn these models from data.

In this chapter, we survey progress in model learning with a strong focus on recent developments. First, we need to study different types of models and how these models can be incorporated in various learning control architectures. Second, we discuss what kind of problems these control architectures create for the applicable learning methods. Finally, we show where these scenarios have been used successfully in several case studies.

2.1 Model Learning for Robotics

Machine learning may allow avoiding that all possible scenarios need to be programmed, but rather learned by the system during operation. There have been many attempts at creating learning frameworks, enabling robots to autonomously learn complex skills ranging from task imitation to motor control (Schaal, 1999; Wolpert and Kawato, 1998; Schaal and Atkeson, 2010). However, learning is not an easy task. For example, reinforcement learning can require more trials and data than one can generate in the life-time of a robot, and black box imitation learning can at best reproduce the desired behavior. Thus, it is essential to study how the basic, underlying mechanisms of the world can be learned. This approach is commonly known as *model learning*.

In recent years, methods to learn models from data have become interesting tools for robotics, as they allow straightforward and accurate model approximation. The reason for this increasing interest is that accurate analytical models are often hard to obtain due to the complexity of modern robot systems and their presence in unstructured, uncertain and human-inhabited environments (Nguyen-Tuong and Peters, 2009; Nakanishi *et al.*, 2008). Model learning can be a useful alternative to manual pre-programming, as the model is estimated directly from measured data. Unknown nonlinearites can be directly taken in account, while they are neglected by the standard physics-based modeling techniques and by hand-crafted models. In order to generalize the learned models to a larger state space and to adapt the models for time dependent changes, online learning of such models is necessary. In general, a model contains essential information about the system and describes the influence



(a) Humanoid Robot DB

(b) Mobile LAGR Robot

(c) Boston Dynamics Little Dog

Figure 2.1: Plattforms with well-known applications of model learning: (a) Schaal et al. learned the complete inverse dynamics model for Humanoid DB (Schaal *et al.*, 2002); (b) Angelova et al. predicted the slip of the mobile LAGR robot based on learned models that required visual features as input (Angelova *et al.*, 2006); (c) Kalakrishnan et al. estimated foothold quality models based on terrain features for the Boston Dynamics little dog (Kalakrishnan *et al.*, 2009).

of an agent on this system. Thus, modeling a system is inherently connected with the question how the model can be used to manipulate, i.e., to control, the system. Model learning has been shown to be an efficient tool in a variety of scenario, such as inverse dynamics control (Nguyen-Tuong and Peters, 2009), inverse kinematics (Reinhart and Steil, 2009; Ting *et al.*, 2008), robot manipulation (Steffen *et al.*, 2009; Klanke *et al.*, 2006), autonomous navigation (Angelova *et al.*, 2006) or robot locomotion (Kalakrishnan *et al.*, 2009).

2.1.1 Problem Statement

Accurate models of the system and its environment are crucial for planning, control and many other applications. In this chapter, we focus on generating learned models of dynamical systems that are in a state $\boldsymbol{s}_k$ taking an action $\boldsymbol{a}_k$ and transfer to a next state $\boldsymbol{s}_{k+1}$, where we can only observe an output $\boldsymbol{y}_k$ that is a function of the current state and action. Thus, we have

$$\begin{aligned} \boldsymbol{s}_{k+1} &= f(\boldsymbol{s}_k, \boldsymbol{a}_k) + \boldsymbol{\epsilon}_f \,, \\ \boldsymbol{y}_k &= h(\boldsymbol{s}_k, \boldsymbol{a}_k) + \boldsymbol{\epsilon}_y \,, \end{aligned} \tag{2.1}$$

where f and h represent the state transition and the output function, $\boldsymbol{\epsilon}_f$ and $\boldsymbol{\epsilon}_y$ denote the noise components. In practice, state estimation techniques are often needed to reduce the noise of the state estimate and to obtain complete state information (Ko and Fox, 2009). While the output function h can often be described straightforwardly by an algebraic equation, it is more difficult to model the state transition

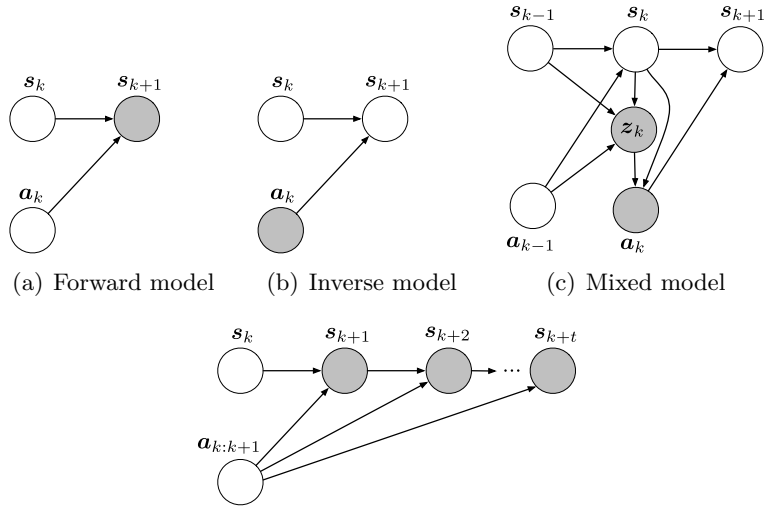
function f, as it includes more complex relationship between states and actions.

The state transition model f predicts the next state $\boldsymbol{s}_{k+1}$ given the current state $\boldsymbol{s}_k$ and action $\boldsymbol{a}_k$. Application of such state transition models in robotics and control has a long history. With the increasing speed of computation and its decreased cost, models have become common in robot control, e.g., in feedforward control and state feedback linearization. At the same time, due to the increasing complexity of robot systems, analytical models are more difficult to obtain. This problem leads to a variety of model estimation techniques which allow the roboticist to acquire models from data. Combining model learning with control has drawn much attention in the control community (Farrell and Polycarpou, 2006). Starting with the pioneering work in adaptive self-tuning control (Aström and Wittenmark, 1995), model based learning control has been developed in many aspects ranging from neural network controllers (Patino *et al.*, 2002) to more modern control paradigms using statistical methods (Kocijan *et al.*, 2004; Nakanishi and Schaal, 2004).

In early days of adaptive control, models are learned by fitting open parameters of pre-defined parametric models. Estimating such parametric models from data has been popular for a long time (Atkeson *et al.*, 1986; Khalil and Dombre, 2002) due to the applicability of well-known system identification techniques and adaptive control approaches (Ljung, 2004). However, estimating the open parameters is not always straightforward, as several problems can occur, such as persistent excitation issues (Narendra and Annaswamy, 1987). Furthermore, the estimated parameters are frequently not physically consistent (e.g., violating the parallel axis theorem or having physically impossible values) and, hence, physical consistency constraints have to be imposed on the regression problem (Ting *et al.*, 2009). Nonparametric model learning methods can avoid many of these problems. Modern nonparametric model learning approaches do not pre-define a fixed model structure but adapt the model structure to the data complexity. There have been efforts to develop nonparametric machine learning techniques for model learning in robotics and, especially, for robot control (Nakanishi *et al.*, 2005; Farrell and Polycarpou, 2006).

2.1.2 Overview

The aim of this chapter is to give a comprehensive overview of past and current research activities in model learning with a particular focus on robot control. The remainder of this chapter is organized as follows. First, we discuss different types of models in Section 2.2.1 and investigate how they can be incorporated into different learning control architectures in Section 2.2.2. In Section 2.2.3, we further discuss the challenges that arise from the application of learning methods in the domain of robotics. In Section 2.2.4, we provide an overview on how models can be learned using machine learning techniques with a focus on statistical regression methods. In Section 2.3, we highlight examples where model learning has proven to be helpful for the action generation in complex robot systems. The chapter will be summarized in Section 2.4.



(a) Forward model (b) Inverse model (c) Mixed model

(d) Example of an operator model

Figure 2.2: Graphical models for different types of models. The white nodes denote the *observed* quantities, while the grey nodes represent the quantities to be *inferred*. (a) The forward model allows inferring the next state given current state and action. (b) The inverse model determines the action required to move the system from the current state to the next state. (c) The mixed model approach combines forward and inverse models in problems where a unique inverse does not exist. Here, the forward and inverse models are linked by a latent variable $\boldsymbol{z}_t$. (d) The operator model is needed when dealing with finite sequences of future states.

2.2 Model Learning

Any rational agent will decide how to manipulate the environment based on its observations and predictions on its influence on the system. Hence, the agent has to consider two major issues. First, it needs to deduce the behavior of the system from some observed quantities. Second, having inferred this information, it needs to determine how to manipulate the system.

The first question is a pure modeling problem. Given some observed quantities, we need to predict the missing information to complete our knowledge about the action and system's reaction. Depending on what kind of quantities are observed (i.e., what kind of missing information we need to infer), we distinguish between *forward models*, *inverse models*, *mixed models* and *operator models*. Section 2.2.1 describes these models in more detail. The second question is related to the learning control architectures which can be employed in combination with these models. In this case, we are interested in architectures that incorporate learning mechanisms into control frameworks. Section 2.2.2 presents three different model learning architectures for control, i.e., *direct modeling*, *indirect modeling* and *distal teacher learning*. In practice, model learning techniques can not be used straightforwardly for many real-world applications, especially, for robot control. Section 2.2.3 gives an

overview of challenges that appear when model learning is used in robotics. Section 2.2.4 approaches the model learning problem from the algorithmic viewpoint, showing how models can be learned using modern statistical regression methods. Here, we will distinguish between *local* and *global* learning approaches.

2.2.1 Prediction Problems and Model Types

To understand the system's behavior and how it reacts due to the agent's actions, we need information about the states and actions (of the past, the presence and sometimes the expected future). However, we have only access to a limited number of these quantities in practice. Thus, we need to predict the missing information given the known information.

If we can observe the current state $\boldsymbol{s}_k$ and the current action $\boldsymbol{a}_k$ is given, we can attempt to predict the next state $\boldsymbol{s}_{k+1}$. Here, the *forward model* can be used to predict the next state given current state and action. The forward model describes the mapping $(\boldsymbol{s}_k, \boldsymbol{a}_k) \rightarrow \boldsymbol{s}_{k+1}$. We can use the *inverse model* to infer the current action, i.e., the relation $(\boldsymbol{s}_k, \boldsymbol{s}_{k+1}) \rightarrow \boldsymbol{a}_k$, if we know the current state and the desired or expected future state. There are also approaches combining forward and inverse models for prediction, which we will refer to as *mixed model* approaches. However, for many applications the system behavior has to be predicted for the next t-steps rather than for the next single step. Here, we need models to predict a series of states; we call such models *operator models*. Figure 2.2 illustrates these introduced models. In this section, we will additionally describe how the different models can be used in control.

2.2.1.1 Forward Models

Forward models predict the next state of a dynamic system given the current action and current state. Note that the forward model directly corresponds to the state transfer function f shown in Equation (2.1). As this function expresses the physical properties of the system, the forward model represents a causal relationship between states and actions. Thus, if such causal mappings have to be learned, it will result in a well-defined problem and learning can be done straightforwardly using standard regression techniques. While forward models of classical physics are unique mappings, there are several cases where forward models alone do not provide sufficient information to uniquely determine the next system's state (Hawes *et al.*, 2010). For instance, when a pendulum is located at an unstable equilibrium point, it is more likely to go to the left or right than to stay at the center. Nevertheless, the center point would be the prediction of a forward model. Here, the modes of a conditional density may be more interesting than the mean function f (Hawes *et al.*, 2010; Skočaj *et al.*, 2010).

An early application of forward models in classical control is the Smith predictor, where the forward model is employed to cancel out delays imposed by the feedback loop (Smith, 1959). Later, forward models have been applied, for example, in the

context of model reference adaptive control (MRAC) (Narendra and Annaswamy, 1989). MRAC is a control system in which the performance of an action is predicted using a forward model (i.e., a reference model). The controller adjusts the action based on the resulting error between the desired and current state. Hence, the policy $\boldsymbol{\pi}$ for the MRAC can be written as

$$\boldsymbol{\pi}(\boldsymbol{s}) = \text{argmin } \left(f_{\text{forward}}(\boldsymbol{s}_{\text{des}}, \boldsymbol{a}_{\text{des}}) - \boldsymbol{s}\right) , \tag{2.2}$$

where $(\boldsymbol{s}_{\text{des}}, \boldsymbol{a}_{\text{des}})$ denotes the desired trajectory and $\boldsymbol{s}$ represents the observed state. MRAC was originally developed for continuous-time system and has been extended later for discrete and stochastic systems (Narendra and Annaswamy, 1989). Applications of MRAC can be found numerously in robot control literature, such as adaptive manipulator control (Nicosia and Tomei, 1984). Further application of forward models can be found in the wide class of model predictive control (MPC) (Maciejowski, 2002). MPC computes optimal actions by minimizing a given cost function over a certain prediction horizon N in the future. The MPC control policy can be described by

$$\boldsymbol{\pi}(\boldsymbol{s}) = \text{argmin} \sum_{k=t}^{t+N} F_{\text{cost}} \left(f_{\text{forward}}(\boldsymbol{s}_k, \boldsymbol{a}_k) - \boldsymbol{s}_{\text{des}}\right) , \tag{2.3}$$

where F_{cost} denotes the cost function to be minimized. MPC is widely used in the industry, as it can deal with constraints in a straightforward way. MPC was first developed for linear system models and, subsequently, extended to more complex nonlinear models (Maciejowski, 2002). Forward models have also been essential in model based reinforcement learning approaches, which relate to the problem of optimal control (Sutton, 1991; Atkeson and Morimoto, 2002; Ng *et al.*, 2004). Here, the forward models describe the so-called transition dynamics determining the probability of reaching the next state given current state and action. In contrast to previous applications, the forward models incorporate a probabilistic description of the system dynamics in this case (Rasmussen and Kuss, 2003; Rottmann and Burgard, 2009). More details about the applications of forward models for optimal control will be given in the case studies in Section 2.3.1.

2.2.1.2 Inverse Models

Inverse models predict the action required to move the systems from the current state to a desired future state. In contrast to forward models, inverse models represent an anti-causal relationship. Thus, inverse models do not always exist or at least are not always well-defined. However, for several cases, such as for the robot's inverse dynamics, the inverse relationship is well-defined. General, potentially ill-posed inverse modeling problems can be solved by introducing additional constraints, as will be discussed in Section 2.2.1.3 in more detail.

For control, applications of inverse models can be traditionally found in computed torque robot control (Craig, 2004), where the inverse dynamics model is used to

predict the torques required to move the robot along a desired joint space trajectory. The computed torque control policy can be described by

$$\boldsymbol{\pi}(\boldsymbol{s}) = f_{\text{inverse}}(\boldsymbol{s}, \boldsymbol{s}_{\text{des}}) + k(\boldsymbol{s} - \boldsymbol{s}_{\text{des}}) \,, \tag{2.4}$$

where $k(\boldsymbol{s} - \boldsymbol{s}_{\text{des}})$ is an error correction term (for example, a PD-controller as both positions, velocities and accelerations may be part of the state) needed for stabilization of the robot. If an accurate inverse dynamics model is given, the predicted torques are sufficient to obtain a precise tracking performance. The inverse dynamics control approach is closely related to the computed torque control method. Here, the error correction term acts through the inverse model of the system (Craig, 2004) and, hence, we have a control policy given by

$$\boldsymbol{\pi}(\boldsymbol{s}) = f_{\text{inverse}}(\boldsymbol{s}, \boldsymbol{s}_{\text{des}}, k(\boldsymbol{s} - \boldsymbol{s}_{\text{des}})) \,. \tag{2.5}$$

If the inverse model perfectly describes the inverse dynamics, inverse dynamics control will perfectly compensate for all nonlinearities occurring in the system. Control approaches based on inverse models are well-known in the robotics community. For example, in motion control inverse dynamics models gain increasing popularity, as the rising of computational power allows to compute more complex models for real-time control. The concept of feedback linearization is another, more general way to derive inverse dynamics control laws and offers possibly more applications for learned models (Slotine and Li, 1991; Luca and Lucibello, 1998).

2.2.1.3 Mixed Models

In addition to forward and inverse models, there are also methods which combine both types of models. As pointed out in preceding sections, modeling the forward relationship is well-defined, while modeling the inverse relation can lead to an ill-posed problem. The basic idea behind the combination of forward and inverse models is that the information encoded in the forward model can help to resolve the non-uniqueness, i.e., the ill-posedness, of the inverse model. A typical ill-posed inverse modeling problem is the inverse kinematics of redundant robots. Given a joint configuration $\boldsymbol{q}$, the task space position $\boldsymbol{x}$ can be determined exactly (i.e., the forward kinematic model is well-defined), but there may be many possible joint configurations $\boldsymbol{q}$ for a given task space position $\boldsymbol{x}$ (i.e., the inverse model could have infinitely many solutions and their combination is not straightforward). Thus, when naively learning such inverse mapping from data, the learning algorithm will potentially average over non-convex sets of the solutions. The resulting mapping will contain invalid solutions which can cause poor prediction performance. The ill-posedness of the inverse model can be resolved when it is combined with the forward model, such that the composite of these models yields an identity mapping (Jordan and Rumelhart, 1992). In this case, the inverse model will provide those solutions which are consistent with the unique forward model.

The mixed model approach, i.e., the composite of forward and inverse models, was first poposed in conjunction with the distal teacher learning approach (Jordan and

Rumelhart, 1992), which will be discussed in details in Section 2.2.2.3. The proposed mixed models approach has subsequently evoked significant interests and has been extensively studied in the field of neuroscience (Wolpert and Kawato, 1998; Kawato, 1999). Furthermore, the mixed model approach is supported by evidence that the human cerebellum can be modeled using forward-inverse composite models, such as MOSAIC (Wolpert *et al.*, 1998; Bhushan and Shadmehr, 1999). While the mixed models have become well-known in the neuroscience community, the application of such models in robot control is not yet widespread. Pioneering work on mixed models in the control community can be found in (Narendra *et al.*, 1995; Narendra and Balakrishnan, 1997), where the mixed models are used for model reference control of an unknown Markov jump system. Even though mixed model approaches are not widely used in control, with the appearance of humanoid robots in the last few years, biologically inspired robot controllers are gaining more popularity. Controllers based on mixed models may present a promising approach (Haruno *et al.*, 2001; Peters and Schaal, 2008; Ting *et al.*, 2008).

2.2.1.4 Operator Models

The models introduced in preceding sections are mainly used to predict a single future state or action. However, in problems such as open-loop control, one would like to have information of the system for the next t-steps in the future. This problem is the multi-step ahead prediction problem, where the task is to predict a sequence of future values without the availability of output measurements in the horizon of interest. We call the models which are employed to solve this problem as *operator models*. It turns out that such operator models are difficult to develop because of the lack of measurements in the prediction horizon. A straightforward idea is to apply single-step prediction models t times in sequence, in order to obtain a series of future predictions. However, this approach seems to be susceptible to the error accumulation problem, i.e., errors made in the past are propagated into future predictions. An alternative to overcome the error accumulation problem is to apply autoregressive models which are extensively investigated in time-series prediction (Akaike, 1970). Here, the basic idea is to use models which employ past predicted values to predict future outcomes.

Combining operator models with control was originally motivated by the need of extension of forward models for multi-step predictions (Keyser and Cauwenberghe, 1980). In more recent work, variations of traditional ARX and ARMAX models for nonlinear cases have been proposed for operator models (Billings *et al.*, 1989; Mosca *et al.*, 1989). However, operator models based on some parametric structures, such as ARX or ARMAX have shown to have difficulties when the system becomes more sophisticated. The situation is even worse in the presence of noise or complex nonlinear dynamics. These difficulties give reasons to employ nonparametric operator models for multi-step predictions (Kocijan *et al.*, 2004; Girard *et al.*, 2002).

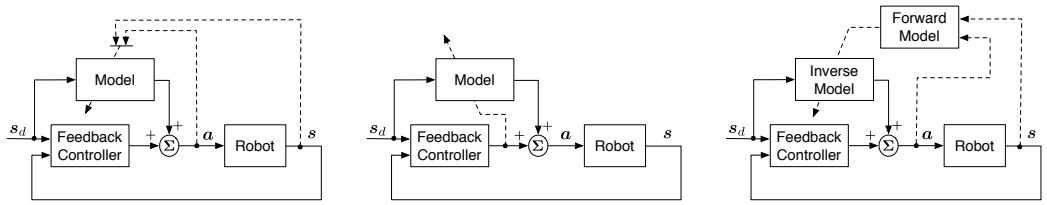
Model Type	Learning Architecture	Example Applications
Forward Model	Direct Modeling	Prediction, Filtering, Learning simulations, Optimization
Inverse Model	Direct Modeling (if invertible), Indirect Modeling	Inverse dynamics control, Computed torque control, Feedback linearization control
Mixed Model	Direct Modeling (if invertible), Indirect Modeling, Distal-Teacher	Inverse kinematics, Operational space control, Multiple-model control
Operator Model	Direct Modeling	Planning, Optimization, Model predictive control, Delay compensation

Table 2.1: Overview on model types associated with applicable learning architectures and example applications.

2.2.2 Learning Architectures

In previous section, we have presented different prediction problems that require different types of models. Depending on what quantities are observed, we need different models to predict the missing information. Here, we distinguished between forward models, inverse models, mixed models and operator models. A central question when incorporating these models into a learning control framework is how to learn and adapt the models while they are being used. We will distinguish between direct modeling, indirect modeling and the distal teacher approach. Table 2.1 shows an overview of model types associated with applicable learning architectures.

In *direct modeling* approaches, we attempt to learn a direct mapping from input data to output data. However, direct model learning is only possible, when the relationship between inputs and outputs is well-defined. In case the input-output relationship is ill-posed (for example, when learning an inverse model) indirect and distal learning techniques can be used instead. When employing *indirect modeling* techniques, the model learning is driven by an error measure. For example, the feedback error of a controller can be used in this case. In *distal teacher* learning approaches, the inverse model of the system is used for control, and the learning of this inverse model is guided by a forward model. Figure 2.3 illustrates these three learning architectures. Compared to the direct modeling approaches, the indirect model learning and the distal teacher learning are *goal-directed* learning techniques. Instead of learning a global mapping from inputs to outputs (as done by direct



(a) Direct Modeling (b) Indirect Modeling (c) Distal Teacher Learning

Figure 2.3: Learning architectures in model learning applied to control. (a) In the direct modeling approach, the model is learned directly from the observations. (b) Indirect modeling approximates the model using the output of the feedback controller as error signal. (c) In the distal teacher learning approach, the inverse model's error is determined using the forward model. The resulting composite model will converge to an identity transformation.

modeling), goal-directed learning approximates a particular solution in the output space. Due to this property, indirect and distal teacher learning approaches can be used for learning when confronting with an ill-posed mapping problem.

2.2.2.1 Direct Modeling

Direct learning is probably the most straightforward way to obtain a model but is not always applicable. In this learning paradigm, the model is directly learned by observing the inputs and outputs. It is probably the most frequently employed learning technique for model approximation in control. Direct model learning can be implemented using most standard regression techniques, such as least square methods (Ljung, 2004), neural networks (Haykin, 1999; Steil, 2004) or statistical approximation techniques (Rasmussen and Williams, 2006; Schölkopf and Smola, 2002).

An early example of direct learning in control was the self-tuning regulator that generates a forward model and adapts it online (Aström and Wittenmark, 1995). Using the estimated forward model, the self-tuning regulator will estimate an appropriate control law online. However, the forward model in the traditional self-tuning regulator has a fixed parametric structure and, hence, it cannot deal automatically with unknown nonlinearities (Mosca *et al.*, 1989; Coito and Lemos, 1991). The main reason why parametric models need to be used in direct modeling techniques is that such model parametrization is necessary for a convenient formulation of the control law and, more importantly, for the rigorous stability analysis. As parametric models are often too restrictive for complex robot systems, learned models with more degrees of freedom are needed, such as neural networks or other machine learning techniques (Vempaty *et al.*, 2009; Layne and Passino, 1996). However, sophisticated learning algorithms for control are difficult to analyze if not impossible. Most work on the analysis of learning control has been done in neural control (Patino *et al.*, 2002) and model predictive control (Gu and Hu, 2002; Negenborn *et al.*, 2005;

Nakayama *et al.*, 2008). The operator model is an extension of forward models to multi-step prediction used in model predictive control. Direct learning of operator models has been done with neural networks (Chow *et al.*, 1998). In more recent work, probabilistic methods are employed to learn such operator models (Girard *et al.*, 2002; Kocijan *et al.*, 2004).

Inverse models can also be learned in a direct manner if the inverse mapping is well-defined. A well-known example is the inverse dynamics model required by computed torque and inverse dynamics control (Craig, 2004; Spong *et al.*, 2006). If direct modeling is applicable, learning becomes straightforward and can be achieved using standard regression techniques (Schaal *et al.*, 2002; Nguyen-Tuong and Peters, 2009; Cao *et al.*, 2006). Early work in learning inverse models for control attempts to adapt a parametric form of the rigid body dynamics model. This model is linear in its parameters and, hence, it can be estimated from data straightforwardly using linear regression (Atkeson *et al.*, 1986; Burdet *et al.*, 1997).

In practice, the estimation of dynamics parameters is not always straightforward. It is hard to create sufficiently rich data sets so that physically plausible parameters can be identified (Nakanishi *et al.*, 2008), and when identified online, additional persistent excitation issues occur (Narendra and Annaswamy, 1987). Due to the fixed parametric structures, these models are not capable of capturing the structured nonlinearities of the real inverse dynamics. Physically implausible values often rise from such structural errors that result from a lack of representation for unmodeled nonlinearities. Hence, more sophisticated models have been introduced for learning inverse dynamics, such as neural networks (Cao *et al.*, 2006; Patino *et al.*, 2002) or statistical nonparametric models (Schaal *et al.*, 2002; Nguyen-Tuong and Peters, 2009, 2010b). There have also been attempts to combine parametric rigid body dynamics model with nonparametric model learning for approximating the inverse dynamics (Nguyen-Tuong and Peters, 2010c). Similar to inverse dynamics control, feedback linearization control can also be used in conjunction with direct model learning. Again, the nonlinear dynamics can now be approximated using neural networks or other nonparametric learning methods (Ge *et al.*, 1998; Nakanishi *et al.*, 2005). Stability analysis of feedback linearization control with learned models is possible, extending the cases where the nonlinear dynamics could not be canceled perfectly (Nakanishi *et al.*, 2005).

While direct learning is mostly associated with learning a single type of model, it can also be applied to mixed models. The mixed model approach (e.g., combining inverse and forward models) find its application in learning control for multiple-module systems. The basic idea is to decompose a (probably) complex system into many simpler sub-systems which can be controlled individually (Narendra and Balakrishnan, 1997). The problem is how to choose an appropriate architecture for the multiple controllers, and how to switch between the multiple modules. Employing the idea of mixed models, each controller module consists of a pair of inverse and forward models. The intuition is that the controller can be considered as an inverse model, while the forward model is essentially used to switch between the different modules (Wolpert and Kawato, 1998). Such multiple pairs of forward and

inverse models can be learned directly from data using gradient-descent methods or expectation-maximization (Haruno *et al.*, 2001).

2.2.2.2 Indirect Modeling

Direct model learning works well when the input-output relationship is well-defined as in inverse dynamics. However, there can be situations where this relationship is not well-defined, such as in the differential inverse kinematics problem. In such cases, these models can often still be learned indirectly. One indirect modeling technique which can solve some of such ill-posed problems is known as feedback error model learning (Kawato, 1990). Feedback error learning relies on the output of a feedback controller that is used to generate the error signals employed to learn the feedforward controller, see Figure 2.3 (b). In several problems, such as feedforward inverse dynamics control (Craig, 2004), this feedback error learning approach can be understood particularly well. If the inverse dynamics model in the feedforward loop is a perfect model, the corresponding feedback controller is silent (and its output will be zero). If the feedback error is non-zero, it corresponds to the error of the inverse model in the feedforward loop (Craig, 2004). The intuition behind feedback error learning is that by minimizing the feedback errors for learning the inverse model, the feedback control term will decrease as the model converges. Thus, the inverse model will describe the inverse dynamics of the system, while the feedback control part becomes irrelevant.

Compared to the direct model learning, feedback error learning is a goal-directed model learning approach resulting from the minimization of feedback errors. Here, the model learns a *particular* output solution for which the feedback error is zero. Another important difference between feedback error learning and direct learning is that feedback error learning has to perform online, while direct model learning can be done both online and offline.

Feedback error learning is biologically motivated due to its inspiration from cerebellar motor control (Kawato, 1999). It has been further developed for control with robotics applications, originally employing neural networks (Shibata and Schaal, 2001; Miyamoto *et al.*, 1988). Feedback error learning can also be used with various nonparametric learning methods (Nakanishi and Schaal, 2004). Conditions for the stability of feedback error learning control in combination with nonparametric approaches have also been investigated (Nakanishi and Schaal, 2004).

Indirect model learning can also be used in the mixed model approach (Gomi and Kawato, 1993). Here, the attempt has been made to combine the feedback error learning with the mixture of experts architecture to learn multiple inverse models for different manipulated objects, where the inverse models are learned indirectly using the feedback error learning approach (Gomi and Kawato, 1993). In this approach, the forward model is used for training a gating network, as it is well-defined. The gating network subsequently generates a weighted prediction of the multiple inverse models, where the predictors determine the locally responsible models.

2.2.2.3 Distal Teacher Learning

The distal teacher learning approach was motivated by the necessity to learn general inverse models, which suffer from the problem of ill-posedness (Jordan and Rumelhart, 1992). Here, the non-uniqueness of the inverse model is resolved when combined with an unique forward model. The forward model is understood as a "distal teacher" which guides the learning of the inverse model. In this setting, the unique forward model is employed to determine the errors made by the inverse model during learning. The aim is to learn the inverse model such that this error is minimized. The intuition behind this approach is that the inverse model will learn a correct solution for a *particular* desired trajectory when minimizing the error between the output of the forward model and the input of the inverse model. Thus, the inverse model will result in solutions that are consistent with the unique forward model.

The distal teacher approach has successfully learned particular solutions for multi-valued mappings, such as inverse kinematics of redundant robots (Jordan and Rumelhart, 1992). Similar to feedback error model learning (Kawato, 1990), distal teacher learning is also a goal-directed learning method applicable for various robot control scenarios. However, unlike the feedback error learning approach, distal teacher learning allows directly aiming at a globally consistent inverse model instead of local on-policy optimization. In practice, the distal teacher employs two interacting learning process: one process where the forward model is learned, and another process where the learned forward model is used for determining the error of the inverse model. In the original distal learning approach, the inverse model's output is validated by the forward model, as the composite of these models yields an identity mapping if perfectly learned (Jordan and Rumelhart, 1992).

The distal learning approach is particularly suitable for control when combining with the mixed models, as it naturally incorporates the mixed model principle. The distal teacher learning approach with mixed models has motivated a number of follow-up projects with several robot control applications (D'Souza *et al.*, 2001; Peters and Schaal, 2008; Ting *et al.*, 2008).

2.2.3 Challenges and Constraints

In previous sections, we give an overview of different types of models and how these models can be incorporated into various learning architectures. However, employing machine learning methods, such as statistical methods (Rasmussen and Williams, 2006; Schölkopf and Smola, 2002; Vijayakumar and Schaal, 2000) for learning such models, is not always straightforward. Several important problems need to be tackled in order to customize general learning algorithms for an application in robotics. In this section, we give an overview of these problems and discuss how these problems can be approached in order to bring machine learning algorithms into robotics. In particular, we consider the problems that arise from *data*, from employed *algorithms* and from *real-world* challenges. These are summarized in Table 2.2.

Data Challenges	Algorithmic Constraints	Real-World Challenges
High-dimensionality, Smoothness, Richness of data, Noise, Outliers, Redundant data, Missing data	Incremental updates, Real-time, Online learning, Efficiency, Large data sets, Prior knowledge, Sparse data	Safety, Robustness, Generalization, Interaction, Stability, Uncertainty in the environment

Table 2.2: Challenges of real-world problems for machine learning

2.2.3.1 Data Challenges

In order to learn a "good" model for applications in robotics, the sampled data has to cover a large region of the model state space though, of course, it can never cover the complete state space. For the purpose of generalization, the generated data has to be sufficiently rich, i.e., it should contain as much information about the system as possible. Thus, generating large and rich data sets for model learning is an essential step (which is sometimes not easy in practice). This step often requires additional excitation of the robot system during data generation, which is known as persistent excitation in classical system identification (Narendra and Annaswamy, 1987). For several systems, the persistent excitation condition is naturally given, such as aircraft systems. For other systems, the persistent excitation condition has to be generated artificially, e.g., by adding small random movements into to the output of the system. For learning inverse dynamics, for example, rich data can be sampled from trajectories by approximately executing desired random point-to-point and rhythmic movements (Swevers *et al.*, 1997; Schaal and Sternad, 1998).

The data used for model learning has to be sufficiently smooth, which is a key assumption for most of machine learning methods. However, there are many applications in robotics where the approximated functions are known to be non-smooth. For example, stiction-friction models are often non-smooth. Such non-smooth functions can sometimes be approximated using kernel methods. As a kernel implicitly incorporates the smoothness of the approximated function, special kernels can be defined in order to take the expected non-smoothness in account (Schölkopf and Smola, 2002; Rasmussen and Williams, 2006). An example of such types of kernels are Matern-kernels which are widely used in Bayesian inference methods (Rasmussen and Williams, 2006). Discontinuities in a non-smooth function can also be approximated by local models and by learning how to switch discontinuously between these local models (Toussaint and Vijayakumar, 2005).

Robot systems that have a large number of DoFs pose a challenging problem due to the high dimensionality of the generated data. For example, in inverse dynamics the learning methods have to deal with data in a space with $4n$ dimensions, where

n is the number of DoFs. This difficulty can be tackled by preprocessing the data using dimensionality reduction, which is a well-studied technique in machine learning (Tenenbaum *et al.*, 2000; Roweis and Saul, 2000). The application of dimensionality reduction is based on the insight that the useful information in the data often lies on a low-dimensional manifold of the original input space. Dimensionality reduction methods have proven to be a powerful method for model learning in high dimensional robot systems (Schaal *et al.*, 2002; Hoffman *et al.*, 2009).

As the data is sampled over a possibly long period of time in many robotics applications (Thrun and Mitchell, 1995), problems of redundant and irrelevant data can occur. In such cases, redundant and irrelevant data can bias the model which severely hurts the generalization. Here, the data can be filtered in an appropriate way by selecting only data points that are informative for the learning process. This filtering step can be combined with the learning step, for example, by using information criteria for inserting and deleting data points (Nguyen-Tuong and Peters, 2010b; Engel *et al.*, 2002).

Noise and outliers have always been a challenging problem for machine learning and for robot learning. Naively learning a model from noisy data can make the model fit the noise (i.e., an over-fit) and, thus, adulterate the model learning performance. In the past decade, considerable efforts have been made in the machine learning community to deal with this problem. In particular, regularization frameworks are developed based on statistical learning theory. The basic idea is to constrain the model to be learned in an appropriate way, attenuating the contributions made by the noisy components in the data. This lead to a variety of model learning methods, such as support vector regression or Gaussian process regression (Smola and Schölkopf, 2004; Rasmussen, 1996). One step in these methods is to estimate the noise-level in the data represented by a regularization parameter, which can either be done by cross-validation or by maximizing the marginal likelihood function (Seeger, 2004). By controlling the noise in the data, these methods can significantly improve the generalization performance.

2.2.3.2 Algorithmic Constraints

There are two scenarios for model learning in robotics: large data and small data. In the first case, learning algorithms have to deal with massive amounts of data, such as in learning inverse dynamics. In this scenario, the algorithms need to be efficient in terms of computation without sacrificing the learning accuracy (Bottou *et al.*, 2007). In the second scenario, there is only few data available for learning, as the data generation may be too tedious and expensive. Here, we need algorithms which allow us to incorporate additional prior knowledge in order to improve the learning performance in the presence of sparse data (Schölkopf *et al.*, 1997; Krupka and Tishby, 2007).

For machine learning techniques, fast, real-time computation is challenging. Standard model learning approaches, such as Gaussian process regression, for example, scale cubically in the number of training data, preventing a straightforward usage in

robotics. Sparse and reduced set methods smartly reduce the size of training data and, thus, decrease the computational effort for the learning and the prediction step (Candela and Rasmussen, 2005). In recent years, there have been serious efforts to speed up machine learning algorithms with efficient implementations using, for example, parallel computation (Genov *et al.*, 2003).

Online learning is also a strong requirement of the domain of robotics. Most of machine learning methods are developed for learning in batch mode, i.e., offline learning using pre-sampled data sets, while online learning requires incremental approximation of the model. However, online learning has found increasing interest over the last few years giving rise to a number of real-time online machine learning approaches, such as in (Vijayakumar *et al.*, 2005; Nguyen-Tuong and Peters, 2009). A major motivation for online model learning is the insight that it is not possible to cover the complete state space with data beforehand, but that only the interesting state space regions are only known during the execution. Thus, online model learning will require incremental acquisition of knowledge and, possibly, even partial forgetting of the recorded information in order to cope with errors as well as change. Furthermore, online learning presents an essential step towards continuous adaptation to a changing world which is essential to make robots more autonomous (Thrun and Mitchell, 1995).

Incorporating prior knowledge into the learning process can be obtained straightforwardly, when statistical learning approaches are used. In kernel methods, prior knowledge can be specified by feature vectors which can be used to define appropriate kernels (Schölkopf *et al.*, 1997). In contrast, probabilistic frameworks allow one to specify priors to capture a priori information (Rasmussen and Williams, 2006). If prior knowledge is given as a parametric model, it can be inserted into nonparametric models in a straightforward way, yielding semiparametric learning approaches (Nguyen-Tuong and Peters, 2010c; Smola *et al.*, 1998). Semiparametric models have shown to be capable in learning competitive models, when only few data is available (Nguyen-Tuong and Peters, 2010c).

2.2.3.3 Real-World Challenges

In order to ensure safe interaction of robots with human beings in everyday life, machine learning algorithms developed for robotics applications have to be fail-safe or at least have to minimize the risk of damage. For critical applications, such as medical or service robotics, robustness and reliability are among the most important criteria which have to be fulfilled by model learning. Model learning can become more robust when feature selection is employed as a preceding step. Feature selection methods remove irrelevant and redundant data and, thus, make the model learning more robust. Feature selection has an inherent connection to sparse and reduced set methods, where the purpose is to filter out information which is crucial for the model approximation (Csato and Opper, 2002; Schölkopf *et al.*, 1999). Feature selection has been an active research field in machine learning for many years and has now found its ways to several robot applications both in robot vision and control (Kröse

et al., 2001; Nguyen-Tuong and Peters, 2010b).

Robustness also requires the learning algorithm to deal with missing data. This problem is encountered in every robotics set-up where the sensor information is imperfect, such as in terrain modeling or autonomous navigation. In particular, measurement errors often result in missing data. In the recent years, the problem of missing data has attracted much attention with the rise of probabilistic learning methods. As the models are probabilistic, it is now possible to infer the missing "pieces" in the training data (Titsias and Lawrence, 2010). A further advantage of the probabilistic methods consists of a straightforward way to assign its uncertainty to each predicted value and, hence, make it easier to deal with insufficient data.

2.2.4 Applicable Regression Methods

In preceding section, we summarize some problems which need to be overcome when employing machine learning in robotics. In this section, we approach the model learning from the algorithmic point of view and provide an overview of how models can be learned using machine learning techniques. We will focus on modern statistical methods for learning models from data. However, connections to other popular learning approaches such as neural networks will also be discussed (Steil, 2004; Haykin, 1999).

In general, model learning is a supervised learning approach. It is assumed that the input $\boldsymbol{x}$ and output $\boldsymbol{y}$ are given, where the true output data is corrupted by noise $\boldsymbol{\epsilon}$, i.e.,

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}\,. \tag{2.6}$$

Approximating the underlying function f is the goal of supervised learning methods. Given unknown input data the learned model should be able to provide precise predictions of the output values. Different supervised learning techniques make different assumptions on how to model the function f.

Here, we distinguish between *global* and *local* techniques used to model the underlying function f. Global regression techniques model the underlying function f using *all* observed data to construct a single global prediction model (Hastie *et al.*, 2001). In contrast to global methods, the local regression estimates the underlying function f *within* a local neighborhood around a query input point. Beyond the local and global types of model learning, there are also approaches which combine both ideas. An example of such hybrid approaches is the mixture of experts (Jacobs *et al.*, 1991; Nowlan and Hinton, 1991). Here, the data is partitioned into smaller local models in a first step and, subsequently, a gating network is used to fuse these local models for global prediction. Mixture of experts approaches have been further embedded into the Bayesian framework giving rise to a number of Bayesian hybrid approaches such as committee machines (Treps, 2000), mixtures of Gaussian models (Treps, 2001) or infinite mixtures of experts (Rasmussen and Ghahramani, 2002).

Method	Type	Mode	Online	Complexity	Learning Applications
Locally Weighted Projection Regression (Vijayakumar and Schaal, 2000)	Local	Incremental	Yes	$\mathcal{O}(n)$	Inverse dynamics (Schaal *et al.*, 2002), Foothold quality model (Kalakrishnan *et al.*, 2009)
Local Gaussian Process Regression (Nguyen-Tuong and Peters, 2009)	Local	Incremental	Yes	$\mathcal{O}(m^2)$	Inverse dynamics (Nguyen-Tuong and Peters, 2009)
Gaussian Mixture Model (Jacobs *et al.*, 1991)	Semi-Local	Batch	No	$\mathcal{O}(Mn)$	Human motion model (Calinon *et al.*, 2010)
Bayesian Comittee Machine (Treps, 2000)	Semi-Local	Batch	No	$\mathcal{O}(m^2n)$	Inverse dynamics (Rasmussen and Williams, 2006)
Sparse Gaussian Process Regression (Csato and Opper, 2002)	Global	Incremental	Yes	$\mathcal{O}(n^2)$	Transition dynamics (Rottmann and Burgard, 2009), Task model (Grollman and Jenkins, 2008)
Gaussian Process Regression (Seeger, 2004)	Global	Batch	No	$\mathcal{O}(n^3)$	Terrain model (Plagemann *et al.*, 2008), State estimation model (Ko and Fox, 2009)
Support Vector Regression (Schölkopf and Smola, 2002)	Global	Batch	No	$\mathcal{O}(n^2)$	ZMP control model (Ferreira *et al.*, 2007), Grasp stability model (Pelossof *et al.*, 2004)
Incremental Support Vector Machine (Ma *et al.*, 2005)	Global	Incremental	Yes	$\mathcal{O}(n^2)$	Inverse dynamics (Choi *et al.*, 2007)

Table 2.3: A large variety of machine learning methods have been applied in model learning for robotics. We distinguish between global and local methods, as well as semi-local methods which combine both approaches. The methods differ in the training mode and their online capabilities. For computational complexity, n denotes the total number of training points, m is number of data points in a local model, and M is the number of local models. We further provide several application examples for model learning in robotics.

2.2.4.1 Global Regression

A straightfoward way to model the function f in Equation (2.6) is to assume a parametric structure, such as linear or polynomial models or multilayer perceptron neural networks, and, subsequently, fit the model parameters using training data (Hastie *et al.*, 2001; Haerdle *et al.*, 2004; Haykin, 1999). However, fixing the model with a parametric structure beforehand may not suffice to explain the sampled data, which motivates nonparametric model learning frameworks (Haerdle *et al.*, 2004; Schölkopf and Smola, 2002; Rasmussen and Williams, 2006). In the modern parametric and nonparametric regression, the function f is usually modeled as

$$f(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}) \,, \tag{2.7}$$

where $\boldsymbol{\theta}$ is a weight vector and $\boldsymbol{\phi}$ is a nonlinear function projecting the input $\boldsymbol{x}$ into some high-dimensional spaces. The basic idea behind nonparametric regression is that the optimal model structure should be obtained from the training data. Hence, the size of the weight vector $\boldsymbol{\theta}$ is not fixed but can increase with the number of training data points in most statistical learning methods. It determines the structure of the model given in Equation (2.7). Compared to nonparametric statistical approaches, other popular function approximation methods such as neural networks fix the model structure beforehand (Haykin, 1999). For instance, in traditional neural network learning, the number of nodes and their connections have to be determined before starting the training procedure (Haykin, 1999). It is worth noting that there have been attempts to put artificial neural networks into the Bayesian framework (MacKay, 1992; Neal, 1996) and, thus, establishing the connection between the two learning approaches. It has also been observed that certain neural networks with one hidden layer converge to a Gaussian process prior over functions (Neal, 1996).

In kernel methods, the model structure is determined by the model complexity (Schölkopf and Smola, 2002). Learning a model includes finding a tradeoff between the model's complexity and the best fit of the model to the observed data. It is desirable to have a model which is simple but at the same time can explain the data well. Using kernel methods, the weight vector $\boldsymbol{\theta}$ in Equation (2.7) can be first expanded in term of n training data points and, subsequently, regularized in an optimization step. Intuitively, the weight vector $\boldsymbol{\theta}$ represents the complexity of the resulting model.

Having a close link to the kernel framework, probabilistic regression methods additionally provide a Bayesian interpretation of nonparametric kernel regression (Rasmussen and Williams, 2006). Instead of expanding the weight vector as done in kernel methods, probabilistic methods place a prior distribution over $\boldsymbol{\theta}$. The prior parameters can be subsequently obtained by optimizing the corresponding marginal likelihood. Thus, the trade-off between data-fit and model complexity can be obtained in a straightforward and plausible way (Rasmussen and Williams, 2006).

Kernel and probabilistic methods have proven to be successful tools for model learning over the last decade, resulting in a number of widely applied regression

methods, such as support vector regression or Gaussian process regression (Schölkopf *et al.*, 2000; Smola and Schölkopf, 2004; Rasmussen, 1996). These methods are known to be capable of being applicable to high-dimensional data. They can also deal well with noisy data, as the noise is taken in account indirectly by regularizing the model complexity. Furthermore, they are relatively easy to use, as several black-box implementations are available. However, the major drawback of these methods are the computational complexity. For instance, for Gaussian process regression the complexity scales cubically in term of number of training data. Thus, one active research line in machine learning is to reduce the computational cost of those approaches. Due to several advances in customizing machine learning techniques for robotics, kernel and probabilistic regression techniques have aroused increasing interests and have found their ways to several robotics applications, such as robot control (Nguyen-Tuong and Peters, 2010b), sensor modeling (Plagemann *et al.*, 2007) or state estimation (Ko and Fox, 2009).

2.2.4.2 Local Learning

The basic idea of local regression techniques is to estimate the underlying function f *within* a local neighborhood around a query input point $\boldsymbol{x}_q$. The data points in this neighborhood can then be used to predict the outcome for the query point. Generally, local regression models can be obtained by minimizing the following cost function J using n training data points

$$J = \sum_{k=1}^{n} w\left(\frac{\boldsymbol{x}_k - \boldsymbol{x}_q}{h}\right)\left(\boldsymbol{y}_k - \hat{f}(\boldsymbol{x}_k)\right)^2 . \tag{2.8}$$

As indicated by the Equation (2.8), the essential ingredients for a local regression model are the neighborhood function w and the local model $\hat{f}$. The neighborhood function w, which is controlled by a width parameter h, basically measures the distance between a query point $\boldsymbol{x}_q$ to the points in the training data. The local model $\hat{f}$ described the function structure used to approximate f within the neighborhood around $\boldsymbol{x}_q$ (Cleveland and Loader, 1996; Fan and Gijbels, 1996). Depending on the complexity of the data, different function structures can be assumed for the local model $\hat{f}$, such as a linear or a polynomial model. The open-parameters of $\hat{f}$ can be estimated straightforwardly by minimizing J with respect to these parameters. However, the choice of the neighborhood function and its width parameter is more involved. Several techniques have been suggested for estimating the width parameters for a given w, including the minimization of the leave-one-out cross validation error and adaptive bandwidth selection (J.Fan and I.Gijbels, 1995; Moore and Lee, 1994).

Because of their simplicity and computational efficiency, local regression techniques have become widespread in model learning for robotics (Moore, 1992; Atkeson *et al.*, 1997b; Tevatia and Schaal, 2008). In the last decade, novel local regression approaches have been further developed in order to cope with the demands in many robotics real-time applications, such as locally weighted projection regression

(Atkeson *et al.*, 1997a; Vijayakumar *et al.*, 2005). Inspired by local regression techniques, these methods first employ a partitioning of the input space into smaller local regions, for which locally linear models are approximated. In addition to being computational efficient, local methods can deal with less smooth functions and do not require the same smoothness and regularity conditions as global regression methods. However, it has been shown in practice that local methods suffer from problems induced by high-dimensional data, as notions of locality break down for sparse, high-dimensional data. Furthermore, the learning performance of local methods may be sensitive to noise and heavily depends on the way how the input space is partitioned, i.e., the configuration of the neighborhood function w. These problems still present an active research topic (Edakunni *et al.*, 2007; Ting *et al.*, 2008).

Several attempts have been made to scale local regression models to higher dimensional problems as required for many modern robotics systems. For example, locally weighted projection regression combines local regression with dimensionality reduction by projecting the input data into a lower dimensional space, where local regression is employed afterwards (Vijayakumar *et al.*, 2005). Other methods combine nonparametric probabilistic regression, such as Gaussian process regression, with the local approaches while exploiting the strength of probabilistic methods for model learning in high-dimensions (Nguyen-Tuong and Peters, 2009; Ting *et al.*, 2008; Urtasun and Darrell, 2008).

2.3 Application of Model Learning

In this section, we discuss three case studies on model learning in different robot applications. The presented cases illustrate several different aspects of model learning discussed in previous sections. This list of examples is obviously not exhaustive but gives an overview on possible applications of model learning in robotics. In Section 2.3.1, an application of forward models is illustrated from several examples. In Sections 2.3.2 and 2.3.3, we highlight cases where inverse models and mixed models are useful.

2.3.1 Simulation-based Optimization

As forward models directly describe the dynamic behavior of the system, learning such models has evoked much attention in the field of robot control for a long time. A key application of learned forward models is the optimization of control problems. In this situation, a policy that has been optimized for a hand-crafted model is likely to be biased by the large model errors, while optimization on the real system is too costly. Hence, policy optimization based on learned forward models is an interesting alternative.

Atkeson et al. (Atkeson and Morimoto, 2002; Jacobson and Mayne, 1973) were among the first to explore this approach using differential dynamic programming for optimizing open-loop control policies. The basic idea of Atkeson et al. is to use receptive field-weighted regression (a type of locally weighted regression) to learn the

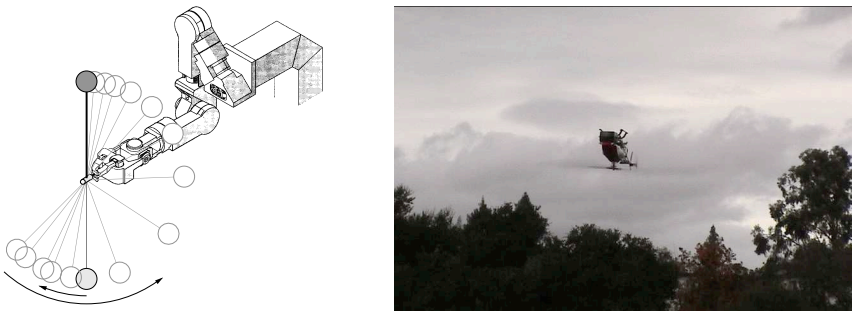


Figure 2.4: Learning the pendulum swing up task (with permission of Stefan Schaal) and learning inverted flight with a helicopter (with permission of Andrew Y. Ng). In both cases, the forward model is used to learn the dynamics of the system for policy optimization.

models of both cost and state transition. Differential dynamic programming locally linearizes the state transition model and generates a local quadratic approximation of the cost. These approximations are used to improve an open-loop policy where the linearizations are also updated after every policy update (Atkeson and Morimoto, 2002). Atkeson et al. used the method to learn the underactuated pendulum swing up task, where a pole is attached to the endeffector of the robot and maximal torque has been limited to a fixed value. The goal of the robot is to bring the pole from an hanging to an upright position. Hence, the system needs to "pump" energy into the pendulum in order to swing it up. Subsequently, it needs to limit the energy so that it can stabilize the pole at the upright position (Atkeson and Schaal, 1997). Starting from an unconstrained human demonstration, the robot was able to successfully learn the swing up and balance task after three trials (Atkeson and Schaal, 1997). The local trajectory optimization technique has been further extended to biped robot walking (Morimoto *et al.*, 2003). More recently, a related approach with parametric function approximation has been applied by Abbeel et al. to learn autonomous helicopter flight (Abbeel *et al.*, 2007). The authors also reported fast convergence of this approach when learning different moves for the helicopter, such as flip and roll movements (Abbeel *et al.*, 2007).

While Atkeson (Atkeson and Morimoto, 2002) and Abbeel (Abbeel *et al.*, 2007) used the forward model as an implicit simulator, Ng et al. (Ng *et al.*, 2004) use it as an explicit simulator (as originally suggested by Sutton (Sutton, 1991) in form of the DYNA model). Here, the forward model acts as a simulator for generating complete trajectories or roll-outs. The predictions of the forward model are further perturbed by Gaussian noise with a repeating, fixed noise history (e.g., by resetting the random seed, a trick well-known in simulation optimization (Glynn, 1987) and known as PEGASUS (Ng and Jordan, 2000)). This perturbation step is required to make the system more robust to noise and model errors, while the re-use of the noise history limits the variance in the policy updates (which results in a major speed-up). This simulator based on a learned forward model is used for generating

complete roll-outs from a similar start-state set for a control policy. The resulting performance of different control policy can be compared, which allows policy updates both by pair-wise comparison or by gradient-based optimization. The approach has been able to successfully stabilize an helicopter in an inverted flight. A few similar examples in model predictive control (see Section 2.2.1 for an explanation) exist which employ a variety of different learning approaches such as statistical learning methods (Kocijan *et al.*, 2004), neural networks (Akesson and Toivonen, 2006) both for robot navigation (Gu and Hu, 2002) and helicopter control (Wan and Bogdanov, 2001).

2.3.2 Approximation-based Inverse Dynamics Control

Inverse models, such as inverse dynamics models, are frequently used in robotics (Spong *et al.*, 2006). Inverse dynamics models characterize the required joint torques $\boldsymbol{\tau} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ to achieve a desired configuration $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$, where $\boldsymbol{q}$ is the joint position and $\dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ are the corresponding velocity and acceleration, respectively. In classical robot control, inverse dynamics model can be analytically given by the rigid body dynamics model

$$\boldsymbol{\tau}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}\right) = \mathbf{M}\left(\boldsymbol{q}\right)\ddot{\boldsymbol{q}} + \mathbf{F}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right) , \tag{2.9}$$

where $\mathbf{M}(\boldsymbol{q})$ is the generalized inertia matrix of the robot, $\mathbf{F}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is a vector defined by the forces, such as Coriolis forces, centripetal forces and gravity. This model relies on series of assumptions, such as that the robot's links are rigid, there is no friction or stiction, nonlinearities of actuators are negligible etc. (Spong *et al.*, 2006). However, modern robotics systems with highly nonlinear components, such as hydraulic tubes or elastic cable drive, can no longer be accurately modeled with the rigid body dynamics. An inaccurate dynamics model can lead to severe losses in control performance and, in the worst case, instability. Instead of modeling the inverse dynamics manually based on physics and human insight, an inverse dynamics model can be learned from sampled data. Such a data-driven approach has the advantage that all nonlinearities encoded in the data will be approximated by the model (Nguyen-Tuong and Peters, 2009).

As the inverse model is a unique mapping from joint space into torque space, learning inverse dynamics models is a standard regression problem. In order to generalize the learned models for a larger state space and to adapt the models for time dependent changes in the dynamics, real-time online learning becomes necessary. However, online learning poses difficult challenges for any regression method. These problems have been addressed by real-time learning methods such as locally weighted projection regression (Vijayakumar and Schaal, 2000). Nguyen-Tuong et al. (Nguyen-Tuong and Peters, 2009, 2010b) combine the basic ideas behind the locally weighted projection regression method with the global probabilistic Gaussian process regression method (Rasmussen and Williams, 2006), attempting to combine the efficiency of local learning with the high accuracy of Gaussian process regression. The resulting method has shown to be capable for real-time online learning of the

Figure 2.5: The 7-DoF anthropomorphic SARCOS arm used in learning inverse dynamics, inverse kinematics and operational space control (with permission of Stefan Schaal).

robot's inverse dynamics. Instead of using local models, data sparsification methods can be employed to speed up kernel regression approaches for real-time learning (Nguyen-Tuong and Peters, 2010b).

It is worth noting that such inverse dynamics model learning approaches can also be motivated from a biological point of view. Kawato et al (Kawato, 1999) have suggested that the cerebellum may act as an inverse dynamics model. Motivated by this insight, Shibata et al. (Shibata and Schaal, 2001) proposed a biologically inspired vestibulo-oculomotor control approach based on feedback-error learning of the inverse dynamics model. The problem is to stabilize the gaze in the face of perturbations due to body movement, where the cerebellum is known to predict the forces required to keep image stabilized on the retina (based on efferent motor signals and inputs from the vestibular system). In this work, Shibata et al. employ the locally weighted projection regression approach to learn the inverse model of the eye dynamics online (Vijayakumar and Schaal, 2000). The same locally weighted projection regression technique has also been used to learn a complete inverse dynamics model for the humanoid DB (Schaal *et al.*, 2002).

2.3.3 Learning Operational Space Control

Operational space control (OSC) allows the robot to follow given desired trajectories in the task space (Khatib, 1987; Nakanishi *et al.*, 2008). Before explaining how OSC can be viewed as a learning problem, we will review the most basic form of OSC laws from a classical robotics point of view.

The relationship between the task space and joint space of the robot is defined by the classical forward kinematics models $\boldsymbol{x} = f(\boldsymbol{q})$, where $\boldsymbol{q}$ denotes a joint space configuration and $\boldsymbol{x}$ represents the corresponding task space position. The task space velocity and acceleration are given by $\dot{\boldsymbol{x}} = \mathbf{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{x}} = \dot{\mathbf{J}}(\boldsymbol{q})\dot{\boldsymbol{q}} + \mathbf{J}(\boldsymbol{q})\ddot{\boldsymbol{q}}$, respectively, where $\mathbf{J}(\boldsymbol{q}) = \partial f / \partial \boldsymbol{q}$ is the Jacobian. To obtain the joint torques required for the

task space control, the dynamics model (as given in Equation (2.9)) is needed. The direct combination of dynamics and kinematics model yields one possible operational space control law

$$\boldsymbol{u} = \mathbf{M}\mathbf{J}_W^+(\ddot{\boldsymbol{x}} - \dot{\mathbf{J}}\dot{\boldsymbol{q}}) + \mathbf{F}\,, \tag{2.10}$$

where $\mathbf{J}_W^+$ denotes the weighted pseudo-inverse of $\mathbf{J}$ (Sciavicco and Siciliano, 1996; Peters *et al.*, 2008) and $\boldsymbol{u}$ represents the joint control torques. Equation (2.10) can be employed to generate the joint torques necessary for tracking a task space trajectory determined by a reference task-space acceleration (Nakanishi *et al.*, 2008). Note that the practical application of such control laws often requires further terms, such as the so-called null-space control law for joint-space stabilization (Nakanishi *et al.*, 2008).

As discussed in Section 2.3.2, dynamics models can be hard to obtain and, thus, learning can be an attractive alternative. Learning an operational space control law corresponds to learning an inverse model such as $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}) \rightarrow \boldsymbol{u}$ (Peters and Schaal, 2008). However, learning such OSC models is an ill-posed problem, as there are infinitely many inverse models possible. For example, we could create infinitely many solutions for a redundant robot analytically by simply varying the metric W of the weighted pseudo-inverse in Equation (5.11). As the space of possible solutions is not convex, such OSC models cannot be learned straightforwardly using regression models (unless the system has no redundant degrees of freedom). Similar problems appear in the limited case of differential inverse kinematics (D'Souza *et al.*, 2001).

Both D'Souza et al. (D'Souza *et al.*, 2001) and Peters et al. (Peters and Schaal, 2008) noticed that local linearizations of the mapping in Equation (5.11) will always form a convex space. Hence, data sets generated by such systems will also be locally convex. They furthermore realized that the predictive abilities of forward models allows determining local regions, where a locally consistent forward model can be learned. However, extremely different and inconsistent local models may form, depending on the local data distribution. As a result, the global consistency can no longer be ensured. This insight lead to two significantly different approaches. D'Souza (D'Souza *et al.*, 2001) created a differential inverse kinematics learning system (i.e., a limited special case of an operational space control law) and chose to bias the learning system by selectively generating data. However, he also realized that such an approach will generically be limited by the trade-off between this intentional bias and the inverse model's accuracy. Peters et al. (Peters and Schaal, 2008) treated learning of complete operational space control laws. They realized that a re-weighting of the data using an additional reward function allows regularizing these inverse models towards a globally consistent solution. Inspired by a result in analytical OSC (Peters *et al.*, 2008), they suggest appropriate reward functions both for learning full OSC and differential inverse kinematics. The resulting mapping was shown to work on several robot systems. Ting et al. (Ting *et al.*, 2008) presented an implementation of Peters et al.'s (Peters and Schaal, 2008) approach with modern Bayesian machine learning which sped up the performance significantly.

Instead of learning a direct OSC control law as done by Peters et al. (Peters and Schaal, 2008), Salaün et al. (Salaun *et al.*, 2009) attempt to learn the well-defined differential forward kinematics as a first step (i.e., learning the Jacobian) using locally weighted projection regression. The corresponding weighted pseudo-inverse of the Jacobian is subsequently computed using SVD decomposition techniques. The obtained differential inverse kinematics model is combined with an inverse dynamics model to generate the joint space control torques (Salaun *et al.*, 2009). Approximating inverse kinematics models has also been investigated using neural network learning (Reinhart and Steil, 2008; Jordan and Rumelhart, 1992). More recently, Reinhart et al. employ a reservoir computing architecture which allows to jointly learn the forward and inverse kinematics.

2.4 Conclusion

In this chapter, we gave a survey of past and current research activities of model learning for robotics. First, we discussed different types of models and how these models can be incorporated in various learning architectures. Subsequently, we pointed out what kind of problems these architectures and the domain of robotics imply for the learning methods. We further discussed the challenges that arise from the application of learning methods in the domain of robotics. An overview on how models can be learned using machine learning techniques with a focus on statistical regression methods was given. In several case studies, we showed where the model learning scenarios have been used successfully.

Model learning is gaining increasing interest in the robotics community, as physically modeling of complex, modern robot systems become more difficult. It can be a useful alternative to manual pre-programming, as the model is estimated directly from measure data. Unknown nonlinearities are taken in account, while they are neglected by the standard physics-based modeling techniques. Model learning has been shown to be an efficient tool in variety of applications. Especially, for robot control model learning, it has proven to be useful, as it provides accurate models of the system allowing the application of compliant, energy-efficient controls.

3 Model Learning with Local Gaussian Process Regression

Nonparametric regression methods, such as Gaussian process regression (GPR) or locally weighted projection regression (LWPR) offer a flexible framework for approximating models from data. While GPR has been known for its learning accuracy and its ability to deal with high-dimensional data, it suffers from the extensive computational complexity. In this chapter, inspired by the idea of local learning methods we present a local approximation to the standard GPR, called local GPR (LGP), appropriate for real-time online model learning. The approach combines the strengths of both regression methods, i.e., the high accuracy of GPR and the fast speed of LWPR. The approach is shown to have competitive learning performance for high-dimensional data while being sufficiently fast for real-time learning.

3.1 Introduction

Precise models of technical systems can be crucial in technical applications. In robot tracking control, only a well-estimated inverse dynamics model allow both high accuracy and compliant, low-gain control. For complex robots such as humanoids or light-weight arms, it is often hard to analytically model the system sufficiently well and, thus, modern regression methods can offer a viable alternative (Schaal *et al.*, 2002; Vijayakumar *et al.*, 2005). However, highly accurate regression methods such as Gaussian process regression (GPR) suffer from high computational cost, while fast real-time learning algorithms such as locally weighted projection regression (LWPR) are not straightforward to use, as they require manual adjustment of many data dependent parameters.

In this chapter, we attempt to combine the strengths of both approaches, i.e., the high accuracy and comfortable use of GPR with the fast learning speed of LWPR (Nguyen-Tuong *et al.*, 2008c). We will proceed as follows: firstly, we briefly review both model based control as well as two nonparametric learning approaches, i.e., standard GPR and LWPR. We will discuss the necessity of estimating the inverse dynamics model and further discuss the advantages of both regression methods in learning this model. Subsequently, we describe our local Gaussian process models (LGP) approach and related work. We show that LGP inherits both the precision of GPR and a higher speed similar to LWPR.

In Section 3.4, the learning accuracy and performance of the presented LGP approach will be compared with several relevant regression methods, e.g., standard GPR (Rasmussen and Williams, 2006), ν-support vector regression (ν-SVR)

(Schölkopf and Smola, 2002), sparse online GP (OGP) (Csato and Opper, 2002) and LWPR (Vijayakumar *et al.*, 2005; Schaal *et al.*, 2002). The applicability of the LGP for low-gain model based tracking control and real-time learning is demonstrated on a Barrett whole arm manipulator (WAM) (Nguyen-Tuong *et al.*, 2008b). We can show that its tracking performance exceeds analytical models (Craig, 2004) while remaining fully compliant.

3.1.1 Background

Model based control, e.g., computed torque control (Spong *et al.*, 2006), enables high speed and compliant robot control while achieving accurate control with small tracking errors for sufficiently precise robot models. The controller is supposed to move the robot that is governed by the system dynamics (Spong *et al.*, 2006)

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \boldsymbol{\epsilon}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}) = \mathbf{u}\,, \tag{3.1}$$

where $\mathbf{q},\ \dot{\mathbf{q}},\ \ddot{\mathbf{q}}$ are joint angles, velocities and accelerations of the robot, respectively, $\mathbf{u}$ denotes the applied torques, $\mathbf{M}(\mathbf{q})$ the inertia matrix of the robot and $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ Coriolis and centripetal forces, $\mathbf{G}(\mathbf{q})$ gravity forces and $\boldsymbol{\epsilon}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ represents nonlinearities of the robot which are not part of the rigid body dynamics due to hydraulic tubes, friction, actuator dynamics, etc.

The model based tracking control law determines the joint torques $\mathbf{u}$ necessary for following a desired trajectory $\mathbf{q}_d,\ \dot{\mathbf{q}}_d,\ \ddot{\mathbf{q}}_d$ using a dynamics model while employing feedback in order to stabilize the system. For example, the dynamics model of the robot can be used as a feed-forward model that predicts the joint torques $\mathbf{u}_{\mathrm{FF}}$ required to perform the desired trajectory, while a feedback term $\mathbf{u}_{\mathrm{FB}}$ ensures the stability of the tracking control with a resulting control law of $\mathbf{u}=\mathbf{u}_{\mathrm{FF}}+\mathbf{u}_{\mathrm{FB}}$. The feedback term can be a linear control law such as $\mathbf{u}_{\mathrm{FB}}=\mathbf{K}_p\mathbf{e}+\mathbf{K}_v\dot{\mathbf{e}}$, where $\mathbf{e}=\mathbf{q}_d-\mathbf{q}$ denotes the tracking error and $\mathbf{K}_p,\mathbf{K}_v$ position-gain and velocity-gain, respectively. If an accurate model in the form of Equation (3.1) can be obtained, e.g., for negligible unknown nonlinearities $\boldsymbol{\epsilon}$, the resulting feedforward term $\mathbf{u}_{\mathrm{FF}}$ will largely cancel the robots nonlinearities (Spong *et al.*, 2006).

3.1.2 Problem Statement

For complex robots such as humanoids or light-weight arms, it is often hard to model the system sufficiently well using the rigid body dynamics. Unknown nonlinearities $\boldsymbol{\epsilon}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ such as flexible hydraulic tubes, complex friction, gear boxes, etc, couple several degrees of freedom together and result in highly altered dynamics. In particular, for the Barrett WAM several degrees of freedom are jointly actuated in a differential setup, and as a result, there is a complex friction function. Additionally, several spinning drives are in different reference frames from the actuated joint while only one can be measured resulting in effects such as reflective inertias. Thus, the dynamics can no longer be fully captured by standard rigid body dynamics (Townsend, 2007).

Such unknown nonlinearities can dominate the system dynamics and deteriorate the analytical model (Nakanishi *et al.*, 2005). The resulting tracking error needs to be compensated using large gains (Spong *et al.*, 2006). High feedback gains prohibit compliant control and, thus, make the robot less safe for the environment. High-gain control also causes many practical problems such as actuator saturation, excitation of unmodeled dynamics, increase energy consumption, and may result in large tracking errors in presence of noise, etc. To avoid high-gain feedback, it is essential to improve the accuracy of the dynamics model for predicting $\mathbf{u}_{\mathrm{FF}}$. Since $\mathbf{u}_{\mathrm{FF}}$ is a function of $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$, it can be obtained with supervised learning using measured data. The resulting problem is a regression problem that can be solved by learning the mapping $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \mathbf{u}$ on sampled data (Burdet *et al.*, 1997; Schaal *et al.*, 2000; Nguyen-Tuong *et al.*, 2008a) and, subsequently, using the resulting mapping for determining the feedforward motor commands. As trajectories and corresponding joint torques are sampled directly from the real robot, learning the mapping will include all nonlinearities and not only the ones described in the rigid body model.

3.1.3 Challenges in Real-time Learning

Due to high computational complexity of nonlinear regression techniques, inverse dynamics models are frequently only learned offline for pre-sampled desired trajectories (Burdet *et al.*, 1997; Nguyen-Tuong *et al.*, 2008a). In order to take full advantage of a learning approach, online learning is absolute necessity as it allows the adaption to changes in the robot dynamics, load or the actuators. Furthermore, a training data set will never suffice for most robots with a large number of degrees of freedom and, thus, fast online learning is necessary if the trajectory leads to new parts of the state-space. However, for most real-time applications online model learning poses a difficult regression problem due to three constraints, i.e., firstly, the learning and prediction process should be very fast (e.g., learning needs to take place at a speed of 20-200Hz and prediction may take place at 200Hz up to 5kHz). Secondly, the learning system needs to be capable at dealing with large amounts of data (i.e., with data arriving at 200Hz, less than ten minutes of runtime will result in more than a million sampled data points). And, thirdly, the data arrives as a continuous stream, thus, the model has to be continuously adapted to new training examples over time.

3.2 Nonparametric Regression Methods

As any realistic inverse dynamics is a well-defined functional mapping of continuous, high-dimensional inputs to outputs of the same kind, we can view it as a regression problem. Given the input $\mathbf{x} \in \mathbb{R}^n$ and the target $\mathbf{y} \in \mathbb{R}^n$, the task of regression algorithms is to learn the mapping describing the relationship from input to target using samples. In this section, we will review the locally weighted projection regression (LWPR) and the Gaussian process regression (GPR). Locally-weighted projection regression is currently the standard real-time learning method in robot

control applications and has been shown to scale into very high-dimensional domains (Vijayakumar and Schaal, 2000; Schaal *et al.*, 2002; Vijayakumar *et al.*, 2005). However, it also requires skillful tuning of the meta parameters for the learning process in order to achieve competitive performance. Gaussian process regression on the other hand achieves a higher performance (Rasmussen and Williams, 2006; Seeger, 2004) with very little tuning but also suffers of a significantly higher computational complexity.

3.2.1 Regression with LWPR

LWPR predicts the target values by approximating them with a combination of M individually weighted locally linear models. The weighted prediction $\hat{y}$ is then given by $\hat{y} = \mathbb{E}\{\bar{y}_k|\mathbf{x}\} = \sum_{k=1}^{M} \bar{y}_k p(k|\mathbf{x})$. According to the Bayesian theorem, the probability of the model k given query point $\mathbf{x}$ can be expressed as

$$p(k|\mathbf{x}) = \frac{p(k,\mathbf{x})}{p(\mathbf{x})} = \frac{p(k,\mathbf{x})}{\sum_{k=1}^{M} p(k,\mathbf{x})} = \frac{w_k}{\sum_{k=1}^{M} w_k}. \tag{3.2}$$

Hence, we have

$$\hat{y}(\mathbf{x}) = \frac{\sum_{k=1}^{M} w_k \bar{y}_k(\mathbf{x})}{\sum_{k=1}^{M} w_k}, \tag{3.3}$$

with $\bar{y}_k = \bar{\mathbf{x}}_k^T \hat{\boldsymbol{\theta}}_k$ and $\bar{\mathbf{x}}_k = [(\mathbf{x} - \mathbf{c}_k)^T, 1]^T$, where w_k is the weight or attributed responsibility of the model, $\hat{\boldsymbol{\theta}}_k$ contains the estimated parameters of the model and $\mathbf{c}_k$ is the center of the k-th linear model. The weight w_k determines whether a data point $\mathbf{x}$ falls into the region of validity of model k, similar to a receptive field, and is usually characterized with a Gaussian kernel

$$w_k = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{c_k})^T \mathbf{D_k} (\mathbf{x} - \mathbf{c_k})\right), \tag{3.4}$$

where $\mathbf{D_k}$ is a positive definite matrix called the distance matrix. During the learning process, both the shape of the receptive fields $\mathbf{D_k}$ and the parameters $\hat{\boldsymbol{\theta}}_k$ of the local models are adjusted such that the error between the predicted values and the observed targets is minimal. The regression parameter $\hat{\boldsymbol{\theta}}_k$ can be computed incrementally and online using the partial least squares method (Vijayakumar and Schaal, 2000; Schaal *et al.*, 2002). The distance matrix $\mathbf{D_k}$ determines the size and shape of each local model; it can be updated incrementally using leave-one-out cross validation (Vijayakumar *et al.*, 2005).

3.2.2 Regression with standard GPR

A powerful alternative for accurate function approximation in high-dimensional space is Gaussian process regression (GPR) (Rasmussen and Williams, 2006). Given a set of n training data points $\{\mathbf{x}_i, y_i\}_{i=1}^n$, we would like to learn a function $f(\mathbf{x}_i)$

transforming the input vector $\mathbf{x}_i$ into the target value y_i given a model $y_i = f(\mathbf{x}_i) + \epsilon_i$, where ϵ_i is Gaussian noise with zero mean and variance σ_n^2 (Rasmussen and Williams, 2006). As a result, the observed targets can also be described by a Gaussian distribution $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I})$, where $\mathbf{X}$ denotes the set containing all input points $\mathbf{x}_i$ and $\mathbf{K}(\mathbf{X}, \mathbf{X})$ the covariance matrix computed using a given covariance function. Gaussian kernels are probably the frequently used covariance functions (Rasmussen and Williams, 2006) and are given by

$$k(\mathbf{x}_p, \mathbf{x}_q) = \sigma_s^2 \exp\left(-\frac{1}{2}(\mathbf{x}_p - \mathbf{x}_q)^T \mathbf{W} (\mathbf{x}_p - \mathbf{x}_q)\right), \tag{3.5}$$

where σ_s^2 denotes the signal variance and $\mathbf{W}$ represents the widths of the Gaussian kernel. Other choices for possible kernels can be found in (Schölkopf and Smola, 2002; Rasmussen and Williams, 2006). The joint distribution of the observed target values and predicted value $f(\mathbf{x}_*)$ for a query point $\mathbf{x}_*$ is given by

$$\begin{bmatrix} \mathbf{y} \\ f(\mathbf{x}_*) \end{bmatrix} \sim \mathcal{GP}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} & \boldsymbol{k}(\mathbf{X}, \mathbf{x}_*) \\ \boldsymbol{k}(\mathbf{x}_*, \mathbf{X}) & k(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right). \tag{3.6}$$

Conditioning the joint distribution yields the predicted mean value $f(\mathbf{x}_*)$ with the corresponding variance $V(\mathbf{x}_*)$

$$\begin{aligned} f(\mathbf{x}_*) &= \boldsymbol{k}_*^T \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{y} = \boldsymbol{k}_*^T \boldsymbol{\alpha}, \\ V(\mathbf{x}_*) &= k(\mathbf{x}_*, \mathbf{x}_*) - \boldsymbol{k}_*^T \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} \boldsymbol{k}_*, \end{aligned} \tag{3.7}$$

with $\boldsymbol{k}_* = \boldsymbol{k}(\mathbf{X}, \mathbf{x}_*)$, $\mathbf{K} = \mathbf{K}(\mathbf{X}, \mathbf{X})$ and $\boldsymbol{\alpha}$ denotes the so-called prediction vector. The hyperparameters of a Gaussian process with Gaussian kernel are given by $\boldsymbol{\theta} = [\sigma_n^2, \sigma_f^2, \mathbf{W}]$ and remain the only open parameters. Their optimal value for a particular data set can be automatically estimated by maximizing the log marginal likelihood using standard optimization methods such as Quasi-Newton methods (Rasmussen and Williams, 2006).

3.2.3 Comparison of these Approaches

The major drawback of LWPR is the currently necessary manual adjustment of the metaparameters[1] required for the update of the kernel width $\mathbf{D_k}$ and the regression vector $\hat{\boldsymbol{\theta}}_k$. These values are highly data dependent making it difficult to find an optimal set of parameters. Furthermore, as linear models are used in LWPR, a large number of local models may be required to achieve competitive prediction accuracy, since only relatively small regions can be fit using such linear models. Nevertheless, LWPR is the fastest and most task-appropriate real-time learning algorithm for inverse dynamics to date; currently, it can be considered the state of the art in real-time learning. On the other hand, GPR is more comfortable to apply while often

[1] Current work by Ting et al. (Ting *et al.*, 2008) indicates that automatic metaparameter estimation may be possible on the long run.

achieving a higher prediction accuracy. All open parameters of a Gaussian process model, i.e., the hyperparameters $\boldsymbol{\theta}$, can be automatically adjusted by maximizing the marginal likelihood. As a result, GPR is relatively easy and flexible to use. However, the main limitation of standard GPR is that computational complexity scales cubically with the number of training examples. This drawback prevents standard GPR from applications which need large amounts of training data and require fast computation, e.g., model online learning for robot control.

3.3 Local Gaussian Process Regression

Model learning with GPR suffers from the expensive computation of the inverse matrix $(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}$ which yields a cost of $\mathcal{O}(n^3)$, see Equation (3.7). Inspired by locally weighted regression (Schaal *et al.*, 2002; Vijayakumar *et al.*, 2005), we propose a method for speed-up the training and prediction process by partitioning the training data in local regions and learning an independent Gaussian process model (as given in Section 3.2.2) for each region. The number of data points in the local models is limited, where insertion and removal of data points can be treated in a principled manner. The prediction for a query point is performed by weighted average similar to LWPR. For partitioning and weighted prediction we use a kernel as similarity measure. Thus, our algorithm consists out of three stages: (i) clustering of data, i.e., insertion of new data points into the local models, (ii) learning of corresponding local models and (iii) prediction for a query point.

3.3.1 Partitioning of Training Data

Clustering input data can be performed efficiently using a similarity measure between the input point $\mathbf{x}$ and the centers of the respective local models. From a machine learning point of view, the similarity or proximity of data points can be defined in terms of a kernel. Kernel functions represent the dot product between two vectors in the feature space and, hence, naturally incorporate the similarity measure between data points. The clustering step described in this section results from the basic assumption that nearby input points are likely to have similar target values. Thus, training points that belong the same local region (represented by a center) are informative about the prediction for query points next to this local region.

A specific characteristic in this framework is that we take the kernel for learning the Gaussian process model as similarity measure w_i for the clustering process. If a Gaussian kernel is employed for learning the model, the corresponding measure will be

$$w_i\left(\mathbf{x}, \mathbf{c}_i\right) = \exp\left(-\frac{1}{2}\left(\mathbf{x} - \mathbf{c}_i\right)^T \mathbf{W}\left(\mathbf{x} - \mathbf{c}_i\right)\right), \tag{3.8}$$

where $\mathbf{c}_i$ denotes the center of the i-th local model and $\mathbf{W}$ a diagonal matrix represented the kernel width. Note that this measure will result in the same weighting as

Algorithm 1 Partitioning the training data with incremental model learning.

Input: new data point $\{\mathbf{x}_{\text{new}}, y_{\text{new}}\}$.
for $i=1$ **to** number of local models **do**
  Compute proximity to the i-th local model:
    $w_i = k\left(\mathbf{x}_{\text{new}}, \mathbf{c}_i\right)$
end for
Take the nearest local model:
  $v = \max_i w_i$
if $v > w_{\text{gen}}$ **then**
  Insert $\{\mathbf{x}_{\text{new}}, y_{\text{new}}\}$ into the nearest local model:
    $\mathbf{X}_{\text{new}} = [\mathbf{X}, \mathbf{x}_{\text{new}}]$, $\mathbf{y}_{\text{new}} = [\mathbf{y}, y_{\text{new}}]$
  Update the corresponding center:
    $\mathbf{c}_{\text{new}} = \text{mean}(\mathbf{X}_{\text{new}})$
  Update the Cholesky matrix and the
  prediction vector of local model:
    Compute $\boldsymbol{l}$ and l_*
    Compute $\mathbf{L}_{\text{new}}$
    If the maximum number of data points is reached
    delete another point by permutation.
    Compute $\boldsymbol{\alpha}_{\text{new}}$ by back-substitution
else
  Create new model:
    $\mathbf{c}_{i+1} = \mathbf{x}_{\text{new}}$, $\mathbf{X}_{i+1} = [\mathbf{x}_{\text{new}}]$, $\mathbf{y}_{i+1} = [y_{\text{new}}]$
  Initialize of new Cholesky matrix $\mathbf{L}$ and
  new prediction vector $\boldsymbol{\alpha}$.
end if

in LWPR, see Equation (3.4). It should be emphasized that for learning the Gaussian process model any admissible kernel can be used. Thus, the similarity measure for the clustering process can be varied in many ways, and, for example, the commonly used Matern kernel (Seeger, 2004) could be used instead of the Gaussian one. For the hyperparameters of the measure, such as $\mathbf{W}$ for Gaussian kernel, we use the *same* training approach as introduced in Section 3.2.2. Since the hyperparameters of a Gaussian process model can be achieved by likelihood optimization, it is straightforward to adjust the open parameters for the similarity measure. For example, we can subsample the available training data and, subsequently, perform the standard optimization procedure.

After computing the proximity between the new data point $\mathbf{x}_{new}$ and all available centers, the data point will be included to the *nearest* local model, i.e., the one with the maximal value of w_i. As the data arrives incrementally over time, a new model with center $\mathbf{c}_{i+1}$ is created if all similarity measures w_i fall below a threshold w_{gen}. The new data point is then used as new center $\mathbf{c}_{i+1}$ and, thus, the number of local models will increase if previously unknown parts of the state space are visited. When

Algorithm 2 Prediction for a query point.

Input: query data point $\mathbf{x}$, M .
Determine M local models closest to $\mathbf{x}$.
for $i = 1$ **to** M **do**
  Compute proximity to the i-th local model:
    $w_i = k(\mathbf{x}, \mathbf{c}_i)$
  Compute local prediction using the k-th local model:
    $\bar{y}_i(\mathbf{x}) = \mathbf{k}_i(\mathbf{x})^T \boldsymbol{\alpha}_i$
end for
Compute weighted prediction using M local models:
  $\hat{y}(\mathbf{x}) = \sum_{i=1}^{M} w_i \bar{y}_i(\mathbf{x}) / \sum_{k=1}^{M} w_i$.

a new data point is assigned to a particular i-th model, i.e., $\max_i w_i(\mathbf{x}) > w_{\text{gen}}$ the center $\mathbf{c}_i$ will be updated to the mean of corresponding local data points.

3.3.2 Incremental Update of Local Models

During online learning, we have to deal with an endless stream of data (e.g., at a 500 Hz sampling rate we get a new data point every 2 ms and have to treat 30 000 data points per minute). In order to cope with the real-time requirements, the maximal number of training examples needs to limited so that the local models do not end up with the same complexity as a standard GPR regression. Since the number of acquired data points increases continuously over time, we can enforce this limit by incrementally deleting old data points when newer and better ones are included. Insertion and deletion of data points can be achieved using first order principles, for example, maximizing the information gain while staying within a budget (e.g., the budget can be a limit on the number of data points). Nevertheless, while the update of the target vector $\mathbf{y}$ and input matrix $\mathbf{X}$ can be done straightforwardly, the update of the covariance matrix (and implicitly the update of the prediction vector $\boldsymbol{\alpha}$, see Equation (3.7)) is more complicated to derive and requires thorough analysis given here.

The prediction vector $\boldsymbol{\alpha}$ can be updated incrementally by directly adjusting the Cholesky decomposition of the Gram matrix $(\mathbf{K} + \sigma_n^2 \mathbf{I})$ as suggested in (Seeger, 2007b). For doing so, the prediction vector can be rewritten as $\mathbf{y} = \mathbf{L}\mathbf{L}^T \boldsymbol{\alpha}$, where the lower triangular matrix $\mathbf{L}$ is a Cholesky decomposition of the Gram matrix. Incremental insertion of a new point is achieved by adding an additional row to the matrix $\mathbf{L}$.

Proposition 3.3.1 *If* $\mathbf{L}$ *is the Cholesky decomposition of the Gram matrix* $\mathbf{K}$ *while* $\mathbf{L}_{new}$ *and* $\mathbf{K}_{new}$ *are obtained by adding additional row and column, such that*

$$\mathbf{L}_{new} = \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \boldsymbol{l}^T & l_* \end{bmatrix}, \quad \mathbf{K}_{new} = \begin{bmatrix} \mathbf{K} & \boldsymbol{k}_{new}^T \\ \boldsymbol{k}_{new} & k_{new} \end{bmatrix}, \tag{3.9}$$

with $\boldsymbol{k}_{new} = \boldsymbol{k}(\mathbf{X}, \mathbf{x}_{new})$ *and* $k_{new} = k(\mathbf{x}_{new}, \mathbf{x}_{new})$*, then* $\boldsymbol{l}$ *and* l_* *can be computed by solving*

$$\mathbf{L}\boldsymbol{l} = \boldsymbol{k}_{new} \tag{3.10}$$

$$l_* = \sqrt{k_{new} - \|\boldsymbol{l}\|^2} \tag{3.11}$$

Proof Multiply out the equation $\mathbf{L}_{new}\mathbf{L}_{new}^T = \mathbf{K}_{new}$ and solve for $\boldsymbol{l}$ and l_*. ∎

Since $\mathbf{L}$ is a triangular matrix, $\boldsymbol{l}$ can be determined from Equation (3.10) by substituting it back in after computing the kernel vector $\boldsymbol{k}_{new}$. Subsequently, l_* and the new prediction vector $\boldsymbol{\alpha}_{new}$ can be determined from Equation (3.11), where $\boldsymbol{\alpha}_{new}$ can be achieved by twice back-substituting while solving $\mathbf{y}_{new} = \mathbf{L}_{new}\mathbf{L}_{new}^T\boldsymbol{\alpha}_{new}$. If the maximal number of training examples is reached, an old data point has to be deleted every time when a new point is being included. The deletion of the m-th data point can be performed efficiently using a permutation matrix $\mathbf{R}$ and solving $\mathbf{y}_{new} = \mathbf{R}\ \mathbf{L}_{new}\mathbf{L}_{new}^T\mathbf{R}\ \boldsymbol{\alpha}_{new}$, where $\mathbf{R} = \boldsymbol{I} - (\boldsymbol{\delta}_m - \boldsymbol{\delta}_n)(\boldsymbol{\delta}_m - \boldsymbol{\delta}_n)^T$ and $\boldsymbol{\delta}_i$ is a zero vector whose i-th element is one (Seeger, 2007b). In practice, the new data point is inserted as a first step to the *last* row (n-th row) according to Equation (3.9) and, subsequently, the m-th data point is removed by adjusting $\mathbf{R}$. The partitioning and learning process is summarized in Algorithm 1. The incremental Cholesky update is very efficient and can be performed in a numerically stable manner as discussed in detail in (Seeger, 2007b).

Due to the Cholesky update formulation, the amount of computation for training can be limited due to the incremental insertion and deletion of data points. The main computational cost for learning the local models is dominated by the incremental update of the Cholesky matrix which yields $\mathcal{O}(N_l^2)$, where N_l presents the number of data points in a local model. Importantly, N_l can be set in accordance with the computational power of the available real-time computer system.

3.3.3 Prediction using Local Models

The prediction for a mean value $\hat{y}$ is performed using weighted averaging over M local GP predictions $\bar{y}_i$ for a query point $\mathbf{x}$ similar to LWPR. The weighted prediction $\hat{y}$ is then given by

$$\hat{y} = \frac{\sum_{i=1}^{M} w_i \bar{y}_i}{\sum_{i=1}^{M} w_i}\,. \tag{3.12}$$

Thus, each local GP prediction $\bar{y}_i = \boldsymbol{k}(\mathbf{X}_i, \mathbf{x})^T\boldsymbol{\alpha}_i$ is additionally weighted by the similarity $w_i\,(\mathbf{x}, \mathbf{c}_i)$ between the corresponding center $\mathbf{c}_i$ and the query point $\mathbf{x}$. The search for M local models can be quickly done by evaluating the proximity between the query point $\mathbf{x}$ and all model centers $\mathbf{c}_i$. The prediction procedure is summarized in Algorithm 2.

3.3.4 Relation to Previous Work

Many attempts have been made to reduce the computational cost of GPR, most of them follow either the strategy of creating (i) sparse Gaussian processes (SGP) or follow a (ii) mixture of experts (ME) approach. In a SGP, the whole input space is approximated using a smaller set of "inducing inputs" (Candela and Rasmussen, 2005; Csato and Opper, 2002; Grollman and Jenkins, 2008). Here, the difficulty lies in the choice of the appropriate set of inducing inputs that essentially summarizes the original input space (Rasmussen and Williams, 2006). In contrast to SGP, the ME approach divides the whole input space in smaller subspaces employing a gating network which activates responsible Gaussian process experts (Rasmussen and Ghahramani, 2002; Snelson and Ghahramani, 2007). Thus, the computational cost of the matrix inversion is significantly reduced due to a much smaller number of data points within an expert. The ME performance depends largely on the number of experts for a particular data set. To reduce the impact of this problem, (Rasmussen and Ghahramani, 2002) allows the learning process to infer the required number of experts for a given data set using a Dirichlet process to determine the appropriate gating network. The proposed algorithm has approximately a complexity of $\mathcal{O}(n^3/M)$ for training and $\mathcal{O}(n^2 d)$ for adapting the gating network parameters, where M denotes the number of experts, n the total number of training data and d the dimension of input vector.

The presented LGP approach is loosely related to the ME approximation. However, the gating network requires competition between the models for the data points, while the locality approach allows cooperation (Schaal. and Atkeson, 1996). Particularly important is the fact that we re-use the kernel measure as similarity measure which results in two advantages, firstly, the metric parameters can be derived directly by optimization procedure which makes it more comfortable and flexible for using. Secondly, the evaluation of the metric can be performed very fast enabling it for real-time application. However, it shows that clustering in higher dimensional space is not always straightforward to perform with the kernel similarity metric. In our experience, partitioning of training data can be done quite well up to 20 dimensions. Since we can localize the training data in much lower spaces than learning the model, this obstacle can often be circumvented. We will discuss this issue in more detail in Section 3.4.

Compared with LWPR, one major advantage is that we use Gaussian process model for training the local models instead of linear models. Thus, we need significantly fewer local models to be competitive in learning accuracy. Gaussian process models have also shown to generalize the training data well and are easier to train as the open parameter can be obtained straightforwardly from the log marginal likelihood. However, a major drawback in comparison to LWPR is that the more complex models result in an increased computational cost.

Figure 3.1: 7-DoFs Barrett WAM arm used for data generation and experiments.

3.4 Learning Inverse Dynamics

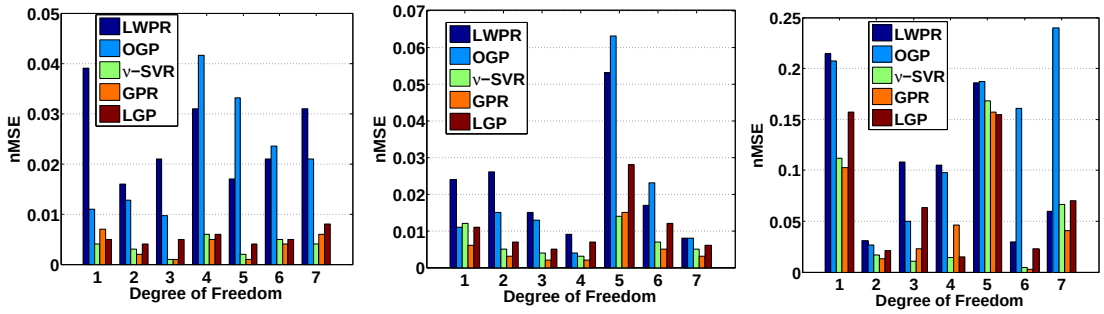
Learning models for control of high-dimensional systems in real-time is a difficult endeavor and requires extensive evaluation. For this reason, we have evaluated our algorithm using high-dimensional data taken from two real robots, e.g., the 7 degree-of-freedom (DoF) anthropomorphic SARCOS master arm and 7-DoF Barrett WAM (as shown in Figures 2.5 and 3.1, respectively), as well as physically realistic simulation using SL (Schaal, 2006). We compare the learning performance of LGP with the state-of-the-art in nonparametric regression, e.g., LWPR, ν-SVR, standard GPR and online Gaussian Process Regression (OGP) in the context of approximating inverse robot dynamics. For evaluating ν-SVR and GPR, we have employed the libraries (Chang and Lin, 2001) and (Seeger, 2007a), respectively. The code for LGP contained also parts of the library (Seeger, 2007a).

3.4.1 Learning Accuracy Comparison

For comparing the prediction accuracy of our proposed method in the setting of learning inverse dynamics, we use three data sets, (i) SL simulation data (SARCOS model) as described in (Nguyen-Tuong *et al.*, 2008a) (14094 training points and 5560 test points), (ii) data from the SARCOS master arm (13622 training points and 5500 test points) (Vijayakumar *et al.*, 2005) as well as (iii) a data set generated from our Barrett arm (13572 training points, 5000 test points).

Given samples $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}]$ as input, where $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ denote the joint angles, velocity and acceleration, respectively, and using the corresponding joint torques $\mathbf{y} = [\mathbf{u}]$ as targets, we have a well-defined, proper regression problem. The considered seven degrees of freedom (DoF) robot arms result in 21 input dimensions (i.e., for each joint, we have an angle, a velocity and an acceleration) and seven target or output dimensions (i.e., a single torque for each joint). The robot inverse dynamics model can be estimated separately for each DoF employing LWPR, ν-SVR, GPR, OGP and LGP, respectively.

The training examples for LGP can be partitioned either in the same input space



(a) **Approximation Error on SL data (SARCOS model)**

(b) **Approximation Error on SARCOS data**

(c) **Approximation Error on Barrett WAM data**

Figure 3.2: The approximation error is represented by the normalized mean squared error (nMSE) for each DoF (1–7) and shown for (a) simulated data from physically realistic SL simulation, (b) real robot data from an anthropomorphic SARCOS master arm and (c) measurements from a Barrett WAM. In all cases, LGP outperforms LWPR and OGP in learning accuracy while being competitive to ν-SVR and standard GPR. The small variances of the output targets in the Barrett data results in a nMSE that is a larger scale compared to SARCOS; however, this increase has no practical meaning and only depends on the training data.

where the local models are learned or in a subspace that has to be physically consistent with the approximated function. In the following, we localize the data depending on the position of the robot. Thus, the partitioning of training data is performed in a seven dimensional space (i.e., consisting of the seven joint angles). The localization should be performed in a low dimensional space, since with increasing input dimensions the partitioning of data may be difficult having negative effects on the learning performances. After determining the similarity metric w_k for all k local models in the partitioning space, the input point will be assigned to the *nearest* local model, i.e., the local model with the maximal value of distance measure w_k. For computing the localization, we will use the Gaussian kernel as given in Equation (5.15) and the corresponding hyperparameters are optimized using a subset of the training set.

Note that the choice of the limit value w_{gen} during the partitioning step is crucial for the performance of LGP and, unfortunately, is an open parameter requiring manual tuning. If w_{gen} is too small, a large number of local models will be generated with small number of training points. As these small models receive too little data for a stable GPR, they do not generalize well to unknown neighboring regions of the state space. If w_{gen} is large, the local models will include too many data points which either results in over-generalization or, if the number of admitted data points is enlarged as well, it will increase the computational complexity. Here, the training data is clustered in about 30 local regions ensuring that each local model has a sufficient amount of data points for high accuracy (in practice, roughly a hundred

data points for each local model suffice) while having sufficiently few that the solution remains feasible in real-time (e.g., on the test hardware, an Intel Core Duo at 2GHz, that implies the usage of up to a 1000 data points per local model). On average, each local model includes approximately 500 training examples, i.e., some models will not fill up while others actively discard data. This small number of training data points enables a fast training for each local model using the previously described fast Cholesky matrix updates.

Figure 3.2 shows the normalized mean squared error (nMSE) of the evaluation on the test set for each of the three evaluated scenarios, i.e., a physically realistic simulation of the SARCOS arm in Figure 3.2 (a), the real anthropomorphic SARCOS master arm in Figure 3.2 (b) and the Barrett WAM arm in Figure 3.2 (c). Here, the normalized mean squared error is defined by nMSE = Mean squared error/Variance of target. During the prediction on the test set using LGP, we take the most activated local models, i.e., the ones which are next to the query point.

When observing the approximation error on the test set shown in Figure 3.2(a-c), it can be seen that LGP generalizes well to the test data during prediction. In all cases, LGP outperforms LWPR and OGP while being close in learning accuracy to of the offline-methods GPR and ν-SVR. The mean prediction for GPR is determined according to Equation (3.7) where we pre-computed the prediction vector $\boldsymbol{\alpha}$ from training data. When a query point appears, the kernel vector $\mathbf{k}_*^T$ is evaluated for this particular point.

3.4.2 Comparison of Computation Speed for Prediction

The computation requirements of kernel-based regression can even be problematic for prediction in real-time, thus, it is an essential component of the LGP that it results in a substantial reduction of prediction latency rendering online prediction feasible even for large data sets. The duration of a prediction of the LGP is significantly lower than the one of GPR and ν-SVR, as only a small amount of local models in the vicinity of the current input data is needed during prediction. Thus, the complexity of the prediction operation is $\mathcal{O}(n)$ for a standard GPR (ν-SVR does not differ in complexity), it will become $\mathcal{O}(NM)$ for LGP, where n denotes the total number of training points, M number of local models used in the prediction and N number of data points in a local model. Note that usually $n >> NM$. The comparison of prediction speed is shown in Figure 5.1. Here, we train LWPR, ν-SVR, GPR and LGP on 5 different data sets with increasing training examples (1065, 3726, 7452, 10646 and 14904 data points, respectively). Subsequently, using the trained models we compute the average time needed to make a prediction for a query point for all 7 DoF. For LGP, we take the same number of local models in the vicinity for prediction as in last experiment. Since assuming a minimal prediction rate at 100 Hz (10 ms) in order to ensure system stability, data sets with more than 15000 points cannot be used with standard GPR or ν-SVR on an Intel Core Duo at 2GHz due to high computation demands for the prediction. In recent time, there are also approaches to speed up the prediction time for standard GPR (Shen *et al.*, 2005;

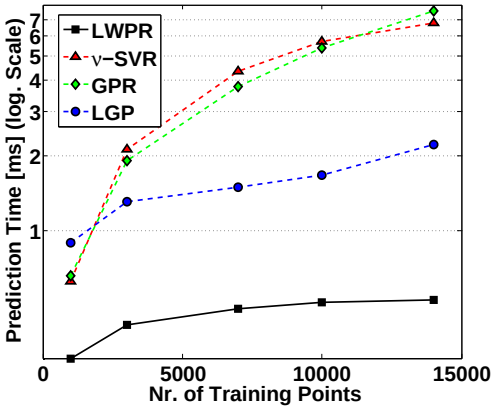


Figure 3.3: Average time in millisecond needed for prediction of 1 query point. The computation time is plotted logarithmically with respect to the number of training examples. The time as stated above is the required time for prediction of all 7 DoF performed sequentially. Here, LWPR presents the fastest method due to simple regression models. Compared to global regression methods such as standard GPR and ν-SVR, local GP makes significant improvement in term of prediction time. For this experiment, 3 local models are taken each time for prediction with LGP.

Suttorp and Igel, 2008). In (Shen *et al.*, 2005), for example, KD-trees are applied to find training data points next to the query point for prediction.

The results given in Figure 5.1 show that the computation time requirements of ν-SVR and GPR rises very fast with the size of training data set as expected. LWPR remains the best method in terms of computational complexity only increasing at a very low pace with the number of data points. However, as shown in Figure 5.1, the cost for LGP is significantly lower than the one for ν-SVR and GPR and increases at a much lower rate. The LGP prediction latency can be bounded by setting the number of local models needed for prediction, i.e., the parameter M. In practice, we need around 1000 data points in the neighborhood of the query point for prediction resulting in the usage of 2 or 3 local models. As shown by the results, LGP represents a compromise between learning accuracy and computational complexity. For large data sets (e.g., more than 5000 training examples), LGP reduces the prediction cost considerably in comparison to standard methods while still having a good learning performance.

3.5 Application in Model-based Robot Control

In this section, we use the inverse dynamics models learned in Section 3.4.1 for a model based tracking control task (Craig, 2004). Here, the model is used for predicting the feedforward torques $\mathbf{u}_{\text{FF}}$ necessary to execute a given desired trajectory $[\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d]$. First, we compare standard rigid body dynamics (RBD) models with several models learned offline on training data sets. For this offline learning com-

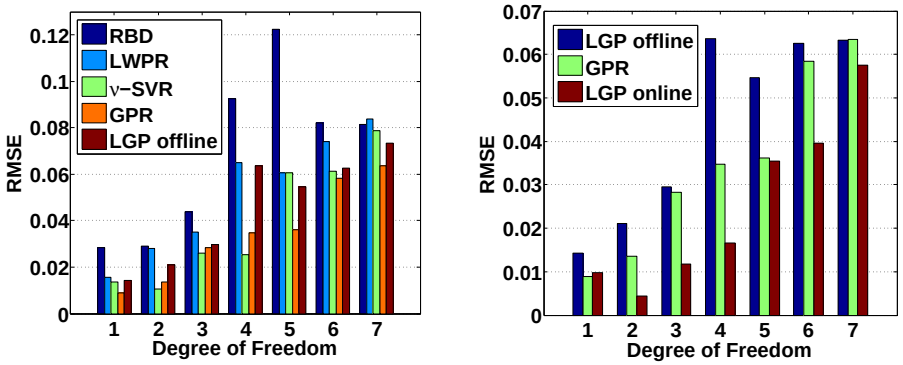
parison, we use LWPR, ν-SVR, standard GPR as well as our LGP as compared learning methods. We show that our LGP is competitive when compared with its alternatives. Second, we demonstrate that online learning is highly beneficial. During online learning, the local GP models are updated in real-time, and the online improvement during a tracking task outperforms the fixed offline model in comparison. Our goal is to achieve compliant tracking in robots without exception handling or force sensing but purely based on using low control gains. Our control gains are three orders of magnitude smaller than the manufacturers in the experiments and we can show that using good, learned inverse dynamics models we can still achieve compliant control. The accuracy of the model has a stronger effect on the tracking performance in this setting and, hence, a more precisely learned model will also results in a significantly lower tracking error.

3.5.1 Tracking using Offline Trained Models

For comparison with the learned models, we also compute the feedforward torque using rigid body (RB) formulation which is a common approach in robot control (Craig, 2004). The control task is performed in real-time on the Barrett WAM, as shown in Figure 3.1. As desired trajectory, we generate a test trajectory which is similar to the one used for learning the inverse dynamics models in Section 3.4.1. Figure 3.4 (a) shows the tracking errors on test trajectory for 7 DoFs. The error is computed as root mean square error (RMSE) which is a frequently used measure in time series prediction and tracking control. Here, LGP provides a competitive control performance compared to GPR while being superior to LWPR and the state-of-the art rigid body model.

3.5.2 Online Learning of Inverse Dynamics Models

In this section, we show that the LGP is capable of online adaptation while being used for predicting the required torques. Since the number of training examples in each local model is limited, the update procedure is sufficiently fast for real-time application. For doing so, we employ the joint torques $\mathbf{u}$ and the resulting robot trajectories $[\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}]$ as samples which are added to the LGP models online as described in Section 3.3.2. New data points are added to the local models until these fill up and, once full, new points replace previously existing data points. The insertion of new data point is performed with information gain (Seeger, 2005), while for the deletion we randomly take an old point from the corresponding local model. A new data point is inserted to the local model, if its information gain is larger than a given threshold value. In practice, this value is set such that the model update procedure can be maintained in real-time (the larger the information gain threshold, the more updates will be performed). Figure 3.4 (b) shows the tracking error after online learning with LGP in comparison with offline learned models. It can be seen that the errors are significantly reduced for LGP with online updates when compared to both standard GPR and LGP with offline learned models.



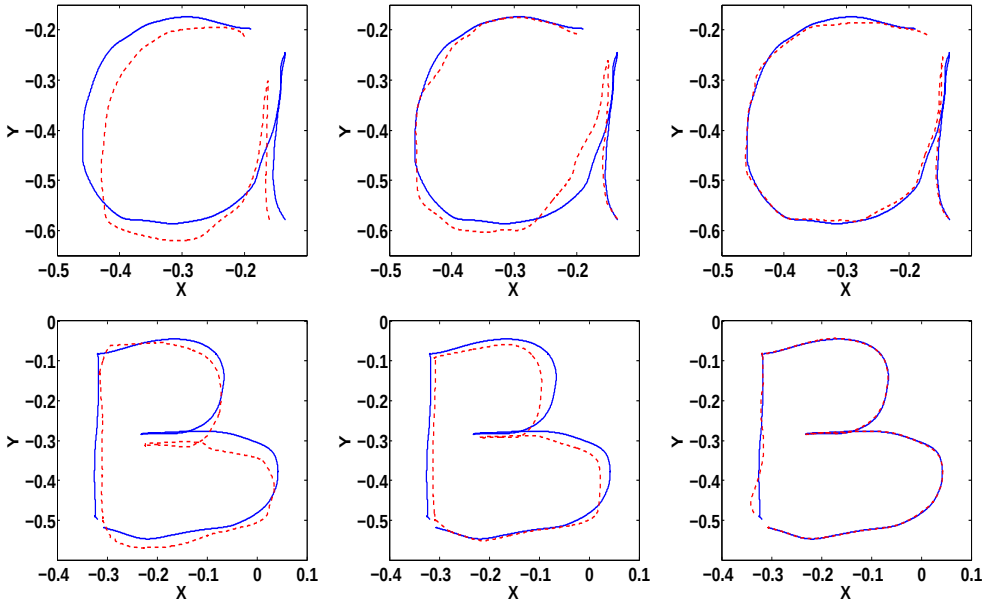
(a) **Tracking errors on Barrett: comparison of offline-learned models**

(b) **Tracking errors on Barrett: full GPR vs. offline and online-learned models with LGP**

Figure 3.4: (a) and **(b)** show the tracking errors (RMSE) on the Barrett WAM. For offline-learned models, LGP is competitive with full GPR and ν-SVR while being better than LWPR and rigid body model. When employing online-updates, LGP can largely improve the tracking results outperforming the offline-learned models using full GPR. The reported results are computed for a test trajectory executed on the robot.

In the following, we create a more complex test case for tracking with inverse dynamics models, i.e., we take the Barrett WAM by the end-effector and guide it along several trajectories which are subsequently used both in learning and control experiments. In order to make these trajectories straightforward to comprehend for humans, we draw all 26 characters of the alphabet in an imaginary plane in task space. An illustration for this data generation process is shown in Figure 3.6 (a). During the imagined writing, the joint trajectories are sampled from the robot. Afterwards, it will attempt to reproduce that trajectory, and the reproductions can be used to generate training data. Subsequently, we used several characters as training examples (e.g., characters from **D** to **O**) and others, e.g., **A** and **B**, as test examples. This setup results in a data set with 10845 samples for training and 1599 for testing.

Similar as in Section 3.4.1, we learn the inverse dynamics models using joint trajectories as input and joint torques as targets. The robot arm is then controlled to perform the joint-space trajectories corresponding to the test characters using the learned models. For LGP, we additionally show that the test characters can be learned online by updating the local models, as described in Section 3.5. The Figure 3.5 shows the tracking results using online learning with LGP in comparison to the offline trained model with standard GPR and a traditional rigid body model. It can be observed that the offline trained models (using standard GPR) can generalize well to unknown characters often having a better tracking performance than the rigid body model. However, the results can be improved even further if the dynamics model is updated online – as done by LGP. The LGP results are shown in Figure

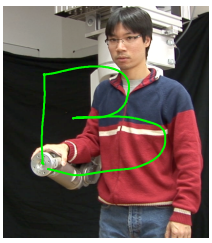


(a) **Tracking test-characters using rigid body model**

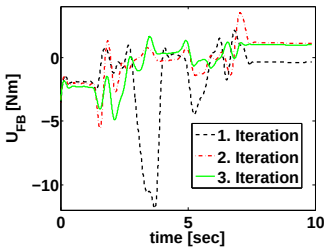
(b) **Tracking test-characters using offline-learned GP model**

(c) **Tracking test-characters after online learning with LGP**

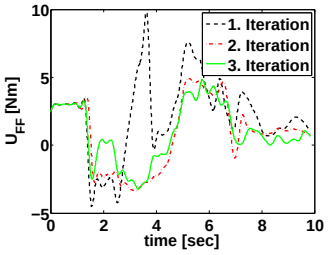
Figure 3.5: Compliant tracking performance on Barrett WAM for two test characters **A** and **B** where the controlled trajectory lies in joint-space while our visualization is in task space for improved comprehensibility. We compare the corresponding rigid body model, an offline trained GP model and an online learning LGP. The thick, blue line denotes the desired trajectory, while the dashed, red line represents the robot trajectory during the compliant tracking task. The results indicate that online learning with LGP outperforms the offline-learned model using full GPR as well as the rigid body dynamics.



(a) **Character acquisition**



(b) **Magnitude of the feedback torque**



(c) **Magnitude of the feedforward torque**

Figure 3.6: This figure illustrates **(a)** Data generation for the learning task. **(b)** and **(c)** show as example the feedback $\mathbf{u}_{\mathrm{FB}}$ and feedforward $\mathbf{u}_{\mathrm{FF}}$ term of the 1. DoF after each iteration of the test character, e.g., **A**. The feedback term degreases gradually, as the model learns to make the required feedforward torques. The feedforward toques $\mathbf{u}_{\mathrm{FF}}$ does not change significantly after 2 iterations. For running through the complete trajectory, e.g., **A**, about 8 seconds are necessary.

3.5 and are achieved after three trials on the test character.

In practice, it is shown that a good tracking performance can already be achieved after 2 iterations of the unknown characters. During the first iteration, the tracking error is large (due to suboptimal prediction in presence of unknown trajectory) resulting in a large correcting feedback term $\mathbf{u}_{\mathrm{FB}}$, see Figure 3.6 (b). In the following iterations, the feedback term gradually degreases, as the model 'learns' to make a correct prediction, i.e, the optimal feedforward torque $\mathbf{u}_{\mathrm{FF}}$ required for the given characters, by using the online sampled input-torques $\mathbf{u}$ as learning target. The feedforward torque $\mathbf{u}_{\mathrm{FF}}$ converges already after 2 or 3 iterations, as shown in Figure 3.6 (c), i.e., after that the feedforward torques do not change significantly.

3.6 Conclusion

The local Gaussian process regression LGP combines the strength of fast computation as in local regression with the potentially more accurate kernel regression methods. As a result, we obtain a real-time capable regression method which works well in robot application. When compared to locally linear methods such as LWPR, the LGP achieves higher learning accuracy while having less computational cost compared to state of the art kernel regression methods such as GPR and ν-SVR. The reduced complexity allows the application of the LGP for online model learning which is necessary for real-time adaptation of model errors or changes in the system. Model based tracking control using online learned LGP models achieves a superior control performance for low gain control in comparison to rigid body models, as well as to offline learned models.

Future research will focus on several important extensions such as finding kernels which are most appropriate for clustering and prediction, and how the choice of a similarity can affect the LGP performance. Partitioning in higher dimension space is still a challenging problem, a possible solution is to perform dimensionality reduction during the partitioning step. It is also interesting to investigate how to infer an optimal value for w_{gen} from data. Furthermore, alternative criteria for insertion and deletion of data points need to be examined more closely. This operation is crucial for online learning as not every new data point is informative for the current prediction task, and on the other hand deleting an old but informative data point may degrade the performance. It also interesting to investigate further applications of the LGP in humanoid robotics with 35 of more DoFs and learning other types of the control such as operational space control.

4 Incremental Online Sparsification for Model Learning

In this chapter, we present a framework for online, incremental sparsification with a fixed budget designed for fast real-time model learning. The proposed approach employs a sparsification method based on an independence measure. In combination with an incremental learning approach such as incremental Gaussian process regression, we obtain a model approximation method which is applicable in real-time online learning. It exhibits competitive learning accuracy when compared with standard regression techniques. Implementation on a real Barrett WAM robot demonstrates the applicability of the approach in real-time online model learning for real world systems.

4.1 Introduction

In recent years, model learning has become an important tool in a variety of robotics applications such as terrain modeling (Lang *et al.*, 2007), sensor evaluation (Plagemann *et al.*, 2007), model-based control (Nakanishi and Schaal, 2004; Schaal *et al.*, 2002) and many others. The reason for this rising interest is that accurate analytical models are often hard to obtain due to the increasing complexity of modern robot systems. Nevertheless, the excessive computational complexity of the more advanced regression techniques still hinders their widespread application in robotics. Models that have been learned offline can only approximate the model correctly in the area of the state space that is covered by the sampled data, and often do not generalize beyond that region. Thus, in order to cope with unknown state space regions online model learning is essential. Furthermore, it also allows the adaptation of the model to changes in the robot dynamics, for example, due to unforeseen loads or time-variant nonlinearities such as backlash, complex friction and stiction.

A few approaches for real-time model learning for robotics have been introduced in the machine learning literature, such as locally weighted projection regression (Vijayakumar *et al.*, 2005) or local Gaussian process regression (Nguyen-Tuong *et al.*, 2008b; Nguyen-Tuong and Peters, 2009). In these methods, the state space is partitioned in local regions for which local models are approximated. As the proper allocation of relevant areas of the state space is essential, appropriate online clustering becomes a central problem for these approaches. For high dimensional data, partitioning of the state space is well-known to be a difficult issue (Vijayakumar *et al.*, 2005; Nguyen-Tuong *et al.*, 2008b). To circumvent this online clustering, an alternative is to find a sparse representation of the *known* data (Rasmussen and

Williams, 2006; Schölkopf and Smola, 2002; Liu *et al.*, 2009). For robot applications, however, it requires finding an incremental sparsification method applicable in real-time online learning – a major challenge tackled in this chapter.

Inspired by the work in (Schölkopf and Smola, 2002; Engel *et al.*, 2004), we propose a method for incremental, online sparsification which can be integrated into several existing online regression methods, making them applicable for model learning in real-time. The suggested sparsification is performed using a test of linear independence to select a sparse subset of the training data points, often called the *dictionary*. The resulting framework allows us to derive criteria for incremental insertion and deletion of dictionary data points, which are two essential operations in such an online learning scenario. For evaluation, we combine our sparsification framework with an incremental approach for Gaussian process regression (GPR) as described in Nguyen-Tuong and Peters (2009). The resulting algorithm is applied in online learning of the inverse dynamics model for robot control (Spong *et al.*, 2006; Nguyen-Tuong *et al.*, 2008a).

The rest of the chapter will be organized as follows: first, we present our sparsification approach which enables real-time online model learning. In Section 4.3, the efficiency of the proposed approach in combination with an incremental GPR update is demonstrated by an offline comparison of learning inverse dynamics models with well-established regression methods, i.e., ν-support vector regression (Schölkopf *et al.*, 2000), standard Gaussian process regression (Rasmussen and Williams, 2006), locally weighted projection regression (Vijayakumar *et al.*, 2005) and local Gaussian process regression (Nguyen-Tuong *et al.*, 2008b). Finally, the capability of incremental GPR using online sparsification for real-time model learning will be illustrated by online approximation of inverse dynamics models for real-time tracking control on a Barrett WAM. A conclusion will be given in Section 4.4.

4.2 Online Sparsification for Real-time Model Learning

In this section, we introduce a sparsification method which – in combination with an incremental kernel regression – enables fast, real-time model learning. The proposed sparsification approach is formulated within the framework of kernel methods. Therefore, we first present the basic intuition behind the kernel methods and motivate the need of online sparsification. Subsequently, we describe the proposed sparsification method in details.

4.2.1 Model Learning with Kernel Methods

By learning a model, we want to approximate a mapping from the input set $\mathbf{X}$ to the target set $\mathbf{Y}$. Given n training data points $\{\mathbf{x}_i, y_i\}_{i=1}^n$, we intend to discover the latent function $f(\mathbf{x}_i)$ which transforms the input vector $\mathbf{x}_i$ into a target value y_i given by the model $y_i = f(\mathbf{x}_i) + \epsilon_i$, where ϵ_i represents a noise term. In general, it is assumed that $f(\mathbf{x})$ can be parametrized as $f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^T \mathbf{w}$, where $\boldsymbol{\phi}$ is a feature vector mapping the input $\mathbf{x}$ into some high dimensional space and $\mathbf{w}$ is the corresponding

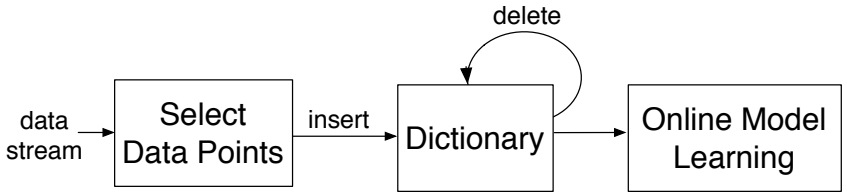


Figure 4.1: Sparsification for online model learning.

weight vector (Schölkopf and Smola, 2002; Rasmussen and Williams, 2006). The weight $\mathbf{w}$ can be represented as a linear combination of the input vectors in the feature space, i.e., $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(\mathbf{x}_i)$ with α_i denoting the linear coefficients. Using these results, the prediction $\hat{y}$ of a query point $\mathbf{x}$ can be given as

$$\begin{aligned} \hat{y} = \hat{f}(\mathbf{x}) = & \sum_{i=1}^{n} \alpha_i \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}) \rangle \,, \\ = & \sum_{i=1}^{n} \alpha_i \mathrm{k}(\mathbf{x}_i, \mathbf{x}) \,. \end{aligned} \tag{4.1}$$

As indicated by Equation (4.1), the inner product of features vectors $\langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}) \rangle$ can be represented as a kernel value $\mathrm{k}(\mathbf{x}_i, \mathbf{x})$ (Schölkopf and Smola, 2002). Thus, instead of finding a feature vector, only appropriate kernels need to be determined. An often used kernel is, for example, the Gaussian kernel

$$\mathrm{k}(\mathbf{x}_p, \mathbf{x}_q) = \exp\left(-\frac{1}{2} (\mathbf{x}_p - \mathbf{x}_q)^T \mathbf{W} (\mathbf{x}_p - \mathbf{x}_q) \right) \,, \tag{4.2}$$

where $\mathbf{W}$ denotes the kernel widths (Schölkopf and Smola, 2002; Rasmussen and Williams, 2006). For employing kernel methods in model learning, however, one needs to estimate the linear coefficients α_i using training examples. State-of-the-art kernel methods such as kernel ridge regression, support vector regression (SVR) or Gaussian process regression (GPR), differ in the approaches for estimating α_i (Schölkopf and Smola, 2002; Rasmussen and Williams, 2006; Hastie *et al.*, 2001). While support vector regression estimates the linear coefficients by optimization using training data (Schölkopf and Smola, 2002), kernel ridge regression and Gaussian process regression basically solve the problem by matrix inversion (Hastie *et al.*, 2001; Rasmussen and Williams, 2006). The complexity of model learning with kernel methods, i.e., the estimation of α_i, depends largely on the number of training examples. In GPR, for example, the computational complexity is $\mathcal{O}(n^3)$, if the model is obtained in batch learning.

However, online model learning requires incremental updates, e.g., incremental estimation of α_i, as the data arrives sequentially. There have been many attempts to develop incremental, online algorithms for kernel methods, such as incremental SVM (Cauwenberghs and Poggio, 2000), sequential SVR (Vijayakumar and Wu, 1999), recursive kernel learning with NARX form (Liu *et al.*, 2009) or the kernel recursive least-squares algorithm (Engel *et al.*, 2004) (for an overview see (Schölkopf and Smola, 2002)). However, most incremental kernel methods do not scale to online

learning in real-time, e.g., for online learning with model updates at 50 Hz or faster. The main reason is that they are neither sparse (Cauwenberghs and Poggio, 2000; Vijayakumar and Wu, 1999), as they use the complete data set for model training, nor do they restrict the size of the sparse set (Engel *et al.*, 2004). To overcome these shortcomings, we propose the setup illustrated in Figure 4.1.

To ensure real-time constraints, we train the model using a dictionary with a fixed budget. The budget of the dictionary, i.e., the sparse set, needs to be determined from the intended learning speed and available computational power. To efficiently make use of the stream of continuously arriving data, we select only the most informative data points for the dictionary. If the budget limit is reached, dictionary points will need to be deleted. Finally, for the model training using dictionary data, most incremental kernel regression methods can be employed, e.g., incremental GPR as described in (Nguyen-Tuong and Peters, 2009), sequential SVR (Vijayakumar and Wu, 1999) or incremental SVM (Cauwenberghs and Poggio, 2000).

Inspired by the work in (Engel *et al.*, 2004; Schölkopf *et al.*, 1999), we use a linear independence measure to select the most informative points given the current dictionary. Based on this measure, we derive criteria to remove data points from the dictionary, if a given limit is reached. The following sections describe the proposed approach in detail.

4.2.2 Sparsification using Linear Independence Test

The main idea in our sparsification approach is that we intend to cover the relevant state space at the best, given a limited number of dictionary points. At any point in time, our algorithm maintains a dictionary $\mathcal{D} = \{\boldsymbol{d}_i\}_{i=1}^{m}$ where m denotes the current number of dictionary points $\boldsymbol{d}_i$. The choice of the dictionary element $\boldsymbol{d}_i$ might be crucial for particular application and will be discussed in Section 4.2.5. To test whether a new point $\boldsymbol{d}_{m+1}$ should be inserted into the dictionary, we need to ensure that it can not be approximated in the feature space spanned by the current dictionary set. This test can be performed using a measure δ defined as

$$\delta = \left\| \sum_{i=1}^{m} \mathrm{a}_i \boldsymbol{\phi}(\boldsymbol{d}_i) - \boldsymbol{\phi}(\boldsymbol{d}_{m+1}) \right\|^2 , \tag{4.3}$$

(see, e.g., (Schölkopf *et al.*, 1999; Schölkopf and Smola, 2002) for background information), where a_i denote the coefficients of linear dependence. Equation (4.3) can be understood as a distance of the new point $\boldsymbol{d}_{m+1}$ to the linear plane spanned by the dictionary set $\mathcal{D}$ in the feature space as illustrated in Figure 4.2. Thus, the value δ can be considered as an independence measure indicating how well a new data point $\boldsymbol{d}_{m+1}$ can be approximated in the feature space of a given data set. Thus, the larger the value of δ is, the more *independent* is $\boldsymbol{d}_{m+1}$ from the dictionary set $\mathcal{D}$, and the more *informative* is $\boldsymbol{d}_{m+1}$ for the learning procedure.

The coefficients a_i from Equation (4.3) can be determined by minimizing δ. For-

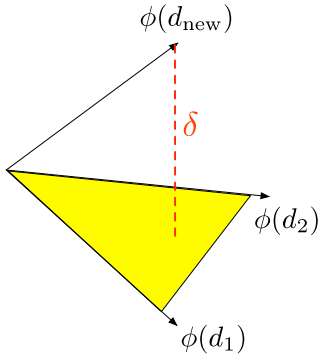


Figure 4.2: Geometrical interpretation of the independence measure δ. Here, the dictionary consists of two data points $\{\boldsymbol{d}_1, \boldsymbol{d}_2\}$ which span a linear plane in the feature space. The independence measure δ for a new data point $\boldsymbol{d}_{new}$ can be interpreted as the distance to this plane.

mulated in matrix form, the minimization of δ can be given as

$$\boldsymbol{a} = \min_{\mathbf{a}} \left[\mathbf{a}^T \boldsymbol{K} \mathbf{a} - 2\mathbf{a}^T \boldsymbol{k} + k \right] , \tag{4.4}$$

where $\boldsymbol{K} = \mathrm{k}(\mathcal{D}, \mathcal{D})$ represents the kernel matrix, $\boldsymbol{k} = \mathrm{k}(\mathcal{D}, \boldsymbol{d}_{m+1})$ is the kernel vector and $k = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1})$ denotes a kernel value. Note that in Equation (4.4) we make use of the property that inner products of feature vectors can be represented as kernel values. Minimizing Equation (4.4) yields the optimal coefficient vector $\boldsymbol{a} = \boldsymbol{K}^{-1}\boldsymbol{k}$. The parameter $\boldsymbol{a}$ can be further regularized taking in account problems such as outliers. The regularization can be controlled by a regularization-parameter which can be chosen to alleviate the outlier's contribution to the sparsification process. Usually, this parameter can be obtained from training data by cross-validation. However, in this work we omitted this regularization as we did not observe any serious problems due to outliers. After substituting the optimal value $\boldsymbol{a}$ into Equation (4.3), δ becomes

$$\delta = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1}) - \boldsymbol{k}^T \boldsymbol{a} \, . \tag{4.5}$$

Using δ we can decide whether to insert new data points into the dictionary. The sparsification procedure is summarized in Algorithm 3.

The independence measure δ as given in Equation (4.5), can be used as a criterion for selecting new data points by thresholding, see Algorithm 3. The threshold parameter η implicitly controls the level of sparsity. However, for a given threshold value η, the number of dictionary points selected from the online data stream is *not* known beforehand and, thus, can be very large. Large dictionaries are prohibitively expensive in terms of computational complexity and as a result not real-time capable. To cope with this problem, we need to define an upper bound on the dictionary size and delete old dictionary points if this limit is reached.

For deleting old dictionary points, we consider the independence measures δ^i of every dictionary point i as illustrated in Figure 4.3. The value δ^i indicates, how well

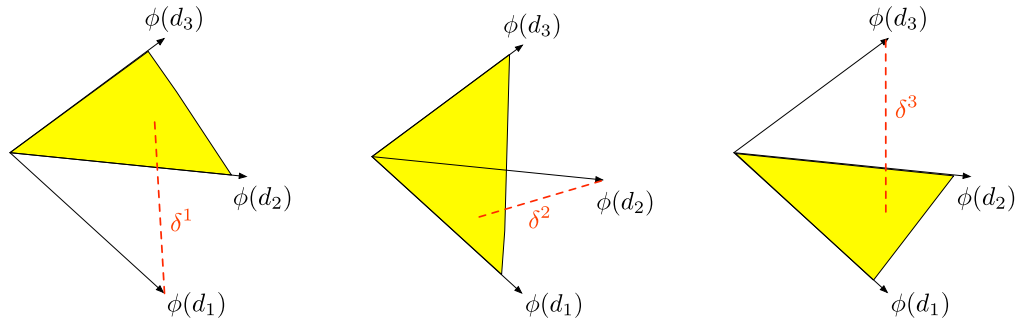


Figure 4.3: Geometrical interpretation of individual independence measures $\delta^i, i = 1..3$, for the dictionary $\mathcal{D}=\{\boldsymbol{d}_1, \boldsymbol{d}_2, \boldsymbol{d}_3\}$. The dictionary point with a *minimal* δ^i is more likely to be deleted. After every dictionary operation (e.g., insertion and deletion of dictionary points), the individual independence measures δ^i have to be incrementally updated.

Algorithm 3 Independence test with online updates of the dictionary.

Input: new point $\boldsymbol{d}_{m+1}$, threshold η, $\mathrm{N}_{\max}$.
Compute: $\boldsymbol{a}=\boldsymbol{K}^{-1}\boldsymbol{k}$ with $\boldsymbol{k}=\mathrm{k}(\mathcal{D}, \boldsymbol{d}_{m+1})$,
Compute: $\delta = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1}) - \boldsymbol{k}^T\boldsymbol{a}$,
if $\delta > \eta$ **then**
  if number of dictionary points $< \mathrm{N}_{\max}$ **then**
    Insert $\boldsymbol{d}_{m+1}$ into $\mathcal{D}$ and update the dictionary using Algorithm 4.
  else
    Insert $\boldsymbol{d}_{m+1}$ into $\mathcal{D}$ and update the dictionary using Algorithm 5, while replacing an old dictionary point from $\mathcal{D}$.
  end if
  Incremental online model training using the new dictionary $\mathcal{D}=\mathcal{D}\cup\boldsymbol{d}_{m+1}$.
end if

the dictionary point i can be approximated by the remainder of the dictionary. The score δ^i has to be updated, when a new data point is inserted into the dictionary or an old data point is removed from the dictionary. In Sections 4.2.3 and 4.2.4, we will discuss an efficient, incremental way of updating the value δ^i applicable for real-time online computation.

4.2.3 Dictionary Update for Inserting New Points

For inserting a new data point $\boldsymbol{d}_{m+1}$ into the dictionary, we have to incrementally update the independence measure δ^i and corresponding coefficients for *every* dictionary point i, as changing the dictionary implies a change for δ^i. This update for an existing dictionary point $\boldsymbol{d}_i$ is achieved by adjusting the corresponding coefficient vector $\boldsymbol{a}^i$. Updating $\boldsymbol{a}^i$ implies an update of $(\boldsymbol{K}^i)^{-1}$ and $\boldsymbol{k}^i$. Here, inserting a new

Algorithm 4 Update the dictionary after inserting a new data point.

Input: new dictionary point $\boldsymbol{d}_{m+1}$.
Update dictionary $\mathcal{D} = \{\boldsymbol{d}_i\}_{i=1}^{m+1}$.
for $i=1$ **to** m **do**
  Compute
    $\boldsymbol{k}_{m+1} = \mathrm{k}(\mathcal{D}^i, \boldsymbol{d}_{m+1})$,
    $k_{m+1} = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1})$,
    $k_{i,m+1} = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_i)$.
  Compute
    $\boldsymbol{\alpha}_i = \left(\boldsymbol{K}^i_{\mathrm{old}}\right)^{-1} \boldsymbol{k}_{m+1}$,
    $\gamma_i = k_{m+1} - \boldsymbol{\alpha}_i^T \boldsymbol{k}_{m+1}$.
  Update δ^i as given in Equation (4.8).
  Update $(\boldsymbol{K}^i_{new})^{-1}$ as given in Equation (4.7).
end for

point will extend $\boldsymbol{K}^i$ by a row/column and $\boldsymbol{k}^i$ by a value, respectively, such that

$$\begin{aligned} \boldsymbol{K}^i_{\mathrm{new}} &= \begin{bmatrix} \boldsymbol{K}^i_{\mathrm{old}} & \boldsymbol{k}_{m+1} \\ \boldsymbol{k}^T_{m+1} & k_{m+1} \end{bmatrix}, \\ \boldsymbol{k}^i_{\mathrm{new}} &= \begin{bmatrix} \boldsymbol{k}^i_{\mathrm{old}} & k_{i,m+1} \end{bmatrix}^T, \end{aligned} \tag{4.6}$$

where $k_{m+1} = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1})$, $k_{i,m+1} = \mathrm{k}(\boldsymbol{d}_i, \boldsymbol{d}_{m+1})$, $\boldsymbol{k}_{m+1} = \mathrm{k}(\mathcal{D}^i, \boldsymbol{d}_{m+1})$ with $\mathcal{D}^i = \mathcal{D} \backslash \{\boldsymbol{d}_i\}$. Using the Equation (4.6), the incremental update of the inverse matrix $(\boldsymbol{K}^i_{\mathrm{new}})^{-1}$ is given by

$$\left(\boldsymbol{K}^i_{\mathrm{new}}\right)^{-1} = \frac{1}{\gamma_i} \begin{bmatrix} \gamma_i (\boldsymbol{K}^i_{\mathrm{old}})^{-1} + \boldsymbol{\alpha}_i \boldsymbol{\alpha}_i^T & -\boldsymbol{\alpha}_i \\ -\boldsymbol{\alpha}_i^T & 1 \end{bmatrix}. \tag{4.7}$$

This result leads to the update rule for the linear independency value δ^i for the i-th dictionary point given by

$$\begin{aligned} \delta^i &= \mathrm{k}(\boldsymbol{d}_i, \boldsymbol{d}_i) - \boldsymbol{k}^{iT}_{\mathrm{new}} \boldsymbol{a}^i_{\mathrm{new}}, \quad \text{with} \\ \boldsymbol{a}^i_{new} &= \frac{1}{\gamma_i} \begin{bmatrix} \gamma_i \boldsymbol{a}^i_{\mathrm{old}} + \boldsymbol{\alpha}_i \boldsymbol{\alpha}_i^T \boldsymbol{k}^i_{\mathrm{old}} - k_{i,m+1} \boldsymbol{\alpha}_i \\ -\boldsymbol{\alpha}_i^T \boldsymbol{k}^i_{\mathrm{old}} + k_{i,m+1} \end{bmatrix}. \end{aligned} \tag{4.8}$$

The variables γ_i and $\boldsymbol{\alpha}_i$ are determined by $\boldsymbol{\alpha}_i = (\boldsymbol{K}^i_{\mathrm{old}})^{-1} \boldsymbol{k}_{\mathrm{m+1}}$ and $\gamma_i = k_{m+1} - \boldsymbol{k}^T_{m+1} \boldsymbol{\alpha}_i$. The procedure for the dictionary update after insertion of new data points is summarized in Algorithm 4.

4.2.4 Dictionary Update for Replacing Points

As the data arrives continuously in an online setting, it is necessary to limit the number of dictionary points so that the computational power of the system is not

exceeded and the real-time constraints are not violated. The individual independence value δ^i – as illustrated in Figure 4.3 – gives rise to a straightforward deletion rule: the *smaller* the independence value δ^i is, the more likely the corresponding dictionary point is to be deleted. The idea is to delete points that are more *dependent* on other dictionary points, i.e., where the corresponding independence value δ^i is small. For the deletion of dictionary points, we additionally consider a temporal allocation of each dictionary point by imposing a time-dependent forgetting rate $\lambda^i \in [0,1]$. Thus, we take the independency value δ^i weighted by the forgetting value λ^i as a deleting score. The role of λ^i will be discussed in detail in Section 4.2.5.

Insertion and additional deletion of dictionary points also change the independence values of other dictionary points which have to be updated subsequently. Insertion of a new point with an additional deletion of the j-th dictionary point implies a manipulation of the j-th row/column of $\boldsymbol{K}^i$ and the j-th value of $\boldsymbol{k}^i$ given by

$$\boldsymbol{K}^i_{new} = \begin{bmatrix} \boldsymbol{K}^i_{\text{old}(1:j)} & \boldsymbol{k}_{m+1(1:j)} & \boldsymbol{K}^i_{\text{old}(j:m)} \\ \boldsymbol{k}^T_{m+1(1:j)} & k_{m+1} & \boldsymbol{k}^T_{m+1(j:m)} \\ \boldsymbol{K}^{iT}_{\text{old}(j:m)} & \boldsymbol{k}_{m+1(j:m)} & \boldsymbol{K}^i_{\text{old}(j:m)} \end{bmatrix}, \qquad \boldsymbol{k}^i_{\text{new}} = \begin{bmatrix} \boldsymbol{k}^i_{\text{old}(1:j)} & k_{i,m+1} & \boldsymbol{k}^i_{\text{old}(j:m)} \end{bmatrix}^T. \tag{4.9}$$

The values $\boldsymbol{k}_{m+1}$, k_{m+1} and $k_{i,m+1}$ are determined as shown in Section 4.2.3. The incremental update of the independence measure δ^i for every i-th dictionary point can be performed directly using an incremental matrix inverse update. Hence,

$$\delta^i = \mathrm{k}(\boldsymbol{d}_i, \boldsymbol{d}_i) - \boldsymbol{k}^{iT}_{\text{new}} \boldsymbol{a}^i_{\text{new}} \quad \text{and} \quad \boldsymbol{a}^i_{\text{new}} = \boldsymbol{A}\, \boldsymbol{k}^i_{\text{new}}, \tag{4.10}$$

where $\boldsymbol{A}$ is computed by the update rule

$$\boldsymbol{A} = \boldsymbol{A}^* - \frac{\text{row}_j[\boldsymbol{A}^*]^T \boldsymbol{r}^T \boldsymbol{A}^*}{1 + \boldsymbol{r}^T \text{row}_j[\boldsymbol{A}^*]^T} \quad \text{with}$$

$$\boldsymbol{A}^* = (\boldsymbol{K}^i_{\text{old}})^{-1} - \frac{(\boldsymbol{K}^i_{\text{old}})^{-1} \boldsymbol{r}\, \text{row}_j[(\boldsymbol{K}^i_{\text{old}})^{-1}]}{1 + \boldsymbol{r}^T \text{row}_j[(\boldsymbol{K}^i_{\text{old}})^{-1}]^T}.$$

Here, the vector $\boldsymbol{r}$ is determined by $\boldsymbol{r} = \boldsymbol{k}_{m+1} - \text{row}_j[\boldsymbol{K}^i_{\text{old}}]^T$ and $\text{row}_j[\boldsymbol{M}]$ denotes the j-th row of a given matrix $\boldsymbol{M}$. The update of $(\boldsymbol{K}^i_{\text{new}})^{-1}$ can be given as $(\boldsymbol{K}^i_{\text{new}})^{-1} = \boldsymbol{A}$. The complete procedure is summarized in Algorithm 5.

4.2.5 Characterization of the Dictionary Space and Temporal Allocation

In previous sections, we described the procedures for incremental insertion and deletion of dictionary points appropriate for real-time computation. The basic idea is that we attempt to approximate the dictionary space (characterized by the dictionary vector $\boldsymbol{d}$) at best using a limited sparse data set $\mathcal{D}$. However, the choice of the dictionary space is a crucial step which may depend on particular applications. As

Algorithm 5 Replace the least important data point in the dictionary by a new one.

Input: new dictionary point $\boldsymbol{d}_{m+1}$.
Compute λ^i as given in Equation (4.12) and, subsequently, find the dictionary point with minimal δ^i weighted by the forgetting rate λ^i:
$j = \min_i(\lambda^i \delta^i)$.
Update dictionary $\mathcal{D}$ by overwriting point j:
$\boldsymbol{d}_j = \boldsymbol{d}_{m+1}$.
for $i = 1$ **to** m **do**
Compute
$\boldsymbol{k}_{m+1} = \mathrm{k}(\mathcal{D}^i, \boldsymbol{d}_{m+1})$,
$k_{m+1} = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1})$,
$k_{i,m+1} = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_i)$.
Compute
$\boldsymbol{r} = \boldsymbol{k}_{m+1} - \mathrm{row}_j[\boldsymbol{K}^i_{\mathrm{old}}]^T$.
Update δ^i as given in Equation (4.10).
Update $(\boldsymbol{K}^i_{\mathrm{new}})^{-1} = \boldsymbol{A}$ in Equation (4.10).
end for

we are dealing with regression here, it is reasonable to choose the dictionary vector as

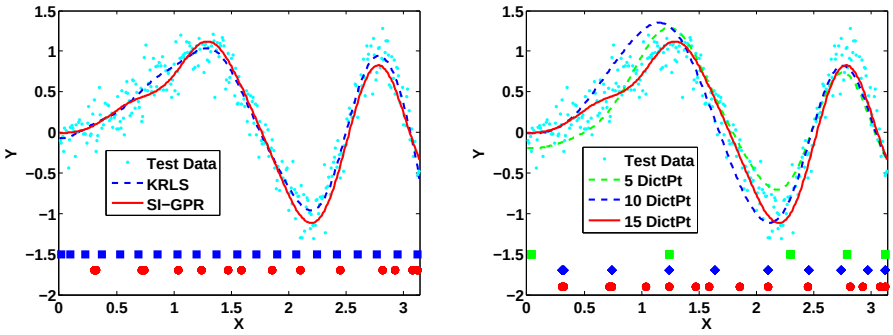
$$\boldsymbol{d} = \begin{bmatrix} \boldsymbol{x} & \boldsymbol{y} \end{bmatrix}^T , \tag{4.11}$$

where $\boldsymbol{x}$ is the input vector and $\boldsymbol{y}$ represents the target values. In so doing, our dictionary space will include the input as well as target distributions. In practice, it shows that input and target distributions both incorporate relevant information for model approximation by regression. For other applications such as learning classification models, considering the input space only might be sufficient.

In addition to the spatial allocation of the dictionary space as achieved by employing the independence measure δ, temporal allocation can be taken into account by introducing a time-variant forgetting factor for every dictionary point i. Here, we consider the forgetting rate

$$\lambda^i(t) = \exp\left(-\frac{(t - t_i)^2}{2h}\right) , \tag{4.12}$$

where t_i is the time when the dictionary point i is included into the sparse set and h represents the intensity of the forgetting rate. For deleting a point from the dictionary, the deletion score is the independence value weighted by the corresponding forgetting value $\lambda^i \in [0, 1]$, as shown in Algorithm 5. By imposing a temporal weight on the independence values, we make sure that temporal information encoded in the data is sufficiently taken in account for the model learning. The forgetting score λ (controlled by the parameter h) represents a trade-off between temporal and spatial



(a) **Comparison with KRLS**

(b) **SI-GPR prediction with different budgets**

Figure 4.4: An illustrative example of the prediction after a single sweep through a toy data set. The dots show the positions of the corresponding dictionary points in the input space. **(a)** The performance of SI-GPR is quite similar to KRLS while the selection of the sparse data sets poses the main difference. SI-GPR selects 15 dictionary points and KRLS 21 data points, respectively. **(b)** Using a fixed budget, e.g., 5, 10 or 15 dictionary points, the most relevant regions in the input space are covered with data points.

covering. If the h value is very small, the temporal effects will gain more importance and the sparsification will behave similar to a time window in common online algorithms.

4.2.6 Comparison to Previous Work

Our work was inspired by the work in (Engel *et al.*, 2004) where the authors also use a linear independence measure to select dictionary points. However, they do not remove dictionary points again but instead show that the dictionary size is bounded for a given threshold η if the data space is assumed to be compact. In practice, for a given threshold value the actual dictionary size is data dependent and unknown beforehand, thus, the dictionary can be very large. In order to cope with the computational constraints in real-time applications, the dictionary size has to be limited. In contrast to the algorithm in (Engel *et al.*, 2004), our approach allows us to formulate an efficient insertion and deletion procedure for a given budget. In (Engel *et al.*, 2004), the spatial allocation is performed in the input space only, resulting in an uniform covering of the complete state space which might be suboptimal for model approximation by regression. As we additionally employ a temporal allocation, the resulting online learning algorithm is able to adapt the model to temporal effects which is an important issue for online learning on real systems.

4.2.6.1 An Illustrative Toy-Example

In the following section, we compare the kernel recursive least square (KRLS) (Engel *et al.*, 2004) with our approach. For the model training, we combine the sparsifi-

cation with an incremental GPR learning (Nguyen-Tuong and Peters, 2009), called sparse incremental GPR (SI-GPR). It should be noted that other incremental model learning methods can also be used in combination with our incremental sparsification, such as sequential SVR or incremental SVM (Vijayakumar and Wu, 1999; Cauwenberghs and Poggio, 2000).

As a toy example, we generate a noisy data set where the relation between the target $\mathbf{y}$ and the input $\mathbf{x}$ is given by $y_i = \sin(x_i^2) + \varepsilon_i$. The data set consists of 315 points, where ε_i is white Gaussian noise with standard deviation 0.2 and x_i ranges from 0 to π. Here, the data is incrementally fed to the algorithms and the prediction is computed after a single sweep through the data set. The results are shown in Figure 4.4, where $\eta = 0.3$ and a Gaussian kernel with width 0.3 is being used. In Figure 4.4 (a), it can be seen that the prediction performance of SI-GPR using dictionary is quite similar to KRLS. Here, the selection of the sparse data points presents the main difference. While KRLS uniformly fills up the complete input space (resulting in 21 dictionary points), the sparsification used by SI-GPR selects the most relevant data points as dictionary points, where the limit of the dictionary is set to be 15. Figure 4.4 (b) shows the performance of SI-GPR for different dictionary sizes, where the prediction improves as expected with increasing dictionary size.

4.3 Evaluations

In this section, we evaluate our sparsification approach used in combination with the incremental GPR (SI-SVR) in several different experimental settings with a focus on inverse dynamics modeling for robot tracking control.

First, we give a short review of learning dynamics models for control. Subsequently, we evaluate the algorithm in the context of learning inverse dynamics. The learning accuracy of SI-GPR will be compared with other regression methods, i.e., LWPR (Vijayakumar *et al.*, 2005), GPR (Rasmussen and Williams, 2006), ν-SVR (Schölkopf *et al.*, 2000) and LGP (Nguyen-Tuong *et al.*, 2008b). For this evaluation in inverse dynamics learning, we employ 2 data sets including synthetic data, as well as real robot data generated from the 7 degrees-of-freedom (DoF) Barrett WAM, shown in Figure 3.1. In Section 4.3.3, SI-GPR is applied for real-time online learning of inverse dynamics models for robot computed torque control (state-of-the-art batch regression methods can not be applied for online model learning). Finally, in Section 4.3.4, we demonstrate the capability of the approach in online learning of a dynamics which changes in presence of different loads.

4.3.1 Learning Dynamics Models for Control

Model based tracking control laws (Spong *et al.*, 2006) determine the joint torques $\mathbf{u}$ that are required for the robot to follow a desired trajectory $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$, where $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$ denote the desired joint angles, velocity and acceleration. In feedforward control, the motor command $\mathbf{u}$ consists of two parts: a feedforward term $\mathbf{u}_{\text{FF}}$ to

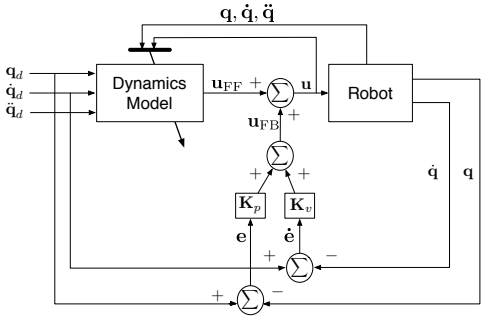


Figure 4.5: Feedforward control with online model learning.

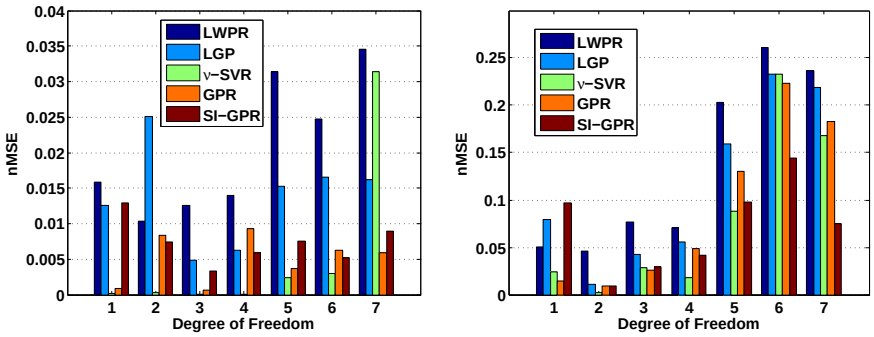
achieve the movement and a feedback term $\mathbf{u}_{\mathrm{FB}}$ to ensure stability of the tracking, as shown in Figure 4.5. The feedback term can be a linear control law such as $\mathbf{u}_{\mathrm{FB}} = \mathbf{K}_p\mathbf{e} + \mathbf{K}_v\dot{\mathbf{e}}$, where $\mathbf{e}$ denotes the tracking error with position gain $\mathbf{K}_p$ and velocity gain $\mathbf{K}_v$. The feedforward term $\mathbf{u}_{\mathrm{FF}}$ is determined using an inverse dynamics model and, traditionally, the analytical rigid-body model is employed (Spong *et al.*, 2006).

If a sufficiently precise inverse dynamics model can be approximated, the resulting control law $\mathbf{u} = \mathbf{u}_{\mathrm{FF}}(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d) + \mathbf{u}_{\mathrm{FB}}$ will accurately drive the robot along the desired trajectory. Due to the high complexity of modern robot systems such as humanoids or service robots, traditional analytical rigid-body model often cannot provide a sufficiently accurate inverse dynamics model. The lack of model precision has to be compensated by increasing the tracking gains $\mathbf{K}_p$ and $\mathbf{K}_v$ making the robot stiff and less safe for the environment (Nguyen-Tuong *et al.*, 2008a). Thus, to fulfill both requirements of compliant control, i.e., having low tracking gains and high tracking accuracy, more precise models are necessary. One possibility to obtain an accurate inverse dynamics model is to learn it *directly* from measured data.

If the dynamics model can be learned online (see Figure 4.5), the robot controller can adapt itself to changes in the robot dynamics, e.g., due to unforeseen load or time-variant disturbances. Online learning of dynamics models using sampled data realized in a setup as in Figure 4.5 can be considered as a self-supervised learning problem.

4.3.2 Offline Comparison in Learning Inverse Dynamics

For the generation of two data sets, i.e., one with simulation data and one with real robot data, we sample joint space trajectories and corresponding torques from an analytical model of the Barrett WAM, as well as from the real robot. This results in two test scenarios, each having 12000 training points and 3000 test points. Given samples $\mathbf{x}=[\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}]$ as input and using the corresponding joint torques $\mathbf{y}=\mathbf{u}$ as targets, we have a regression problem with 21 input dimensions and 7 output dimensions (i.e., a single desired torque for each motor joint). The robot inverse dynamics model is estimated separately for each DoF employing LWPR, ν-SVR,



(a) **Error on simulation data** (b) **Error on real WAM data**

Figure 4.6: (a) Error (nMSE) on simulated data from a robot model for every DoF. **(b)** Error (nMSE) on real robot data for every DoF. As shown by the results, SI-GPR is competitive to standard batch regression methods such as ν-SVR and GPR, local learning methods such as LWPR and LGP. For model learning with SI-GPR, the dictionary size is limited to be 2000 data points.

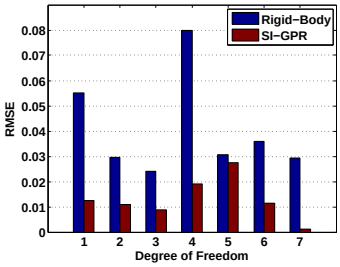
GPR, LGP and SI-GPR.

Employing SI-GPR for learning the inverse dynamics model, we first sparsify the full data sets (i.e., 12000 data points) as described in Section 4.2, and subsequently apply incremental GPR for an offline approximation of the model. For all data sets, the dictionary size is limited to be 2000 points, where the parameter η is set to be 0.1. For the sparsification process and the incremental GPR, we employ the Gaussian kernel, whose hyperparameters are obtained by optimizing the marginal likelihood (Rasmussen and Williams, 2006).

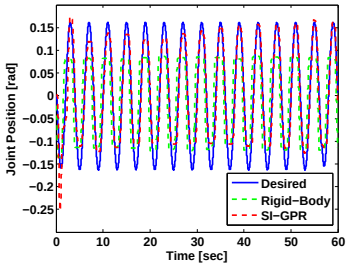
Figures 4.6 (b) and (c) show the offline approximation errors on the test sets evaluated using the normalized mean square error (nMSE) which is defined as the fraction of mean squared error and the variance of target. It can be observed from the results that SI-GPR using sparsification is competitive in learning accuracy despite the smaller amount of training examples (i.e., dictionary points). In practice, it also shows that the models are easier to train using SI-GPR compared to local learning methods such as LWPR and LGP, as the latter require an appropriate clustering of the state space which is not straightforward to perform for many data sets.

4.3.3 Model Online Learning in Computed Torque Control

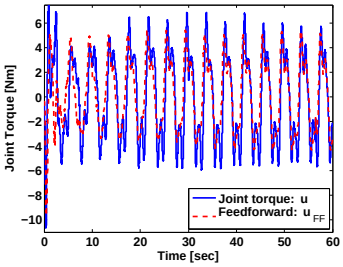
In this section, we apply SI-GPR for the real-time online learning of the inverse dynamics model for torque prediction in robot tracking control, as introduced in Section 4.3.1. The control task with model online learning is performed with 500 Hz sample rate (i.e., we get a new test point every 2 ms) on the real Barrett WAM. In so doing, the trajectory $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ and the corresponding joint torques $\mathbf{u}$ are sampled online as shown in Figure 4.5. The inverse dynamics model, i.e., the mapping $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \mathbf{u}$, is learned in a self-supervised manner during the tracking task using the trajectory



(a) **Tracking error on Barrett WAM in joint space for all 7 DoFs**



(b) **Tracking error on Barrett WAM in joint space for the 1st DoF**



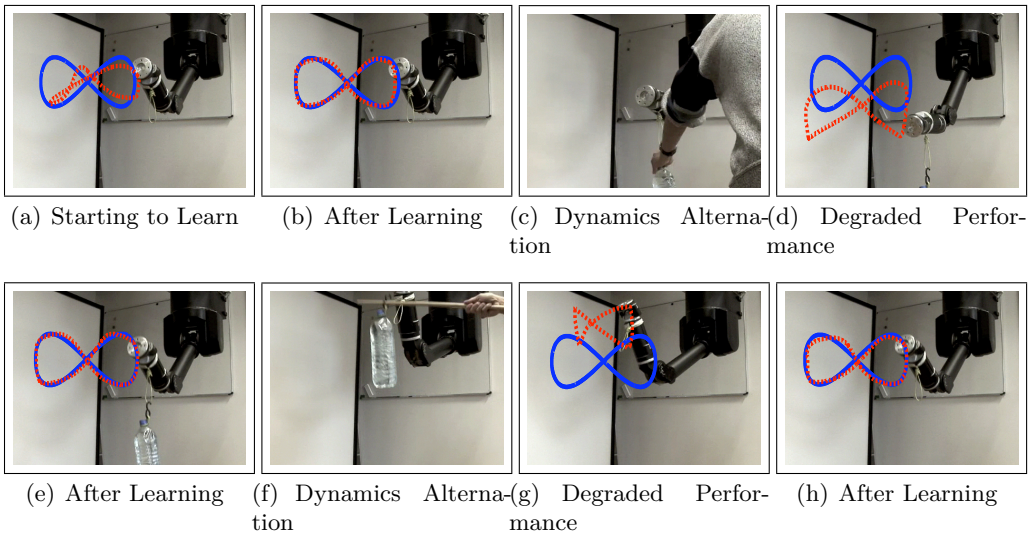
(c) **Feedforward $\mathbf{u}_{\text{FF}}$ and joint torque u of the 1st DoF during SI-GPR online learning**

Figure 4.7: (a) Compliant tracking performance in joint space after 60 sec computed as RMSE. Computed torque control with online learned model (SI-GPR) outperforms common analytical rigid-body model. **(b)** Tracking performance in joint space for the 1st DoF, e.g., the shoulder flexion extension, (other DoFs are similar) during the first 60 sec. **(c)** Predicted feedforward torque $\mathbf{u}_{\text{FF}}$ and joint torque $\mathbf{u}$ of the 1st DoF. The predicted torque $\mathbf{u}_{\text{FF}}$ gradually converges to the joint torque $\mathbf{u}$ as the latter is sampled as target for the model learning.

as input and the joint torque as target, starting with an empty dictionary. As the learned inverse dynamics model is subsequently applied to compute the feedforward torques $\mathbf{u}_{\text{FF}}$, it can be observed that $\mathbf{u}_{\text{FF}}$ gradually converges against $\mathbf{u}$. In this experiment, we set the tracking gains (see Section 4.3.1) to very low values satisfying the requirement of compliant control. In so doing, inaccurate inverse dynamics models, however, will result in large tracking errors.

In this online learning experiment, the maximal dictionary size is set to be 300, $\eta =$ 0.01 and a Gaussian kernel is used whose parameters are determined by optimizing the marginal likelihood. For the forgetting score λ, a forgetting rate $h = 0.4$ is chosen by cross-validation. The desired tracking trajectory is generated such that the robot's end-effector follows a 8-figure in the task space. For comparison, we also apply an analytical rigid-body model for torque prediction (Spong *et al.*, 2006). The tracking results are shown in Figure 4.7, where tracking control task is performed for 60 sec on Barrett WAM. During this time, the dictionary is first incrementally filled up and subsequently updated about 700 times. The inverse dynamics model is incrementally learned online using the dictionary.

Figure 4.7 (a) compares the tracking performance of the online learned model and the analytical rigid-body model in joint space for all 7 DoFs, where the tracking error is computed as root mean square error (RMSE). It can be observed that the tracking accuracy can be significantly improved, if the inverse dynamics model is approximated online. For several DoFs such as the 4th DoF (i.e, the elbow flexion), the rigid-body model fails to described the true dynamics resulting in a large tracking error as shown in Figure 4.7 (a). This example demonstrates the difficulty in



(a) Starting to Learn (b) After Learning (c) Dynamics Alternation (d) Degraded Performance

(e) After Learning (f) Dynamics Alternation (g) Degraded Performance (h) After Learning

Figure 4.8: Tracking experiment with online model learning. The blue, thick lines illustrate the desired trajectory and the red, dotted lines show the robot trajectory in task space. **(a)** As the model is started to be learned online (beginning with an empty dictionary), the robot first shows a transient behavior. **(b)** With a successfully learned model, the robot is able to follow the trajectory well. **(c)** The original dynamics is changed by hanging a heavy water bottle to the arm. **(d)** Due to the modified dynamics, the robot fails to track the desired initial trajectory. Therefore, the robot starts to learn the modified dynamics online. **(e)** As the online model learning converges, the robot gradually moves back to the desired initial position. **(f)** The dynamics is modified again by removing the water bottle. **(f)** The learned model is no more accurate, i.e., the predicted torques are now too large. The modified dynamics is subsequently adapted online. **(g)** As the dynamics model is successfully adapted, the robot returns to the initial desired position. A video showing the experiment can be seen at "http://www.robot-learning.de/".

analytically modeling complex systems which may be alleviated by directly learning the model.

Figures 4.7 (b,c) show the tracking performance in joint space for the 1st DoF, i.e., the shoulder flexion extension, during the first 60 sec, other DoFs are similar. It can be seen that the predicted torque $\mathbf{u}_{\text{FF}}$ (for 1st DoF) consistently converges to the joint torque $\mathbf{u}$ as the latter is sampled as target for the model learning.

4.3.4 Online Learning for Changing Dynamics

In this section, we demonstrate the capability of the algorithm for online model learning and self-adaption to changes in the dynamics in a more complex task. Figures 4.8 (a)-(h) show the progress of the experiment. First, we learn the robot inverse dynamics online starting with an empty dictionary. We apply SI-GPR with the same parameter setting as given in Section 4.3.3. Here, we also first incrementally fill up the dictionary and subsequently update for new data points. Subsequently,

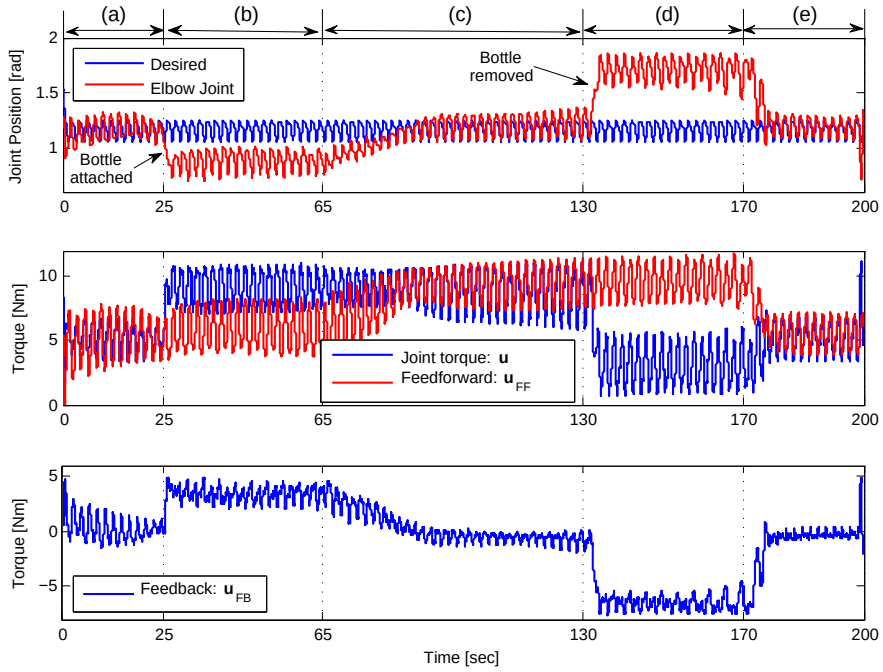


Figure 4.9: Elbow joint angle (4th DoF) and corresponding torques of the Barrett WAM during the experiment. **(a)** The robot starts with an unmodified dynamics. **(b)** As the water bottle is attached to the arm, it causes a jump in the feedback torque $\mathbf{u_{FB}}$ and joint torque $\mathbf{u}$. Due to the compliant tracking mode, the change in the feedback torque $\mathbf{u_{FB}}$ is not sufficient to compensate the resulting tracking error. **(c)** As the dynamics model is learned online, the new dynamics can be incorporated in the prediction of the feedforward torque $\mathbf{u}_{\mathrm{FF}}$. As the online model learning converges, the resulting tracking error is also gradually reduced, i.e., the robot returns to the desired position. Note how the feedback torque $\mathbf{u_{FB}}$ is decreasing, as the model, i.e., the torque prediction, becomes more accurate. **(d)** The online modification of the dynamics, i.e., removing the water bottle, leads to changes in the feedback and joint torques. **(e)** As the adaptation is successfully done and the feedforward torque $\mathbf{u_{FF}}$ converges, the robot moves back to the desired trajectory and the feedback torque decreases again.

we modify the robot dynamics by attaching a heavy water bottle to the arm (beside the changing load, the swinging bottle additionally introduces a time-variant nonlinearity). Due to the change in the dynamics, the robot cannot follow the given trajectory in joint space. As in Section 4.3.3, the desired joint space trajectory is defined such that the robot draw a figure-8 in the task space. The end-effector velocity of the robot during the tracking in task space is about 0.6 m/sec, where the displacement ranges from 0.2 to 0.5 m. The modified dynamics can now be learned online, as shown in the Figures 4.8 (a)-(h).

As the online model learning converges, the modified dynamics is taken into account by the predicted feedforward torques $\mathbf{u}_{\mathrm{FF}}$. As an example, Figure 4.9 shows the joint angle and corresponding torques of the 4th DoF (i.e., the robot's elbow flexion) during the experiment. With the attached water bottle, the predicted feedforward torques $\mathbf{u}_{\mathrm{FF}}$ gradually becomes more precise as the model converges. The decreasing model error results in accurate tracking performance. By continuously updating the dictionary, i.e., by insertion and eventual deletion of dictionary points, the model can adapt to dynamical changes in the environment. This effect can further be demonstrated by removing the water bottle. As observed from Figure 4.9, the predicted torque $\mathbf{u}_{\mathrm{FF}}$ is subsequently reduced, since smaller torques are now required for proper tracking of the desired trajectory.

In this experiment, we employ very low tracking gains. As a result, the model errors will seriously degrade the tracking accuracy unless a more precise inverse dynamics model is learned. Figure 4.9 shows that the feedback torques $\mathbf{u}_{\mathrm{FB}}$ are not able to compensate the model error due to the low tracking gains. As the inverse dynamics model becomes more accurate during the online learning, the feedback torques $\mathbf{u}_{\mathrm{FB}}$ decrease, and, thus, the joint torques $\mathbf{u}$ mainly depend on the feedforward term $\mathbf{u}_{\mathrm{FF}}$. As the torque generation relies more on the inverse dynamics model, we can achieve both compliant control (by using low tracking gains) and high tracking accuracy at the same time.

4.4 Conclusion and Future Work

Motivated by the need of fast online model learning in robotics, we have developed an incremental sparsification framework which can be used in combination with an online learning algorithm enabling an application in real-time online model learning. The proposed approach provides a way to efficiently insert and delete dictionary points taking in account the required fast computation during model online learning in real-time. The implementation and evaluation on a physical Barrett WAM robot emphasizes the applicability in real-time online model learning for real world systems. Our future research will be focused on further extensions such as including a database enabling online learning for large data sets.

5 Special Topics

In this chapter, we explore solutions to two model learning problems. In Section 5.1, we investigate how available prior knowledge from advanced physics-based modeling techniques can be used for model learning. In Section 5.2, we investigate the problem of learning task-space tracking control.

5.1 Using Prior Model Knowledge for Learning Inverse Dynamics

Inserting prior knowledge into the learning process is especially helpful when only sparse and poor data sets are available. In that case, incorporating additional model knowledge may result in faster learning speed, higher accuracy and better generalization. In this section, we present two possible semiparametric regression approaches, where the knowledge of the physical model can either become part of the mean function or of the kernel in a nonparametric Gaussian process regression. These methods are tested on sampled data in the setting of learning inverse dynamics models and, subsequently, used in tracking control on a Barrett WAM.

5.1.1 Introduction

Acquiring accurate models of dynamical systems is an essential step in many technical applications. In robotics, such models are for example required for state estimation (Ko and Fox, 2009) and tracking control (Burdet and Codourey, 1998; Nguyen-Tuong *et al.*, 2008a). It is well-known that the robot dynamics can be modeled by (Spong *et al.*, 2006)

$$\boldsymbol{\tau}\left(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}\right) = \mathbf{M}\left(\mathbf{q}\right)\ddot{\mathbf{q}} + \mathbf{C}\left(\mathbf{q}, \dot{\mathbf{q}}\right) + \mathbf{G}\left(\mathbf{q}\right) + \boldsymbol{\epsilon}\left(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}\right) , \tag{5.1}$$

where $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ are joint angles, velocities and accelerations of the robot, respectively, $\boldsymbol{\tau}$ denotes the joint torques, $\mathbf{M}(\mathbf{q})$ is the generalized inertia matrix of the robot, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ are the Coriolis and centripetal forces and $\mathbf{G}(\mathbf{q})$ is gravity.

As shown in Equation (5.1), the robot dynamics equation consists of a *rigid body dynamics model* (RBD), $\boldsymbol{\tau}_{\mathrm{RBD}} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q})$, and a structured error term $\boldsymbol{\epsilon}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. The model errors are caused by unmodeled dynamics (e.g., hydraulic tubes, actuator dynamics, flexibility and dynamics of the cable drives), ideal-joint assumptions (e.g., no friction), inaccuracies in the RBD model parameters, etc. The RBD model of a manipulator is well-known to be linear in the parameters $\boldsymbol{\beta}$ (Atkeson

et al., 1986; Spong *et al.*, 2006), i.e.,

$$\boldsymbol{\tau}_{\text{RBD}} = \boldsymbol{\Phi}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\boldsymbol{\beta}\,, \tag{5.2}$$

where $\boldsymbol{\Phi}$ is a matrix containing nonlinear functions of joint angles, velocities and accelerations which are often called basis functions. Modeling the robot dynamics using the RBD model in Equation (5.2) requires the identification of the dynamics parameters $\boldsymbol{\beta}$. For our 7 degree-of-freedom (DoF) Barrett WAM, for example, we have to identify 70 dynamics parameters (for each DoF, we have 10 parameters that, in an ideal world, could directly be obtained from the CAD data).

The main advantage of the RBD model is that it provides a "global" and unique relationship between the joint trajectory $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ and the torques $\boldsymbol{\tau}_{\text{RBD}}$. This inverse dynamics model can be computed efficiently and is applicable in real-time. In the past, such parametric models have been employed within *parametric* learning frameworks resulting both in system identification and adaptive control approaches (Burdet and Codourey, 1998; Li, 1990). In adaptive control, for example, the dynamics parameters $\boldsymbol{\beta}$ are continuously adjusted while the model is used for predicting the feedforward torques required for achieving the desired trajectory (Burdet and Codourey, 1998). In practice, estimating the dynamics parameters is not always straightforward. It is hard to create sufficiently rich data sets so that plausible parameters can be identified, and when identified online, additional persistent excitation issues occur. Furthermore, the parameters that optimally fit a data set, are frequently not physically consistent (e.g., violating the parallel axis theorem or having physically impossible values) and, hence, physical consistency constraints have to be imposed on the regression problem (Ting *et al.*, 2009; Nakanishi *et al.*, 2008). Due to the fixed basis functions, parametric models are not capable of capturing the structured nonlinearities of $\boldsymbol{\epsilon}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. Instead, these unmodeled components will bias the estimation of the parameters $\boldsymbol{\beta}$.

Learning methods that are not limited by fixed basis functions will suffer less from many of these problems. Especially, modern *nonparametric* learning techniques, such as Gaussian process regression offer an appealing alternative for model learning as they *infer* the optimal model structure from data (Rasmussen and Williams, 2006). Therefore, nonparametric methods can be used more flexibly and are powerful in capturing higher order nonlinearities resulting in faster model approximation and higher learning accuracy. When learning inverse dynamics, for example, the nonparametric methods will approximate a function describing the relationship $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \boldsymbol{\tau}$ that includes all nonlinearities encoded by the sampled data (Nguyen-Tuong *et al.*, 2008a). Most nonparametric methods attempt to learn the model from scratch and, thus, do not make use of any knowledge available to us from analytical robotics. Nevertheless, many nonparametric learning methods also exhibit several drawbacks. First, very large amounts of data are necessary for obtaining a sufficiently accurate model (e.g., learning an inverse dynamics model requires handling enormous amount of data (Nguyen-Tuong *et al.*, 2008a)). Second, the sampled data has to be sufficiently informative, i.e., the data should contain as much information

about the system dynamics as possible (in the context of adaptive control, this problem is equivalent to the requirement of persistent excitation). Thus, if only small and relatively "poor" data sets are available, nonparametric models will not be able to generalize well for unknown data.

In this work, we want to combine the strengths of both learning approaches, i.e., the parametric and nonparametric model learning, and obtain a *semiparametric* regression framework. We identify two possible approaches to incorporate the parametric model from analytical robotics into the nonparametric Gaussian process model for learning the inverse dynamics. The first approach combines both parametric and nonparametric models directly by approximating the unmodeled robot dynamics using nonparametric models. For so doing, the robot dynamics in Equation (5.1) may be described by the semiparametric model

$$\boldsymbol{\tau}\left(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}\right) = \underbrace{\boldsymbol{\Phi}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\boldsymbol{\beta}}_{\text{parametric}} + \underbrace{\boldsymbol{\epsilon}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})}_{\text{nonparametric}}.$$

Additionally, a prior can be placed on the dynamics parameters $\boldsymbol{\beta}$ of the RBD model. This step further allows taking the uncertainty of the estimated parameters into account. The alternative approach employs a "rigid body dynamics" kernel for the nonparametric regression, where the kernel incorporates the information of the parametric RBD model. As a result, $\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is a nonparametric function with a RBD kernel. In order to absorb the unmodeled dynamics $\boldsymbol{\epsilon}$ which is not described by the RBD kernel, we can combine the RBD kernel with other appropriate kernels.

The remainder of the section will be organized as follows, first, we present in detail how the RBD model can be incorporated into the nonparametric Gaussian process regression. In Section 5.1.3, we give a brief review on inverse dynamics for control and evaluate our semiparametric models in the setting of learning inverse dynamics. Subsequently, we compare the methods in the setting of low-gain robot tracking control, where the estimated feedforward models are applied for online torque prediction (Spong *et al.*, 2006; Nguyen-Tuong *et al.*, 2008a). The compliant tracking task is performed on a Barrett WAM. The section will be summarized in Section 5.2.4.

5.1.2 Semiparametric Regression with Gaussian Process

Using the nonparametric Gaussian process regression (GPR) framework (Rasmussen and Williams, 2006), the robot dynamics can be modelled by $\boldsymbol{\tau}(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), \mathrm{k}(\mathbf{x}, \mathbf{x}'))$, where $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}]^T$ is the input, $m(\mathbf{x})$ is the mean function and $\mathrm{k}(\mathbf{x}, \mathbf{x}')$ the covariance function of the Gaussian process (GP) (see Section 3.2.2 for a quick review or (Rasmussen and Williams, 2006) for many details). Commonly, the GP model is assumed to have zero mean $m(\mathbf{x}) = \mathbf{0}$, i.e., no prior knowledge, and the covariance function $\mathrm{k}(\mathbf{x}, \mathbf{x}')$ is usually a general kernel such as a Gaussian or Matern kernel, which allows reproducing arbitrary functions (Rasmussen and Williams, 2006). A straightforward way to include the RBD model shown in Equation (5.2) is to set

$m(\mathbf{x}) = \boldsymbol{\Phi}(\mathbf{x})\boldsymbol{\beta}$. This approach is equivalent to a semiparametric model

$$\boldsymbol{\tau}(\mathbf{x}) \sim \boldsymbol{\Phi}(\mathbf{x})\,\boldsymbol{\beta} + \mathcal{GP}(\mathbf{0}, \mathrm{k}(\mathbf{x}, \mathbf{x}')) \ . \tag{5.3}$$

The resulting dynamics model in Equation (5.3) consists of a parametric part, i.e., the RBD model, and a nonparametric term given by a zero mean GP. When comparing Equation (5.3) to the robot dynamics in Equation (5.1), it can be observed that the main purpose of the nonparametric term is absorbing the unmodeled dynamics $\boldsymbol{\epsilon}$. For approximating the unmodeled dynamics with a regular GP, any admissible kernels can be used (e.g., a Gaussian kernel).

An alternative to including prior knowledge in the mean function is to define a "rigid body dynamics" kernel $\mathrm{k}_{\mathrm{rbd}}(\mathbf{x}, \mathbf{x}')$ containing the function class that describes the RBD. Thus, the resulting model is now a GP regression with a kernel embedding the parametric RBD model

$$\boldsymbol{\tau}(\mathbf{x}) \sim \mathcal{GP}(\mathbf{0}, \mathrm{k}_{\mathrm{rbd}}(\mathbf{x}, \mathbf{x}')) \ . \tag{5.4}$$

The RBD kernel can be further extended by an additional kernel (e.g., a Gaussian kernel) to absorb the unmodeled dynamics $\boldsymbol{\epsilon}$. In the following sections, we will describe the two introduced semiparametric regression approaches in detail.

5.1.2.1 Gaussian Process Regression with RBD Mean

If the GP mean is not zero but given by a fixed mean function as in Equation (5.3), the GP model is *biased* towards this prior information. In the following, we discuss two approaches to incorporate the RBD mean function into the GP regression. The first approach directly incorporates the RBD as an independently estimated parametric model. The second approach additionally infers the dynamics parameters using a Gaussian prior taking the uncertainty of the estimated parameters into account.

Using Estimated Dynamics Parameters

Inserting the RBD model into the GP as mean function, we have for each DoF k a torque prediction $\bar{\tau}_k$ for a query point $\mathbf{x}_*$

$$\begin{aligned} \bar{\tau}_k(\mathbf{x}_*) &= \boldsymbol{\phi}_k(\mathbf{x}_*)^T\boldsymbol{\beta} + \mathbf{k}_*^T \left(\mathbf{K} + \sigma_n^2\mathbf{I}\right)^{-1} \left(\mathbf{y}_k - \boldsymbol{\Phi}_k(\mathbf{X})\boldsymbol{\beta}\right) \\ &= \boldsymbol{\phi}_k(\mathbf{x}_*)^T\boldsymbol{\beta} + \mathbf{k}_*^T\boldsymbol{\alpha}_k \ , \end{aligned} \tag{5.5}$$

where $\boldsymbol{\phi}_k^T$ is the k-th row of $\boldsymbol{\Phi}$ evaluated on the query point, and $\boldsymbol{\Phi}_k$ is the corresponding matrix evaluated on the training input data $\mathbf{X}$, and $\mathbf{y}_k$ is the sampled torques of joint k. When only pre-estimated parameters $\boldsymbol{\beta}$ are being used, they can be obtained, for example, from data or CAD models of the robot (Ting *et al.*, 2009).

From Equation (5.5), it can be seen that the GP is mainly used to *absorb the errors* between the RBD model and the sampled data. If the RBD model perfectly describes

the robot dynamics, the error $(\mathbf{y}_k - \boldsymbol{\Phi}_k(\mathbf{X})\boldsymbol{\beta})$ will disappear and the prediction will depend only on the RBD term. Equation (5.5) also shows that if the query point $\mathbf{x}_*$ is far away from the training data $\mathbf{X}$, the resulting kernel value $\mathbf{k}_*^T$ will tend to zero (for example, this holds for the Gaussian kernel shown in Section 3.2.2). In that case, the resulting torque prediction $\bar{\tau}_k(\mathbf{x}_*)$ will mainly depend on the RBD term. This property is important, as we can never cover the complete state space using finite (and possibly small) training data sets. If the robot moves to the regions of the state space that are *not* covered by the sampled data (i.e., the learned nonparametric models may not generalize well to these state space regions), the torque prediction will rely on the parametric RBD model.

Placing a Prior on Dynamics Parameters

As the dynamics parameters are estimated offline or even when they are adapted online (Burdet and Codourey, 1998), they may be quite inaccurate due to the difficulties discussed before. Taking the uncertainty of our parameters into account, we can place a Gaussian prior $\mathbf{b}$ on the dynamics parameters given by

$$\mathbf{b} \sim \mathcal{GP}\left(\boldsymbol{\beta}, \mathbf{B}\right) , \tag{5.6}$$

where we take the pre-estimated dynamics parameters $\boldsymbol{\beta}$ as mean and define a diagonal variance matrix $\mathbf{B}$ (Rasmussen and Williams, 2006). The variance expresses our belief about the uncertainty of the estimated dynamics parameters. $\mathbf{B}$ can be estimated using expert knowledge or from measured data. Using the prior given in Equation (5.6), the resulting torque prediction for the DoF k is given by

$$\begin{aligned} \bar{\tau}_k(\mathbf{x}_*) &= \boldsymbol{\phi}_k(\mathbf{x}_*)^T \bar{\boldsymbol{\beta}}_k + \mathbf{k}_*^T \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} \left(\mathbf{y}_k - \boldsymbol{\Phi}_k(\mathbf{X})\bar{\boldsymbol{\beta}}_k\right) \\ &= \boldsymbol{\phi}_k(\mathbf{x}_*)^T \bar{\boldsymbol{\beta}}_k + \mathbf{k}_*^T \bar{\boldsymbol{\alpha}}_k , \end{aligned} \tag{5.7}$$

where $\bar{\boldsymbol{\beta}}_k$ is defined as

$$\bar{\boldsymbol{\beta}}_k = \left(\mathbf{B}^{-1} + \boldsymbol{\Phi}_k^T \tilde{\mathbf{K}}^{-1} \boldsymbol{\Phi}_k\right)^{-1} \left(\boldsymbol{\Phi}_k^T \tilde{\mathbf{K}}^{-1} \mathbf{y}_k + \mathbf{B}^{-1}\boldsymbol{\beta}\right) ,$$

with $\tilde{\mathbf{K}} = (\mathbf{K} + \sigma_n^2 \mathbf{I})$ (see (Rasmussen and Williams, 2006) for more details). Note that due to the probabilistic inference, we have for *each* DoF different values for the dynamics parameters $\bar{\boldsymbol{\beta}}_k$ in this case. However, as it is physically more reasonable to assume that all DoFs share the same dynamics parameters, we can also determine for all DoFs a single $\bar{\boldsymbol{\beta}}$ by summarizing over the DoFs. Thus, we have

$$\bar{\boldsymbol{\beta}} = \left(n\mathbf{B}^{-1} + \textstyle\sum_{k=1}^{n} \boldsymbol{\Phi}_k^T \tilde{\mathbf{K}}^{-1} \boldsymbol{\Phi}_k\right)^{-1} \left(\textstyle\sum_{k=1}^{n} \boldsymbol{\Phi}_k^T \tilde{\mathbf{K}}^{-1} \mathbf{y}_k + n\mathbf{B}^{-1}\boldsymbol{\beta}\right) ,$$

where n denotes the number of DoFs.

5.1.2.2 Gaussian Process Regression with RBD Kernel

An alternative to including prior knowledge in the mean function are appropriate kernels which contain the information given by the basis functions. Incorporating prior knowledge within the kernel framework has been discussed in the past for a few problems (Schölkopf *et al.*, 1997). We adopt these ideas and define a kernel $\mathrm{k}_{\mathrm{rbd}}$ that embeds the function space of our RBD model

$$\mathrm{k}^k_{\mathrm{rbd}}\left(\mathbf{x}_p, \mathbf{x}_q\right) = \boldsymbol{\phi}_k\left(\mathbf{x}_p\right)^T \mathbf{W}_k \boldsymbol{\phi}_k\left(\mathbf{x}_q\right) + \sigma_k^2 \delta_{pq}\,. \tag{5.8}$$

For DoF k, the corresponding kernel is a dot product between its basis function $\boldsymbol{\phi}_k$ scaled by the parameter $\mathbf{W}_k$. A noise term σ_k^2 is additionally added to the diagonal values. Using this RBD kernel, the hyperparameters will be given by $\boldsymbol{\theta}^k_{\mathrm{rbd}} = [\sigma_k^2, \mathbf{W}_k]$. The most principled and general way to obtain the optimal values for $\boldsymbol{\theta}^k_{\mathrm{rbd}}$ is to minimize the corresponding marginal likelihood. Here, we optimize the hyperparameters $\boldsymbol{\theta}^k_{\mathrm{rbd}}$ from training data using the same marginal log likelihood as employed to estimate the hyperparameters of a zero mean GP (Rasmussen and Williams, 2006).

The RBD kernel, as given in Equation (5.8), covers only the function classes described by the RBD basis functions. However, in order to properly model a real robot, the function class describing the RBD is too limited and more functions might be needed. The common RBD does not model, for example, friction or the hydraulics tube dynamics, thus, the corresponding RBD basis functions do not described the dynamics resulting from such elements. The unmodeled elements can be taken in account by adding *additional* kernels to the RBD kernel to enrich the function class. A Gaussian kernel for example allows grasping arbitrary nonlinearities. Such an additional kernel will extend the spaces spanned by the RBD kernel to unknown function classes (Schölkopf and Smola, 2002; Rasmussen and Williams, 2006). Thus, the complete kernel that we used for learning is determined by

$$\mathrm{k}^k\left(\mathbf{x}_p, \mathbf{x}_q\right) = \mathrm{k}^k_{\mathrm{rbd}}\left(\mathbf{x}_p, \mathbf{x}_q\right) + \lambda_k \mathrm{k}\left(\mathbf{x}_p, \mathbf{x}_q\right)\,, \tag{5.9}$$

with the weighting parameter λ_k we can control the contribution of the additional kernel $\mathrm{k}(\cdot,\cdot)$ to the learning process. For learning the model, we combine a zero-mean GP with the kernel defined in Equation (5.9), where the hyperparameters of $\mathrm{k}^k_{\mathrm{rbd}}$ and k are optimized independently. Thus, the torque prediction for the k-th DoF can be given by

$$\bar{\tau}_k(\mathbf{x}_*) = \mathbf{k}_*^{kT}\left(\mathbf{K}_k + \sigma_k^2\mathbf{I}\right)^{-1}\mathbf{y}_k = \mathbf{k}_*^{kT}\boldsymbol{\alpha}_k^{\mathrm{rb}}\,. \tag{5.10}$$

Note that this way of incorporating prior knowledge via kernels is a general approach to include additional information in model learning. It can not only be used in the probabilistic framework introduced here, but in all kernel methods (e.g., support vector machines (Schölkopf *et al.*, 1997; Schölkopf and Smola, 2002)).

5.1.3 Evaluations

In this section, we evaluate the presented approaches in the context of learning inverse dynamics models for computed torque control. First, we give a short review of computed torque control. Subsequently, we show a comparison of the computational complexity and present the results in learning inverse dynamics. In Section 5.1.3.4, we report the performance of the tracking controller on our Barrett WAM shown in Figure 5.2 (a), where the learned models are employed for online torque prediction.

5.1.3.1 Feedforward Torque Control using Inverse Dynamics

The computed torque tracking control law determines the joint torques $\mathbf{u}$ that are required for the robot to follow a desired trajectory $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$ (Spong *et al.*, 2006). In general, the motor command $\mathbf{u}$ consists of two parts, a feedforward term $\mathbf{u}_{\text{FF}}$ to achieve the movement and a feedback term $\mathbf{u}_{\text{FB}}$ to ensure stability of the tracking. The feedback term can be a linear control law such as $\mathbf{u}_{\text{FB}} = \mathbf{K}_p\mathbf{e} + \mathbf{K}_v\dot{\mathbf{e}}$, where $\mathbf{e}$ denotes the tracking error with position gain $\mathbf{K}_p$ and velocity gain $\mathbf{K}_v$. The feedforward term $\mathbf{u}_{\text{FF}}$ is determined using an inverse dynamics model and, traditionally, using the analytical RBD model in Equation (5.2). If a sufficiently precise inverse dynamics model can be estimated, the resulting control law $\mathbf{u}=\mathbf{u}_{\text{FF}}(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d)+\mathbf{u}_{\text{FB}}$ will drive the robot along the desired trajectory accurately. However, if the model is not sufficiently precise, the tracking accuracy degrades drastically and low-gain control may become impossible (Nguyen-Tuong *et al.*, 2008a).

5.1.3.2 Prediction Speed Comparison

First, we study the computational complexity of torque prediction for all 7 DoFs, i.e., we compare the time needed for predicting $\mathbf{u}_{\text{FF}}$ for a query point $(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d)$ after having learned the mapping $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \mathbf{u}$. The results are shown in Figure 5.1. Here, the prediction vectors $\boldsymbol{\alpha}$ have been computed offline for each DoF (see Equations (5.5), (5.7) and (5.10)). During online prediction, we only need to compute the kernel vector $\mathbf{k}$, and, for GPR with RBD mean, the basis functions $\boldsymbol{\phi}$ have to be additionally evaluated for the query point. We compare the prediction speed of our semiparametric models with the "standard" GPR, i.e., a GP with zero mean and Gaussian kernel, and the traditional RBD model on training data sets with different sample size. As shown in Figure 5.1, the GPR models using RBD mean with fixed and with Gaussian prior on the dynamics parameter are as fast as the standard GPR during prediction. The reason is that the computations for the RBD mean term are very fast (see the computation time for the RBD model in Figure 5.1). Therefore, the prediction speed mostly depends on the evaluations of the dot product between the kernel and the prediction vector, which scales linearly in the number of training examples, i.e., $\mathcal{O}(n)$. Compared to the GPR with RBD mean, the computation for the GPR with RBD kernel is more complex, since we have to evaluate the RBD kernel in Equation (5.8) and, additionally, a common Gaussian kernel as shown in Equation (5.9).

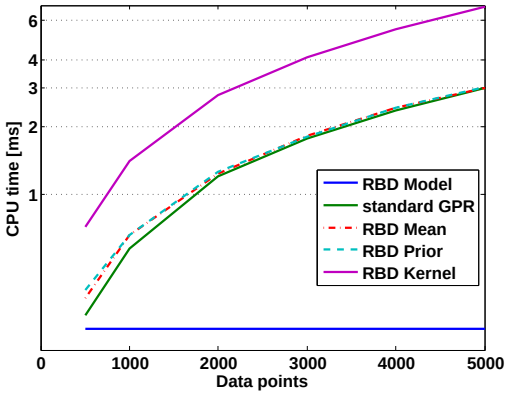
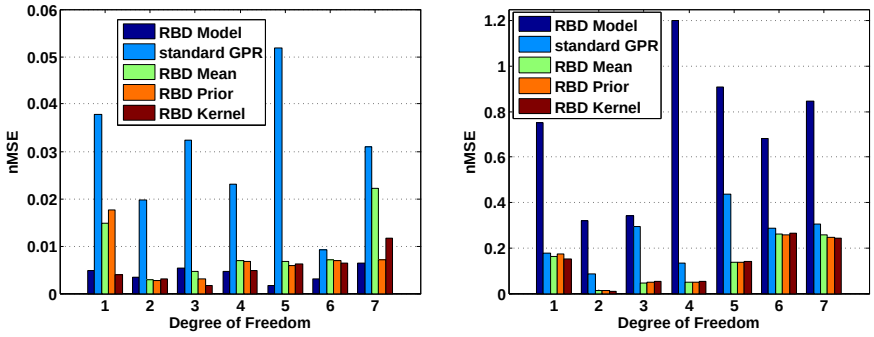


Figure 5.1: Average time in millisecond needed for prediction of 1 query point. For a better visualization, the computation time is plotted logarithmic in respect of the number of training examples. The time as stated above is the required time for prediction of all 7 DoF. For the learned models, the prediction time scales linearly with the training sample size. Comparing the prediction time, standard GPR is as fast as the GPR models with RBD mean. GPR with RBD kernel has the highest computational cost while the traditional RBD model remains the fastest.

5.1.3.3 Comparison in Learning Inverse Dynamics

In the following, we compare the performances of the semipametric models on learning inverse dynamics with the standard GPR and the RBD model. For this comparison, we use two small data sets, i.e, one with simulated and one with real Barrett WAM data, where each data set has 3000 data points for training and 3000 different ones for testing. The training and test trajectories are generated by superposition of sinusoidal movements similar to the one in (Nguyen-Tuong *et al.*, 2008a). Furthermore, we ensure that the test data is sufficiently different from the training data highlighting the generalization ability of the learned models. The results are shown in Figure 5.2 (b) and (c) where the approximation errors for each robot DoF on the test sets are given as normalized mean square error (nMSE), i.e., nMSE = MSE / variance of the target.

For the RBD model, we estimate the dynamics parameters $\boldsymbol{\beta}$ using linear regression from a large data set (130,000 data points) which covers the same part of the state space as the real Barrett training data. During the generation of simulated data, we apply this estimated RBD model for computation of the feedforward torques $\mathbf{u}_{\mathrm{FF}}$ while sampling the resulting joint torques $\mathbf{u}$ and joint trajectory $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ as training data. For so doing, we make sure that the RBD model describes the generated simulation data well. Subsequently, the same RBD model is also applied as mean function for the semiparametric models with RBD mean. The hyperparameters for the Gaussian kernel used by GPR with RBD mean are estimated from training data by optimizing the marginal log likelihood (Rasmussen and Williams, 2006). The optimization is performed using common line search approaches such as quasi-Newton methods (Zhu *et al.*, 1997). We apply the same approach for es-



(a) **Error on simulation data** (b) **Error on real WAM data**

Figure 5.2: (a) Error (nMSE) on simulated data from a robot model for every DoF. Generating the simulation data, the RBD model is used to compute the feedforward torques $\mathbf{u}_{\mathrm{FF}}$, while the joint torques $\mathbf{u}$ and joint trajectory $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ are sampled for model learning. **(b)** Error (nMSE) on real robot data for every DoF. For simulation data, the error of the RBD model is very small, as the model describes the data well. Using this RBD model, the semiparametric models (with RB mean and RBD kernel) show very good learning results similar to the performance of the RBD model. If the RBD model does not explain the data well, such as for the real Barrett WAM data in **(b)**, the semiparametric models can improve the performance by learning the error between sampled data and RBD model. As shown by the results, the standard GPR model is sometimes not able to generalize well for sufficiently different test data due to the small size of the training data sets.

timating the hyperparameters of the RBD kernel as shown in Equations (5.8,5.9), where the control parameter λ is determined by cross-validation to be 1. For the cross-validation, the increment is chosen to be 0.5 in an interval ranging from 0.5 to 5.

As shown in the results in Figure 5.2 (a) and (b), the semiparametric models are able to combine the strengths of both models, i.e., the parametric RBD model and the nonparametric GP model. If the parametric RBD model explains the sampled data well as in the case of simulated data, the semiparametric models rely mostly on the parametric term resulting in learning performances close to the RBD model. If the RBD model does not match the sampled data in case of real robot data, the semiparametric models either attempt to learn the errors made by the parametric model as done by GPR with RBD mean, or use the additional kernel to fit the missing function classes. It can also be seen by the results that pure nonparametric models such as standard GPR may have generalization difficulty, if the training data is not sufficiently large and rich. In that case, the prediction performance of pure nonparametric models degrades for the unknown parts of the state space. Considering the experiments with real robot data in Figure 5.2 (c), semiparametric models are competitive to pure nonparametric models even in the case when the parametric models are not precise. The results in Figure 5.2 (c) can be improved for

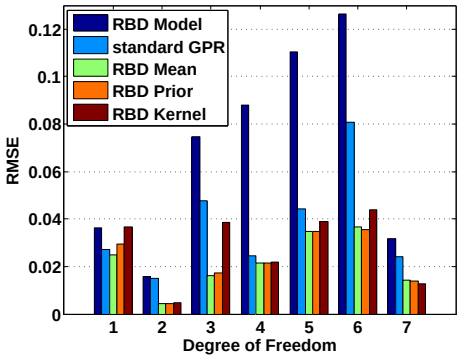


Figure 5.3: Tracking error on the real Barrett WAM computed as RMSE for all 7 DoF. The semiparametric models outperform the RBD model and also the standard GPR model in most cases. Especially, for the robot elbow and wrist (4., 5. and 6. DoF) there is a significant improvement compared to the parametric RBD model, as these DoF suffer from many nonlinearities which are not described by the analytical RBD model.

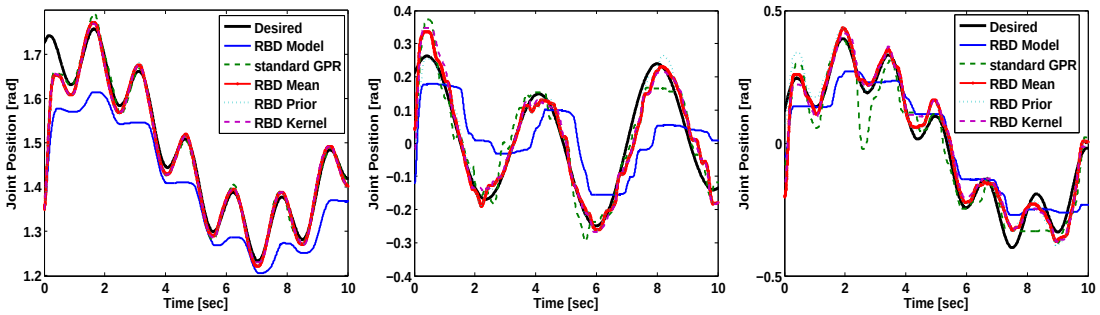
the semiparametric models, if the applied RBD model is estimated more accurately using more sophisticated estimation approaches (Ting *et al.*, 2009). However, Figure 5.2 (b) also shows that GPR with RBD mean function might tend to overfit the data, if the deviations between the RBD model and training data are small (e.g., for 1st DoF). Careful optimization of the hyperparameters for the semiparametric models may alleviate this problem.

Comparing the GP models with fixed RBD mean and with additional prior on the dynamics parameters, the differences in learning performance are relatively small. One explanation for this result is that the uncertainty of the dynamics parameters, as encoded by the variance $\mathbf{B}$ in Equation (5.6), is already taken in account by the nonparametric term for learning the error between the RBD model and observed data. Therefore, the variance $\mathbf{B}$ may not present sufficient new information for improving the model. However, it provides additional robustness and may be very valuable in that way.

5.1.3.4 Application in Robot Computed Torque Control

In this section, we apply the learned models from Section 5.1.3.3 for a robot tracking control task on the Barrett WAM, while the models are used to predict the feedforward torques $\mathbf{u}_{\mathrm{FF}}$ as described in Section 5.1.3.1. The tracking results are reported in Figure 5.3, where the tracking error is computed as root mean square error (RMSE). The error is evaluated after a tracking duration of 60 sec on the robot. For the tracking task, we set the tracking gains $\mathbf{K}_p$ and $\mathbf{K}_v$ to very low values taking in account the requirement of compliance. Furthermore, the generated desired test trajectory is different than the training and test trajectories used in Section 5.1.3.3, highlighting the generalization ability of the learned models.

As shown by the results in Figure 5.3, the semiparametric models largely outper-



(a) **Tracking on WAM for the 4th DoF**

(b) **Tracking on WAM for the 5th DoF**

(c) **Tracking on WAM for the 6th DoF**

Figure 5.4: Tracking performance in joint space on the Barrett WAM for the first 10 sec. **(a)** Performance for the 4th DoF (elbow). **(b)** Performance for the 5th DoF (wrist rotation). **(c)** Performance for the 6th DoF (wrist flexion extension). The RBD model does not provide a satisfying performance while the learned models exhibit good tracking results. Standard nonparametric GPR have some problems in generalization for unknown test trajectory as small training data sets are used, resulting in a deteriorated tracking performance (green dashed line).

form the RBD model and the standard GPR in compliant tracking performance. Especially, for the robot elbow (4th DoF) and the robot wrist (5th and 6th DoF) we observe a significant improvement compared to the RBD model. The reason is that these DoFs suffer from many unknown nonlinearities which can not be explained by the analytical RBD model, such as complex friction, stiction and backlash due to the gear drive etc. The tracking performance for these DoFs is additionally shown in Figure 5.4. Here, one can see that the Barrett WAM using the RBD model fails to follow the rhythmical movements of the desired trajectory in the compliant mode (for example, due to suboptimal friction compensation). While the semiparametric models enable the robot to follow the desired trajectory well, the standard nonparametric GPR exhibits several problems in prediction of the feedforward torques resulting in instantaneous jumps in joint trajectory as shown by Figure 5.4 (b) and (c).

In this experiment, the sampling time of the Barrett WAM is 500 Hz ($\triangleq$ 2 ms). For the real-time online torque prediction, we compute the prediction in parallel to the robot controller in a separate process. Thus, we update the feedforward torques $\mathbf{u}_{\mathrm{FF}}$ according to the computational speed of the prediction models, while maintaining the feedback torques $\mathbf{u}_{\mathrm{FB}}$ for every sampling step ensuring the tracking stability of the robot.

5.1.4 Conclusion of Section 5.1

In this work, we have introduced two semiparametric approaches to learning the inverse dynamics models while combining the strengths of parametric RBD model

and nonparametric GP models. The knowledge of the parametric RBD model is incorporated into the nonparametric GP model either as a RBD mean function or as a RBD kernel. We evaluate the semiparametric models in learning inverse dynamics for robot tracking control. The results on the Barrett WAM show that the semiparametric models provide a higher model accuracy and better generalization for unknown trajectories compared to RBD and standard GPR. The gist of semiparametric models is that they exhibit a competitive learning performance even on small and poor data sets, overcoming the limitation of pure nonparametric learning methods while exploiting the prior information encoded in parametric models.

5.2 Learning Task-Space Tracking Control with Kernels

Task-space tracking control is essential for robot manipulation. In practice, task-space control of redundant robot systems is known to be susceptive to modeling errors. Here, data driven learning methods may present an interesting alternative approach. However, learning models for task-space tracking control from sampled data is an ill-posed problem. In particular, the same input data point can yield many different output values, which can form a non-convex solution space. Because the problem is ill-posed, models cannot be learned from such data using common regression methods. While learning of task-space control mappings is *globally* ill-posed, it has been shown in recent work that it is *locally* a well-defined problem. In this section, we use this insight to formulate a local, kernel-based learning approach for online model learning for task-space tracking control. For evaluations, we show the ability of the method for online model learning for task-space tracking control of redundant robots.

5.2.1 Introduction

Control of redundant robots in operational space, especially task-space tracking control, is an essential ability needed in robotics (Khatib, 1987; Sentis and Khatib, 2005). Here, the robot's end-effector follows a desired trajectory in task-space, while distributing the resulting forces onto the robot's joints. Analytical formulation of task-space control laws requires given kinematics and dynamics models of the robot. However, modeling the kinematics and dynamics is susceptive to errors. One promising possibility to overcome such inaccurate hand-crafted models is to learn them from data. From a machine learning point of view, learning of such models can be understood as a regression problem. Given input and output data, the task is to learn a model describing the input to output mapping.

Using standard regression techniques, such as Gaussian process regression (Rasmussen and Williams, 2006), support vector regression (Smola and Schölkopf, 2004) or locally weighted regression (Atkeson *et al.*, 1997a), a model can be approximated to describe a *single-valued* mapping (i.e., one-to-one) between the input and output data. The single-valued property requires that the same input point should

always yield the same single output value, resulting in a well-defined learning problem. However, this situation changes when learning a torque prediction model for task-space tracking control of redundant robots. Here, we are confronted with the problem of learning *multi-valued* or one-to-many mappings. In this case, standard regression techniques can not be applied. Naively learning such multi-valued mappings from sampled data using standard regression will average over multiple output values in a potentially non-convex solution space (Peters and Schaal, 2008). Thus, the resulting model will output degenerate predictions, which lead to poor control performance and may cause damage to the redundant robot system.

However, despite being a globally ill-posed problem, learning such task-space control mappings is locally well-defined (D'Souza *et al.*, 2001; Peters and Schaal, 2008). In this work, we employ this insight to formulate an online local learning approach, appropriate for learning models that allow prediction with such multi-valued mappings. The key idea is to localize a model in configuration space, while continuously updating this model online by including new data points and, eventually, removing old points. Here, local data points are inserted or removed based on a kernel distance measure. Due to the local consistency, a prediction model can be learned. The proposed model parametrization allows us to apply the kernel-trick and, therefore, enables a formulation within the kernel learning framework (Schölkopf and Smola, 2002). Kernel methods have been shown to be a flexible and powerful tool for learning general nonlinear models. In task-space tracking control of redundant robots, the model parametrization enables a projection of the joint-space stabilization torques into the task's null-space.

The remainder of the section will be organized as follows: first, we give a brief overview of task-space control and provide a review of related work. In Section 5.2.2, we describe our approach to learn task-space tracking control. The proposed method will be evaluated on redundant robot systems, e.g., a simulated 3-DoF robot and 7-DoF Barrett WAM, for task-space tracking control in Section 5.2.3. The most important lessons from this research project will be summarized in Section 5.2.4.

5.2.1.1 Problem Statement

To obtain an analytical task-space control law, we first need to model the robot's kinematics (Khatib, 1987). The relationship between the task-space and the joint-space of the robot is usually given by the forward kinematics model $\boldsymbol{x} = f(\boldsymbol{q})$. Here, $\boldsymbol{q} \in \mathbb{R}^m$ denotes the robot's configuration in the joint-space and $\boldsymbol{x} \in \mathbb{R}^d$ represents the task-space position and orientation. For redundant robot systems, it is necessary that $m > d$. The task-space velocity and acceleration are $\dot{\boldsymbol{x}} = \mathbf{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{x}} = \dot{\mathbf{J}}(\boldsymbol{q})\dot{\boldsymbol{q}} + \mathbf{J}(\boldsymbol{q})\ddot{\boldsymbol{q}}$, where $\mathbf{J}(\boldsymbol{q}) = \partial f / \partial \boldsymbol{q}$ is the Jacobian. For computing the joint torques necessary for the robot to follow the task-space trajectory, a model of the robot dynamics is required. A typical dynamics model can be given in the form of $\boldsymbol{u} = \mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{F}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, see (Spong *et al.*, 2006) for more details. Here, $\boldsymbol{u}$ denotes the joint torque, $\mathbf{M}(\mathbf{q})$ is the generalized inertia matrix of the robot, and $\mathbf{F}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is a vector containing forces, such as gravity, centripetal and Coriolis forces. Combining

the dynamics model with the kinematics model yields one possible operational space control law

$$\boldsymbol{u} = \mathbf{M}\mathbf{J}_w^{\dagger}(\ddot{\boldsymbol{x}}_{\text{ref}} - \dot{\mathbf{J}}\dot{\boldsymbol{q}}) + \mathbf{F}\,, \tag{5.11}$$

where $\mathbf{J}_w^{\dagger}$ denotes the weighted pseudo-inverse of $\mathbf{J}$, as described in (Peters *et al.*, 2008; Nakanishi *et al.*, 2008). In Equation (5.11), a task-space attractor $\ddot{\boldsymbol{x}}_{\text{ref}}$ is employed for tracking the actual task-space acceleration $\ddot{\boldsymbol{x}}$ (Nakanishi *et al.*, 2008). Here, the task-space attractor is formulated as $\ddot{\boldsymbol{x}}_{\text{ref}} = \ddot{\boldsymbol{x}}_{\text{des}} + \mathbf{G}_{vv}(\dot{\boldsymbol{x}}_{\text{des}} - \dot{\boldsymbol{x}}) + \mathbf{G}_{pp}(\boldsymbol{x}_{\text{des}} - \boldsymbol{x})$, where $\boldsymbol{x}_{\text{des}}$, $\dot{\boldsymbol{x}}_{\text{des}}$ and $\ddot{\boldsymbol{x}}_{\text{des}}$ denote the desired task-space trajectory. $\mathbf{G}_{vv}$ and $\mathbf{G}_{pp}$ are positive task-space gain matrices.

To ensure stable tracking in the robot's joint-space, the controller command $\boldsymbol{u}$ in Equation (5.11) is usually extended by a null-space controller term $\boldsymbol{u}_0$. Thus, the total joint torque command $\boldsymbol{u}_{\text{joint}}$ is given as

$$\boldsymbol{u}_{\text{joint}} = \boldsymbol{u} + \left(\mathbf{I} - \mathbf{J}_w^{\dagger}\mathbf{J}\right)\boldsymbol{u}_0\,. \tag{5.12}$$

The term $\boldsymbol{u}_0$ can be interpreted as joint-space stabilizing torques which are only effective in the task's null-space and, thus, do not interfere with the task achievement (Nakanishi *et al.*, 2008). The null-space controller command $\boldsymbol{u}_0$ can be chosen such that the redundant robot is pulled towards a desired rest posture $\boldsymbol{q}_{\text{rest}}$, i.e., $\boldsymbol{u}_0 = -\mathbf{G}_v\dot{\boldsymbol{q}} - \mathbf{G}_p(\boldsymbol{q} - \boldsymbol{q}_{\text{rest}})$, where $\mathbf{G}_p$ and $\mathbf{G}_v$ are positive joint-space gain matrices.

As indicated by Equations (5.11, 5.12), an analytical formulation for task-space control requires given analytical kinematics and dynamics models. As modeling these relationships can be inaccurate in practice, model learning presents a promising alternative. In the task-space tracking problem shown in Equation (5.11), we want to learn mappings from inputs $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}})$ to targets $\boldsymbol{u}$. However, this mapping is one-to-many (D'Souza *et al.*, 2001; Peters and Schaal, 2008), as there can be many torques $\boldsymbol{u}$ which correspond to the same task-space acceleration $\ddot{\boldsymbol{x}}$ given $\boldsymbol{q}, \dot{\boldsymbol{q}}$. Thus, naively learning a task-space control model for redundant robots from sampled data may result in a degenerate mapping. In practice, such degenerate models will provide inconsistent torque predictions.

5.2.1.2 Related Work

Learning multi-valued mappings has previously been investigated in the field of neural motor control (Jordan and Rumelhart, 1992; Wolpert and Kawato, 1998). In (Jordan and Rumelhart, 1992), the multi-valued relationship is resolved for a particular output solution by jointly approximating the forward and inverse mapping. Originally, the introduced forward-inverse learning principle has been formulated in the framework of neural networks (Jordan and Rumelhart, 1992). In a neural networks based implementation, the forward model is chained with the multi-valued inverse model, where the prediction errors made by the forward model are used to adapt the weight values of the inverse model for a given output solution. However, training such neural networks is well-known to be problematic due to local minima, instability and difficulties in selecting the network structures. Nevertheless,

this framework of learning forward and inverse models initiated a number of follow-up research projects, such as (Wolpert and Kawato, 1998; Bhushan and Shadmehr, 1999). For example, Bhushan and Shadmehr (1999) have presented considerable evidence that the forward-inverse models approach may be useful for explaining human motor control. Wolpert and Kawato (1998) approximate pairwise forward-inverse models for different motor control tasks and, subsequently, combine them for prediction.

In the broader sense, the pairwise forward-inverse model approach (Wolpert and Kawato, 1998) can be understood as a *local* learning method, where the data is first partitioned into local regions for which local forward-inverse model pairs are subsequently approximated. A local learning approach is also employed by Tevatia and Schaal (2008) and D'Souza *et al.* (2001) to learn models for robot inverse kinematics, where locally weighted regression techniques are used. Peters and Schaal (2008) further extend the locally weighted regression approach for learning operational space robot control. While Peters and Schaal (2008) attempt to learn a direct mapping for predicting the joint torques for control, Salaun *et al.* (2009) first learn a forward kinematics model and invert the learned model afterwards. Subsequently, they combine it with an inverse dynamics model to generate the required torque command.

Compared to previous local learning approaches, we attempt to learn a *single* localized model while continuously updating this local model depending on the robot's current configuration. Due to the local consistency, the model learning problem is well-defined. We propose a model parameterization which enables kernel-based learning of torque prediction models for task-space tracking control. The model parametrization also allows a null-space projection, which is necessary to stabilize the robot in the joint-space without interfering with the task-space performance.

5.2.2 Learning Task-Space Tracking with Kernels

In this problem, we want to learn the mapping from inputs $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}})$ to outputs $\boldsymbol{u}$, similar to the one described by Equation (5.11). This mapping is subsequently used for predicting the outputs for given query points. As such one-to-many mappings are locally well-defined (D'Souza *et al.*, 2001; Peters and Schaal, 2008), they can be approximated with a local kernel learning approach. Here, our model will be localized in the robot's joint position space. The local data is incrementally updated, as the robot moves to new state space regions. Every local data point is weighted by its distance to the most recent joint position. Thereby, we ensure that the local data points form a well-defined set that is appropriate for model learning. Using the weighted local data points, the model's parameters can be obtained by minimizing a cost function. To place the model into the kernel learning framework, we propose a model parametrization appropriate for the application of the kernel-trick. The parametrization is also suitable for robot tracking control in the task-space.

In the following sections, we will describe how the model is localized and updated in an online setting. We present the parametrization of the local model and show

how the corresponding parameters can be obtained from data. Subsequently, we show how the learned local model can be used in online learning for task-space robot control.

5.2.2.1 Model Localization

For learning task-space tracking, we use a *single* local model for torque prediction, where the model is localized in the robot's joint position space. This local data set needs to be continuously updated in the online setting, as the robot frequently moves to new state space regions. In this section, we describe the measures needed to localize and update the model during online learning. The procedure includes insertion of new data points into the local data set and removal of old ones.

Insertion of New Data Points. For deciding whether to insert a new data point into the local data set, we consider the distance measure δ, as proposed by Schölkopf *et al.* (1999) and Nguyen-Tuong and Peters (2010a). This measure is defined by

$$\delta(\boldsymbol{q}^*) = \left\| \sum_{i=1}^{N} a_i \boldsymbol{\psi}(\boldsymbol{q}_i) - \boldsymbol{\psi}(\boldsymbol{q}^*) \right\|^2 , \tag{5.13}$$

where $\boldsymbol{\psi}$ is a feature vector and a_i denote the coefficients of linear expansion. The value $\delta(\boldsymbol{q}^*)$ is a measure that indicates the *distance* of a point $\boldsymbol{q}^*$ to the surface spanned by the current set $\boldsymbol{L} = \{\boldsymbol{q}_i\}_{i=1}^N$. The coefficients a_i in Equation (5.13) is given by $\boldsymbol{a} = \boldsymbol{K}_a^{-1}\boldsymbol{k}$ (Nguyen-Tuong and Peters, 2010a). Here, $\boldsymbol{K}_a = k(\boldsymbol{L}, \boldsymbol{L})$ is the kernel matrix evaluated for the local joint positions $\boldsymbol{L}$, and $\boldsymbol{k} = k(\boldsymbol{L}, \boldsymbol{q}^*)$ is the kernel vector. Using this result, δ can be written as

$$\delta(\boldsymbol{q}^*) = k(\boldsymbol{q}^*, \boldsymbol{q}^*) - \boldsymbol{k}^T \boldsymbol{a} . \tag{5.14}$$

The value δ increases with the distance of $\boldsymbol{q}^*$ from the surface defined by $\boldsymbol{L}$. Using Equation (5.14), we can make decisions for inserting new data points. If the δ values of new data points exceed a given threshold η, we will insert these points into the local model. The employed measure δ ensures that new data points will be included into the local set, when the robot moves to new joint-space regions.

Removal of Old Data Points. For removing data points from the local set, we select the point which is the *farthest* from the most recent joint position $\boldsymbol{q}$. Here, we employ a Gaussian kernel as a distance measure between $\boldsymbol{q}$ and other local data points $\boldsymbol{q}_i$

$$k(\boldsymbol{q}, \boldsymbol{q}_i) = \exp\left(-\frac{1}{2} (\boldsymbol{q} - \boldsymbol{q}_i)^T \mathbf{W} (\boldsymbol{q} - \boldsymbol{q}_i) \right) , \tag{5.15}$$

where $\mathbf{W}$ denotes the kernel width. Removing the farthest local data point implies that its kernel measure $k(\cdot, \cdot)$ is the smallest. By continuously inserting and removing local data points, we make sure that the local data set is suitable for the current region of the state space.

5.2.2.2 Model Parametrization

The described insertion and removal operations result in a data set localized in the joint position space. Given this local data set $\boldsymbol{D} = \{\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, \ddot{\boldsymbol{x}}_i, \boldsymbol{u}_i\}_{i=1}^N$, we can now learn a model for torque prediction for task-space control. From Equation (5.11), we can see that the joint torque $\boldsymbol{u}$ is *linear* in the task-space acceleration $\ddot{\boldsymbol{x}}$, while it is *nonlinear* in the joint position $\boldsymbol{q}$ and velocity $\dot{\boldsymbol{q}}$. Using this insight, we propose the following parametrization for the local model

$$\boldsymbol{u} = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\theta}_0^T \ddot{\boldsymbol{x}}\,, \tag{5.16}$$

where $\boldsymbol{\phi}$ is a vector containing nonlinear functions projecting $[\boldsymbol{q}, \dot{\boldsymbol{q}}]$ into some high-dimensional spaces. Generally, $\ddot{\boldsymbol{x}}$ can have d dimensions and $\boldsymbol{u}$ is a m dimensional vector. Following the representer theorem (Schölkopf and Smola, 2002), the coefficients $\boldsymbol{\theta}, \boldsymbol{\theta}_0$ in Equation (5.16) can be expanded in term of N local data points. Hence, we have

$$\boldsymbol{\theta} = \textstyle\sum_{i=1}^N \alpha_i \boldsymbol{\phi}(\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)\,, \quad \boldsymbol{\theta}_0 = \sum_{i=1}^N \alpha_0^i \ddot{\boldsymbol{x}}_i\,,$$

where α_i, α_0^i are the corresponding linear expansion coefficients. Inserting the linear expansions into Equation (5.16) and re-writing it in term of N sample data points yields

$$\boldsymbol{U} = \boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{P}\boldsymbol{\alpha}_0\,, \tag{5.17}$$

where $\boldsymbol{U} = \{\boldsymbol{u}_i\}_{i=1}^N$. The elements $[\boldsymbol{K}]_{ij} = \langle \boldsymbol{\phi}(\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i), \boldsymbol{\phi}(\boldsymbol{q}_j, \dot{\boldsymbol{q}}_j) \rangle$ are the pairwise inner-products of the feature vectors. Thus, $[\boldsymbol{K}]_{ij}$ can be represented with kernels (Schölkopf and Smola, 2002), i.e., $[\boldsymbol{K}]_{ij} = \tilde{k}([\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i], [\boldsymbol{q}_j, \dot{\boldsymbol{q}}_j])$. The matrix $\boldsymbol{K}$ is thus a kernel matrix evaluated at the joint position and velocity employing the kernel $\tilde{k}(\cdot, \cdot)$. Using this so-called kernel-trick, only the kernel function $\tilde{k}$ needs to be determined instead of an explicit feature mapping $\boldsymbol{\phi}$ (Schölkopf and Smola, 2002). Similarly, the elements $[\boldsymbol{P}]_{ij} = \langle \ddot{\boldsymbol{x}}_i, \ddot{\boldsymbol{x}}_j \rangle$ represent the pairwise inner-products of the task-space acceleration $\ddot{\boldsymbol{x}}$. Thus, $\boldsymbol{P}$ can be understood as a kernel matrix where linear kernels are applied.

5.2.2.3 Online Learning of the Local Model

Learning requires the estimation of the expansion parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ in Equation (5.17) from the local data set. Employing the learned model, we can predict the output for a query point. In particular, for online learning the expansion parameters have to be estimated incrementally, as the data arrives as a stream over time.

Estimation of Model Parameters. Using the model parametrization in Section 5.2.2.2, the expansion parameters can be estimated from data by minimizing an appropriate cost function $\mathcal{L}$ given by

$$\mathcal{L} = \frac{\gamma}{2}\left(\boldsymbol{\alpha}^T \boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{\alpha}_0^T \boldsymbol{P}\boldsymbol{\alpha}_0\right) + \frac{1}{2}\left(\boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{P}\boldsymbol{\alpha}_0 - \boldsymbol{U}\right)^T \mathbf{N}\left(\boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{P}\boldsymbol{\alpha}_0 - \boldsymbol{U}\right). \tag{5.18}$$

Algorithm 6 Online learning of the local model.

Given: local data set $\boldsymbol{D}=\{\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, \ddot{\boldsymbol{x}}_i, \boldsymbol{u}_i\}_{i=1}^{N}$, $N_{\max}$, threshold value η.
Input: new input $\{\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}\}$ and output $\boldsymbol{u}$.

Evaluate the distance of $\boldsymbol{q}$ to the surface defined by $\boldsymbol{L}=\{\boldsymbol{q}_i\}_{i=1}^{N}$
based on the measure $\delta(\boldsymbol{q})$ from Equation (5.14).
if $\delta(\boldsymbol{q}) > \eta$ **then**
  for $i=1$ **to** N **do**
    Compute: $\mathbf{N}(i,i) = k(\boldsymbol{q}, \boldsymbol{q}_i)$ using Equation (5.15).
  end for
  if $N < N_{\max}$ **then**
    Include the new point: $\boldsymbol{D}_{N+1}=\{\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}, \boldsymbol{u}\}$.
  else
    Find the farthest point: $j=\min_i \mathbf{N}(i,i)$.
    Replace the j-th data point by the query point: $\boldsymbol{D}_j=\{\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}, \boldsymbol{u}\}$.
  end if
  Update the expansion parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ incrementally using Equation (5.19),
  while re-weighting every local data point with the new distance metric $\mathbf{N}$.
end if

The first term in Equation (5.18) acts as regularization, while the second term represents a squared loss based data-fit. In Equation (5.18), the parameter γ controls the regularization and the diagonal matrix $\mathbf{N}$ denotes the *weight* for each data point in the local set. The minimization of $\mathcal{L}$ w.r.t. $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ yields the analytical solution

$$\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{K} + \gamma \mathbf{N}^{-1} & \boldsymbol{P} \\ \boldsymbol{K} & \boldsymbol{P} + \gamma \mathbf{N}^{-1} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{U} \end{bmatrix}. \tag{5.19}$$

The weighting metric $\mathbf{N}$ incorporates a distance measure of each local data point to the most recent point in the local set. Here, we employ a kernel distance measure in the joint position space, as given in Equation (5.15). The weighting metric $\mathbf{N}$ ensures that the local data will form a well-defined set appropriate for the model learning step.

Online Model Learning. As the data arrives continuously in the online setting, Equation (5.19) has to be updated incrementally. Such incremental updates require adjusting the corresponding row and column of the inverse matrix, i.e., a rank-one update of the inverse matrix (Seeger, 2007b; Nguyen-Tuong and Peters, 2009). Additionally, every data point in the local set has to be re-weighted by its distance to the most current point after every insertion and removal step. In practice, we initialize the inverse matrix in Equation (5.19) as a diagonal matrix, where the number $N_{\max}$ of local data points is fixed. During online learning, the inverse matrix is first updated $N_{\max}$ times while filling up the local data set. Subsequently, old data points have to be removed when new points are inserted. The complete procedure for learning the local model is summarized in the Algorithm 6.

Algorithm 7 Online prediction for task-space control.

Given: a rest posture $\boldsymbol{q}_{\text{rest}}$, local data $\ddot{\boldsymbol{X}} = \{\ddot{\boldsymbol{x}}_i\}_{i=1}^N$ and $\boldsymbol{Q} = \{\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i\}_{i=1}^N$, expansion parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$.
Input: query point $\{\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{x}_{\text{des}}, \dot{\boldsymbol{x}}_{\text{des}}, \ddot{\boldsymbol{x}}_{\text{des}}\}$.

Compute null-space control torque $\boldsymbol{u}_0$.
Compute null-space projection matrix $\boldsymbol{H} = \boldsymbol{\alpha}_0^T \ddot{\boldsymbol{X}}$.
Compute task-space attractor $\ddot{\boldsymbol{x}}_{\text{ref}}$.
Compute joint torque control $\boldsymbol{u}_{\text{joint}}$ as given in Equation (5.21).

Prediction. With the optimization results from Equation (5.19), the prediction $\hat{\boldsymbol{u}}$ for a query point $[\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}_{\text{ref}}]$ can be computed as

$$\hat{\boldsymbol{u}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}_{\text{ref}}) = \boldsymbol{\alpha}^T \tilde{k}(\boldsymbol{Q}, [\boldsymbol{q}, \dot{\boldsymbol{q}}]) + \boldsymbol{\alpha}_0^T \langle \ddot{\boldsymbol{X}}, \ddot{\boldsymbol{x}}_{\text{ref}} \rangle \,, \tag{5.20}$$

where $\ddot{\boldsymbol{X}} = \{\ddot{\boldsymbol{x}}_i\}_{i=1}^N$ and $\boldsymbol{Q} = \{\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i\}_{i=1}^N$.

5.2.2.4 Using Local Model for Task-Space Control

Up to now, we have learned a well-defined local model to predict the joint torques required to drive the robot along a desired task-space trajectory. However, even after obtaining a perfect prediction of the necessary torques, it is not clear whether the robot will be stable in the joint-space. Thus, we need to explore ways to stabilize the robot in the joint-space without interfering the task-space performance, as done in analytical task-space control (see Equation (5.12)). Here, the key idea is to *project* the stabilizing torques $\boldsymbol{u}_0$ into the null-space of the "task relevant" part.

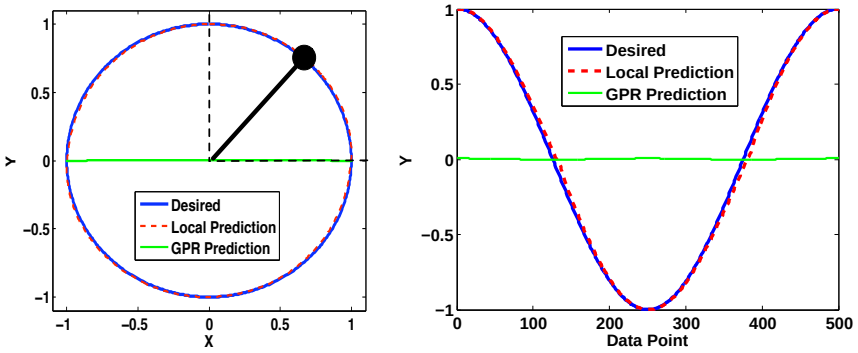
From Equation (5.20), it can be seen that the second term is the task relevant part, as this term explicitly depends on $\ddot{\boldsymbol{x}}_{\text{ref}}$. Therefore, for the robot joint-space stabilization, we can project the stabilization torques $\boldsymbol{u}_0$ into the null-space of this term. Hence, the total joint torque controller command $\boldsymbol{u}_{\text{joint}}$ can be computed as

$$\boldsymbol{u}_{\text{joint}} = \boldsymbol{\alpha}^T \tilde{k}(\boldsymbol{Q}, [\boldsymbol{q}, \dot{\boldsymbol{q}}]) + \boldsymbol{\alpha}_0^T \langle \ddot{\boldsymbol{X}}, \ddot{\boldsymbol{x}}_{\text{ref}} \rangle + (\mathbf{I} - \boldsymbol{H}(\boldsymbol{H}^T \boldsymbol{H})^{-1} \boldsymbol{H}^T) \boldsymbol{u}_0 \,. \tag{5.21}$$

The null-space projection is then given by the matrix $\boldsymbol{H} = \boldsymbol{\alpha}_0^T \ddot{\boldsymbol{X}}$. The resulting null-space projection allows joint-space stabilization based on $\boldsymbol{u}_0$ without interfering the task performance. The procedure for online torque prediction in task-space tracking control is summarized in the Algorithm 7.

5.2.3 Robot Evaluations

In this section, we evaluate the proposed approach for learning task-space control, as described in Section 5.2.2. First, we show for a toy example how a non-unique function can be learned in the online setting using this local learning approach. This example further illustrates the basic idea behind the local learning principle when used for approximating a multi-valued mapping. Subsequently, we show the



(a) **Trajectory of a pendulum in the $x-y$ space**

(b) **Prediction of the outcome y**

Figure 5.5: An example of learning a non-unique function. For the pendulum shown in (a), we have for each input position x two possible output values. Naively learning a global mapping $\boldsymbol{x} \rightarrow \boldsymbol{y}$ using GPR (Rasmussen and Williams, 2006) results in an average over multiple output solutions, as shown in (b). However, when the mapping is learned locally within the vicinity of the query point in an online setting, the model learning problem is well-defined resulting in a proper prediction.

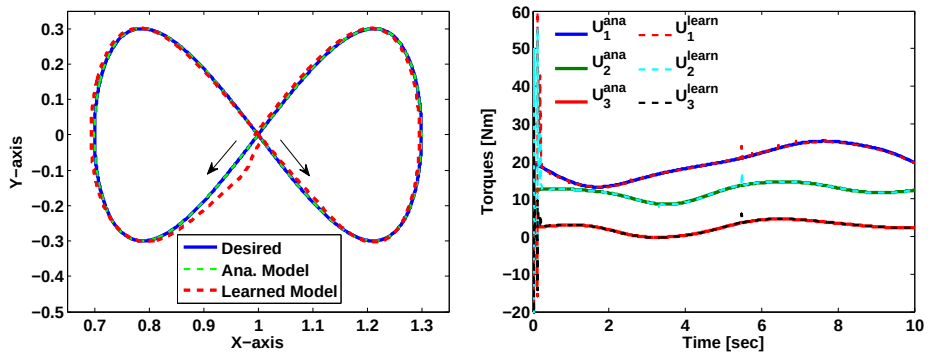
ability of our method in learning torque prediction models for task-space tracking control of redundant robot systems. The control experiments are performed with both simulated 3-DoF robot and 7-DoF anthropomorphic Barrett arm as shown in Figure 3.1.

5.2.3.1 Online Learning of a Non-unique Function

As benchmark example, we create a one-dimensional non-unique function shown in Figure 5.5. In this example, there is a pendulum that can rotate in the $\boldsymbol{x}-\boldsymbol{y}$ plane. For a circular rotation, the trajectory of $\boldsymbol{x}$ and $\boldsymbol{y}$ is given by $x_i = \sin(t_i)$ and $y_i = \cos(t_i)$ for t_i ranging from 0 to 2π. For the experiment, we sample 500 data points from the generated trajectory. If we employ $\boldsymbol{x}$ as input and $\boldsymbol{y}$ as the target output, we will have a non-unique prediction problem.

In this example, the parametrization of the local model is given by $\boldsymbol{y} = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x})$. While the model is localized in the $\boldsymbol{x}$ space, we update the local data set and learn the model in the online setting, as described in Section 5.2.2. For online model learning, we incrementally feed the data to the algorithm. Figure 5.5 shows the results after one sweep through the data set. To highlight the difficulty in learning such multi-valued mappings from data, the well-known Gaussian process regression (Rasmussen and Williams, 2006) is employed to globally approximate the mapping $\boldsymbol{x} \rightarrow \boldsymbol{y}$. The comparison between the two methods is given in Figure 5.5.

In the experiment, the size of the local data set is chosen to be 10. Here, we first fill the local set up incrementally and, subsequently, update the local model online by insertion and removal. We employ the Gaussian kernel for the localization step,



(a) **Task-space tracking** (b) **Control torque for each DoF**

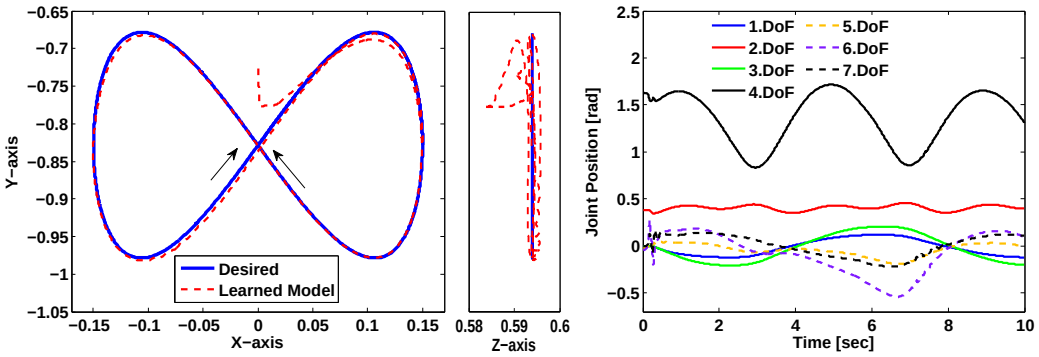
Figure 5.6: Task-space tracking by a 3-DoF robot. Here, we compared task-space tracking using perfect analytical model with one that was learned online. As a perfect model is used, the analytical task-space tracking control yields a perfect tracking, as shown in (a). During online learning, the learned task-space controller continuously improves the task-space tracking performance. As shown in (b), the learned task-space control torques $\boldsymbol{u}_{\text{joint}}$ converge to the perfect analytical torques after a short transient phase.

as well as for model learning. The kernel width $\mathbf{W}$ is optimized by cross-validation, and the threshold η is set to be 0.001. As the farthest point in the input space is removed when a new point is inserted, one can observe that the local data set always covers a region in the vicinity of the recent query point. Since the local data set forms a convex solution space, the local model can be learned and results in a proper prediction of the targets $\boldsymbol{y}$, shown in Figure 5.5 (b).

5.2.3.2 Online Model Learning for Task-Space Tracking Control

In this section, we apply the proposed method to learning torque prediction models for task-space control of a simulated 3-DoF robot and the simulated 7-DoF Barrett WAM. In the experiments, the models are learned online, while the robots are controlled to track a task-space trajectory. Here, the task-space trajectory is given by the positions of the end-effector in Cartesian space. The tracking results in task-space for the 3-DoF robot and the Barrett WAM are shown in Figures 5.6 and 5.7, respectively. The figures show the tracking performance during the first 10 seconds.

In the experiment using the 3-DoF robot model shown in Figure 5.6, we compare the task-space tracking control performance, when employing the online learned model and the perfect analytical model. Using the perfect analytical model knowledge, the joint torques are computed as given in Equation (5.12). Thus, the robot performs perfect task-space tracking, as shown in Figure 5.6 (a). In this example, the rest posture is set to be $\boldsymbol{q}_{\text{rest}} = [-\pi/3, \pi/3, \pi/3]^T$. For the online learning of the task-space control model, the torque prediction is computed as given in Equation (5.21). The size of the local set is determined to be 30 and $\eta = 0.01$. Here, we employ a Gaussian kernel, where the kernel width $\mathbf{W}$ is optimized beforehand. As



(a) **task-space tracking performance** (b) **joint-space trajectory**

Figure 5.7: Task-space tracking control of a simulated 7-DoF Barrett WAM with online learned model. (a) During online model learning, the task-space controller is able to compute the required torques to follow the task-space trajectory. (b) joint-space trajectory during the online learning. Here, the rest posture is given by $\boldsymbol{q}_{\text{rest}} = [0.0, 0.5, 0.0, 1.9, 0.0, 0.0, 0.0]^T$.

shown in Figure 5.6 (b), the predicted joint torques converge to the perfect analytical torques after a short transient phase. As a result, the robot achieves good task-space tracking performance after a few seconds of online model learning.

In the next experiment, we employ the proposed approach to control the more complex 7-DoF Barrett WAM in simulation. Similar to the previous experiment, the robot is controlled to follow a figure-8 in task-space while learning the torque prediction model online. Here, the local set consists of 150 data points, $\eta = 0.05$ and a Gaussian kernel is used. During online learning, the model is incrementally updated 300 times. The results for the first 10 seconds are shown in Figure 5.7. It can be observed that the robot is able to follow the task-space trajectory well, while keeping the joint-space trajectory in the vicinity of the rest posture $\boldsymbol{q}_{\text{rest}}$.

5.2.4 Conclusion of Section 5.2

In this work, we employed local, kernel-based learning for the online approximation of a multi-valued mapping. This approach is based on the key insight that an approximation of such mappings from data is globally an ill-posed problem, while it is locally well-defined. Our proposed method uses an online procedure for updating the local model by inserting and removing data points. We further proposed a parametrization for the local model that allows learning task-space tracking control. As evaluation, we showed that the approach was able to learn torque prediction models for task-space tracking of redundant robots in several setups. The results show that the presented kernel-based approach can be used to approximate multi-valued mappings for task-space tracking control. Implementation on real robots is in progress. Here, practical issues need to be considered in more details, such as gain tuning and efficient implementation.

6 Conclusion

For model based robot control, model learning has proven to be helpful for controlling the systems compliantly and accurately. However, employing kernel-based regression techniques for real-time online model learning in control presents a challenge, due to the high computational complexity of such approaches. This thesis has made several contributions to the fields of robot control and kernel-based model learning by introducing novel approaches that enable fast, real-time online model learning for control. In this chapter, we summarize the results of the thesis. Subsequently, we discuss several open problems in model learning and provide an outlook on how these problems can be approached in the future.

6.1 Summary of the Thesis

To introduce the thesis, we motivate the need of model learning for control in Chapter 2. Here, we presented an extensive survey of model learning in robotics, with a focus on robot control. We first discussed different types of models and how these models can be incorporated in various learning architectures. Subsequently, we explained the problems that these architectures and the domain of robotics pose to learning methods. An overview of how models can be learned using machine learning techniques was given, with a focus on statistical regression methods. In several case studies, we further showed where the model learning scenarios have been used successfully.

While model learning is an appealing alternative to analytical modeling techniques, employing model learning using statistical learning methods for real-time applications is not straightforward. In Chapter 3, we presented a local approximation to the nonparametric Gaussian process regression, called local Gaussian process regression (LGP). The basic idea behind this approach is to partition the data space into local regions, for which independent local Gaussian models are learned. Learning and prediction can be sped up by using these local models, as the number of local data points is kept small. Thus, LGP combines the strength of fast computation, as in local regression, with potentially more accurate Bayesian regression methods. As a result, we obtained a real-time regression method that is applicable to model based robot control. The reduced complexity allowed the application of the LGP to online model learning. For model based tracking control, we used LGP to learn the inverse dynamics model online, while adapting the controller to unknown nonlinearities. Model based tracking control using online learned LGP models achieved a superior control performance for low gain control in comparison to rigid body models, as well as to offline learned models.

For the LGP algorithm, we employ the idea of localization and data partitioning to reduce the computational complexity. Another approach to reduce the computational complexity is to employ data sparsification. Here, the basic concept is to select only those points from the data stream that are informative for the model learning. In Chapter 4, we presented a framework for online, incremental sparsification designed for fast real-time model learning. The proposed sparsification employed a kernel independence measure to incrementally insert new data points into a sparse set and, eventually, remove old points from this set. This sparsification framework can be employed to speed up incremental learning methods for real-time model learning. For evaluation, we combined the incremental GPR with the proposed sparsification framework. In this way, we obtained a model approximation method which is applicable in real-time online learning. It exhibited competitive learning accuracy when compared with standard regression techniques. The approach was shown to be capable of online learning of inverse dynamics models. We further demonstrate the capability of the approach in online learning of robot's dynamics, which changes when different loads are applied.

In Chapter 5, we explored solutions to two model learning problems, i.e., learning models from small and potentially sparse data sets, and learning task-space tracking control. When only sparse and poor data is available, the learning performance can be improved by incorporating additional prior knowledge into the model learning process and, thus, semiparametric model learning can be used. In Section 5.1, we presented two possible semiparametric regression approaches, where the knowledge of the physical model can either become part of the mean function or of the kernel in a nonparametric Gaussian process regression. These methods are tested on sampled data in the setting of learning inverse dynamics models and, subsequently, used in tracking control on a Barrett WAM. In Section 5.2, we investigate the problem of learning task-space tracking control. Learning models for task-space tracking control from sampled data is an ill-posed problem. We formulate a local, kernel-based learning approach for online model learning for task-space tracking control. This approach was motivated by the insight that the model learning problem for task-space control is locally well-defined, while it is globally ill-posed as multiple solutions may form a non-convex data space.

6.2 Open Problems and Outlook

While this thesis presents encouraging results in bringing statistical learning methods to model based robot control, we envision several future improvements.

Analysis of Model Learning Control. While stability analysis of approximated model based control has been examined in the context of neural controllers (Patino *et al.*, 2002) and traditional adaptive controllers (Aström and Wittenmark, 1995; Narendra and Annaswamy, 1989), such investigations have been shown to be difficult when employing general nonparametric model approximation. Despite the pioneering work of Nakanishi et al. and Farrell et al. (Nakanishi *et al.*, 2005; Far-

rell and Polycarpou, 2006), approximated model based control using nonparametric models still suffers from a lack of proper analysis of stability and convergence properties. One major obstacle in analyzing such controllers is that the error bounds are often difficult to obtain. In (Nakanishi *et al.*, 2005), Nakanishi et al. employ locally weighted learning to approximate a class of nonlinear systems. Employing the linear local models, approximation error bounds can be estimated under the assumption that terms higher than quadratic order in a local approximation are negligible. Based on the estimated error bounds, conditions for stability of the controller can be given (Nakanishi *et al.*, 2005). As the structures of nonparametric kernel-based models are not fixed and may change with the data, deriving a consistent error bound is often difficult. Modern statistical learning theory might offer appropriate tools to estimate such error bounds. For example, generalization bounds (Schölkopf and Smola, 2002) can be used to estimate the learning and prediction performance of the controller and, thus, further statements about stability can be made. Recently, statistical bounds, such as PAC-Bayesian bounds, have been proposed to estimate the learning performance of general nonparametric models (McAllester, 1999; Seeger, 2005). PAC-Bayesian bounds give a worst case prediction bound for the learned models given the observation. This bound has proven to be very tight and has been used to analyze several nonparametric learning approaches, including support vector machines and Gaussian processes. Employing such statistical bounds can help to estimate an error bound for approximated model based controllers when using nonparametric kernel-based model learning.

Online Learning of Model's Hyperparameters. Hyperparameters are open parameters of a model that need to be optimized during the training process. For kernel-based learning approaches, these hyperparameters include the kernel widths which properly scale the input dimensions of a data point. Typically, these hyperparameters have to be optimized beforehand, such as by optimizing the marginal likelihood of a Gaussian process model (Rasmussen and Williams, 2006) or by cross-validation when using support vector machines (Schölkopf and Smola, 2002). The hyperparameters are usually estimated offline using pre-sampled data, as shown in preceding chapters. As the optimal hyperparameters may vary depending on the state-space regions, it is necessary to estimate the locations where the data might be sampled during operation. Based on this prior knowledge, optimal hyperparameters can be inferred for that state-space region. However, the robot may move out of these pre-estimated state-space regions during online learning. In that case, the offline inferred hyperparameters might be suboptimal. Thus, online adaptation of these parameters might be useful. Estimating hyperparameters is computationally expensive in many cases. Hence, approaches for efficient online hyperparameter estimation and adaptation will be needed to cope with the real-time requirements. One promising approach would be to employ online optimization techniques, such as gradient descent methods, to incrementally adapt the hyperparameters depending on the observed data. Another direction would be to learn a model for predicting the optimal hyperparameters given the current state space locations. Here, one

could generate samples of locally optimal hyperparameters for different state-space regions and, subsequently, use them for learning the model with supervised learning methods. As a result, optimal hyperparameters values could be obtained for larger state-space regions.

Learning Models for Control under Task Constraints. Controlling a robot under multiple task constraints is a general problem. For example, this control paradigm can be used for designing humanoid robot controllers (Sentis and Khatib, 2005). These systems are difficult to control, as they need to coordinate many low-level behaviors to accomplish complex tasks while being responsive to the changes in the environment. Control under multiple task constraints is an extension of the operational space control problem (discussed in Chapter 5.2) to a hierarchical structure (Peters *et al.*, 2008). In order to obtain an approximated model based controller under multiple task constraints, the multiple task-relevant controllers have to be prioritized in a hierarchical control structure. Here, less prioritized task controllers will act in the null-space of the controllers for higher prioritized tasks (Peters *et al.*, 2008). Similar to the case of learning operational space control, learning models for control under multiple task constraints is an ill-posed problem. A method for learning ill-posed models using kernels was introduced in Chapter 5.2. Another idea that can be used to solve this problem is to employ conditional random fields (Lafferty *et al.*, 2001). In general, the problem of learning multi-valued mappings can be considered as a non-unique labeling problem, i.e., many input points may have the same target output. The conditional random fields framework builds probabilistic models to segment and label sequences of data. A conditional model specifies the probabilities of possible target outputs given an observation sequence of past input and targets. As the target outputs are conditioned on the current observations, non-uniqueness in the mappings can be resolved. Therefore, conditional random fields could be employed to learn such ill-posed mappings necessary for learning operational space control models and its hierarchical extension.

Model Transfer Learning. In Chapter 4, we learn inverse dynamics models for robot tracking control, while the end-effector load is changing over time. When the load changes, the inverse dynamics model has to be adapted and, thus, the robot potentially discards previously obtained knowledge. Learning inverse dynamics models for different loads can thus be understood as learning different dynamics models. It would be beneficial to investigate how previously obtained model knowledge can be used to accelerate the current model learning process. Recent research on learning robot control has predominantly focussed on learning single models that were studied in isolation. However, there is an opportunity to transfer knowledge between models, which is known as transfer learning (Ben-David and Schuller, 2003). To achieve this goal, robots need to learn the invariants of the individual model and, subsequently, exploit them when learning new tasks. One way to achieve such transfer of knowledge is to combine model learning with dimensionality reduction (Pan *et al.*, 2008). Here, the idea would be to learn a latent space which is common for different models. The insight is that, if the models are related to each

other, there exist several common latent variables that dominate the observed data. Subsequently, the data can be projected onto this latent space, where models can be learned. Such model-independent knowledge, e.g., the latent variables, can be employed to make the system more efficient when learning a new model. In this context, similarities between models also need to be investigated and how they can be employed to generalize to new models. It may be useful to continue studying the underlying structures between models which can help to accelerate model learning (Tsochantaridis *et al.*, 2005).

Semi-supervised Model Learning. In this thesis, we focused on supervised learning methods, as sufficiently many labeled data can be generated in our learning setting. However, target outputs, i.e., labeled data points, are not always available for many robot applications. One example is learning models for terrain classification using vision features. In that case, exact labeling of the vision features is not always possible and, furthermore, manually labeling such features is expensive and susceptive to errors. Here, semi-supervised learning techniques can be useful to learn such models (Chapelle *et al.*, 2006). Semi-supervised learning employs labeled as well as unlabeled data for model learning and can help to overcome the sparse labeling problem. While semi-supervised learning is well-studied in the field of machine learning, applications of semi-supervised model learning in robotics is still limited. It would also be beneficial to develop online versions of semi-supervised approaches for real-time adaptation and learning.

6.3 Publications

Excerpts of the research presented in this thesis have led to the following articles. The results have received attention both in the field of machine learning and in robotics.

Journal Publications

D. Nguyen-Tuong, M. Seeger and J. Peters. Model Learning with Local Gaussian Process Regression. *Advanced Robotics* 23(15), pages 2015–2034, 2009.

D. Nguyen-Tuong and J. Peters. Incremental Sparsification for Real-time Online Model Learning. *Neurocomputing*, 2010. In press.

D. Nguyen-Tuong and J. Peters. Model Learning for Robotics: A Survey. *Cognitive Systems*, 2011. Submitted.

Conference Publications

D. Nguyen-Tuong, J. Peters, M. Seeger and B. Schölkopf. Learning Inverse Dynamics: A Comparison. *Proceedings of the European Symposium on Artificial Neural Networks*

(ESANN), pages 13–18, 2008.

J. Peters and D. Nguyen-Tuong. Real-time Learning of Resolved Velocity Control on a Mitsubishi PA-10. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2872–2877, 2008.

D. Nguyen-Tuong and M. Seeger and J. Peters. Computed Torque Control with Nonparametric Regression Models. *Proceedings of the American Control Conference (ACC)*, pages 212–217, 2008.

D. Nguyen-Tuong and J. Peters. Local Gaussian Processes Regression for Real-time Model-based Robot Control. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 380–385, 2008.

D. Nguyen-Tuong, M. Seeger and J. Peters: Local Gaussian Process Regression for Real-time Online Model Learning and Control. *Advances in Neural Information Processing Systems 21 (NIPS)*, pages 1193–1200, 2009.

D. Nguyen-Tuong, B. Schölkopf and J. Peters. Sparse Online Model Learning for Robot Control with Support Vector Regression. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3121–3126, 2009.

J. Peters, K. Mülling, J. Kober, D. Nguyen-Tuong and O. Kroemer. Towards Motor Skill Learning for Robotics. *Proceedings of the 14th International Symposium on Robotics Research (ISRR)*, pages 1–14, 2010.

D. Nguyen-Tuong and J. Peters. Incremental Sparsification for Real-time Online Model Learning. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 557–564, 2010.

D. Nguyen-Tuong and J. Peters. Using Model Knowledge for Learning Inverse Dynamics. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2677–2682, 2010.

Roberto Lampariello, Duy Nguyen-Tuong, Claudio Castellini, Gerd Hirzinger and Jan Peters. Energy-Optimal Robot Catching in Real-time. *IEEE International Conference on Robotics and Automation (ICRA)*, 2011. Accepted.

D. Nguyen-Tuong and J. Peters. Learning Task-Space Tracking Control with Kernels. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011. Under Review.

Bibliography

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

Akaike, H. (1970). Autoregressive model fitting for control. *Annals of the Institute of Statistical Mathematics*, **23**, 163–180.

Akesson, B. M. and Toivonen, H. T. (2006). A neural network model predictive controller. *Journal of Process Control*, **16**(9), 937–946.

Angelova, A., Matthies, L., Helmick, D., and Perona, P. (2006). Slip prediction using visual information. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA.

Aström, K. J. and Wittenmark, B. (1995). *Adaptive Control*. Addison Wesley, Boston, MA, USA.

Atkeson, C. G. and Morimoto, J. (2002). Nonparametric representation of policies and value functions: A trajectory-based approach. *Advances in Neural Information Processing Systems*.

Atkeson, C. G. and Schaal, S. (1997). Robot learning from demonstration. In *Proceedings of the 14-th International Conference on Machine Learning*.

Atkeson, C. G., An, C. H., and Hollerbach, J. M. (1986). Estimation of inertial parameters of manipulator loads and links. *International Journal of Robotics Research*, **5**(3).

Atkeson, C. G., Moore, A. W., and Schaal, S. (1997a). Locally weighted learning. *Artificial Intelligence Review*, **11**(1–5), 11–73.

Atkeson, C. G., Moore, A. W., and Schaal, S. (1997b). Locally weighted learning for control. *Artificial Intelligence Review*, **11**(1-5), 75–113.

Ben-David, S. and Schuller, R. (2003). Exploiting task relatedness for multiple task learning. In *Proceedings of the Conference on Learning Theory*.

Bhushan, N. and Shadmehr, R. (1999). Evidence for a forward dynamics model in human adaptive motor control. *Advances in Neural Information Processing Systems*.

Billings, S. S., Chen, S., and Korenberg, G. (1989). Identification of mimo nonlinear systems using a forward-regression orthogonal estimator. *International Journal of Control*, **49**, 2157–2189.

Bottou, L., Chapelle, O., DeCoste, D., and Weston, J. (2007). *Large-Scale Kernel Machines*. MIT Press, Cambridge, MA.

Burdet, E. and Codourey, A. (1998). Evaluation of parametric and nonparametric nonlinear adaptive controllers. *Robotica*, **16**(1), 59–73.

Burdet, E., Sprenger, B., and Codourey, A. (1997). Experiments in nonlinear adaptive control. *International Conference on Robotics and Automation*, **1**, 537–542.

Calinon, S., D'halluin, F., Sauser, E., Caldwell, D., and Billard, A. (2010). A probabilistic approach based on dynamical systems to learn and reproduce gestures by imitation. *IEEE Robotics and Automation Magazine*, **17**, 44–54.

Candela, J. Q. and Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*.

Candela, J. Q. and Winther, O. (2003). Incremental gaussian processes. *Advances in Neural Information Processing Systems*.

Cao, H., Yin, Y., Du, D., Lin, L., Gu, W., and Yang, Z. (2006). Neural network inverse dynamic online learning control on physical exoskeleton. In *13th International Conference on Neural Information Processing*.

Cauwenberghs, G. and Poggio, T. (2000). Incremental and decremental support vector machine learning. *Advances in Neural Information Processing Systems*.

Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*.

Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-Supervised Learning*. MIT Press, Cambridge, MA.

Choi, Y., Cheong, S. Y., and Schweighofer, N. (2007). Local online support vector regression for learning control. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*.

Chow, C. M., Kuznetsov, A. G., and Clarke, D. W. (1998). Successive one-step-ahead predictions in multiple model predictive control. *International Journal of Control*, **29**, 971–979.

Cleveland, W. S. and Loader, C. L. (1996). Smoothing by local regression: Principles and methods. In *Statistical Theory and Computational Aspects of Smoothing*.

Coito, F. J. and Lemos, J. M. (1991). A long-range adaptive controller for robot manipulators. *The International Journal of Robotics Research*, **10**, 684–707.

Craig, J. J. (2004). *Introduction to Robotics: Mechanics and Control*. Prentice Hall, New Jersey.

Csato, L. and Opper, M. (2002). Sparse online gaussian processes. *Neural Computation*.

D'Souza, A., Vijayakumar, S., and Schaal, S. (2001). Learning inverse kinematics. In *IEEE International Conference on Intelligent Robots and Systems*.

Edakunni, N. U., Schaal, S., and Vijayakumar, S. (2007). Kernel carpentry for online regression using randomly varying coefficient model. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*.

Engel, Y., Mannor, S., and Meir, R. (2002). Sparse online greedy support vector regression. *European Conference on Machine Learning*.

Engel, Y., Mannor, S., and Meir, R. (2004). The kernel recursive least-square algorithm. *IEEE Transaction on signal processing*, **52**.

Fan, J. and Gijbels, I. (1996). *Local Polynomial Modelling and Its Applications*. Chapman and Hall.

Farrell, J. A. and Polycarpou, M. M. (2006). *Adaptive Approximation Based Control*. John Wiley and Sons, New Jersey.

Ferreira, J. P., Crisostomo, M., Coimbra, A. P., and Ribeiro, B. (2007). Simulation control of a biped robot with support vector regression. In *IEEE International Symposium on Intelligent Signal Processing*.

Ge, S. S., Lee, T. H., and Tan, E. G. (1998). Adaptive neural network control of flexible joint robots based on feedback linearization. *International Journal of Systems Science*, **29**(6), 623–635.

Genov, R., Chakrabartty, S., and Cauwenberghs, G. (2003). Silicon support vector machine with online learning. *International Journal of Pattern Recognition and ArtiÞcial Intelligence*, **17**, 385–404.

Girard, A., Rasmussen, C. E., Candela, J. Q., and Smith, R. M. (2002). Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. *Advances in Neural Information Processing Systems*.

Glynn, P. W. (1987). Likelihood ratio gradient estimation: an overview. In *Proceedings of the 1987 Winter Simulation Conference*.

Gomi, H. and Kawato, M. (1993). Recognition of manipulated objects by motor learning with modular architecture networks. *Neural Networks*, **6**(4), 485–497.

Grollman, D. H. and Jenkins, O. C. (2008). Sparse incremental learning for interactive robot control policy estimation. In *IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA.

Gu, D. and Hu, H. (2002). Predictive control for a car-like mobile robot. *Robotics and Autonomous Systems*, **39**, 73–86.

Haerdle, W. K., Mueller, M., Sperlich, S., and Werwatz, A. (2004). *Nonparametric and Semiparametric Models*. Springer, New York, USA.

Haruno, M., Wolpert, D. M., and Kawato, M. (2001). Mosaic model for sensorimotor learning and control. *Neural Computation*, **13**(10), 2201–2220.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer, New York.

Hawes, N., Wyatt, J. L., Sridharan, M., Kopicki, M., Hongeng, S., Calvert, I., Sloman, A., Kruijff, G.-J., Jacobsson, H., Brenner, M., Skočaj, D., Vrečko, A., Majer, N., and Zillich, M. (2010). The playmate system. *Cognitive Systems*, **8**, 367–393.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall, New Jersey.

Hoffman, H., Schaal, S., and Vijayakumar, S. (2009). Local dimensionality reduction for non-parametric regression. *Neural Processing Letters*.

Jacobs, R., Jordan, M., Nowlan, S., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, **3**, 79–87.

Jacobson, D. H. and Mayne, D. Q. (1973). *Differential Dynamic Programming*. New York, American Elsevier.

J.Fan and I.Gijbels (1995). Data driven bandwidth selection in local polynomial fitting. *Journal of the Royal Statistical Society*, **57**(2), 371–394.

Jordan, I. and Rumelhart, D. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, **16**, 307–354.

Kalakrishnan, M., Buchli, J., Pastor, P., and Schaal, S. (2009). learning locomotion over rough terrain using terrain templates. In *IEEE International Conference on Intelligent Robots and Systems*.

Kawato, M. (1990). Feedback error learning neural network for supervised motor learning. *Advanced Neural Computers*.

Kawato, M. (1999). Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, **9**(6), 718–727.

Keyser, R. D. and Cauwenberghe, A. V. (1980). A self-tuning multistep predictor application. *Automatica*, **17**, 167–174.

Khalil, W. and Dombre, E. (2002). *Modeling, Identification and Control of Robots*. Taylor & Francis, Inc., Bristol, PA, USA.

Khatib, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *Journal of Robotics and Automation*, **3**(1), 43–53.

Klanke, S., Lebedev, D., Haschke, R., Steil, J. J., and Ritter, H. (2006). Dynamic path planning for a 7-dof robot arm. In *Proceedings of the 2009 IEEE International Conference on Intelligent Robots and Systems*.

Ko, J. and Fox, D. (2009). GP-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, **27**(1), 75–90.

Kocijan, J., Murray-Smith, R., Rasmussen, C., and Girard, A. (2004). Gaussian process model based predictive control. *Proceeding of the American Control Conference*.

Kröse, B. J., Vlassis, N., Bunschoten, R., and Motomura, Y. (2001). A probabilistic model for appearance-based robot localization. *Image and Vision Computing*, **19**, 381–391.

Krupka, E. and Tishby, N. (2007). Incorporating prior knowledge on features into learning. In *International Conference on Artificial Intelligence and Statistics*, San Juan, Puerto Rico.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*.

Lang, T., Plagemann, C., and Burgard, W. (2007). Adaptive non-stationary kernel regression for terrain modeling. *Robotics: Science and Systems (RSS)*.

Layne, J. R. and Passino, K. M. (1996). Fuzzy model reference learning control. *Journal of Intelligent and Fuzzy Systems*, **4**, 33–47,.

Li, W. (1990). *Adaptive Control of Robot Motion*. Ph.D. thesis, Massachusetts Institute of Technology (MIT).

Liu, Y., Wang, H., Yu, J., and Lia, P. (2009). Selective recursive kernel learning for online identification of nonlinear systems with NARX form. *Journal of Process Control*, pages 181–194.

Ljung, L. (2004). *System Identification - Theory for the User*. Prentice-Hall, New Jersey.

Luca, A. D. and Lucibello, P. (1998). A general algorithm for dynamic feedback linearization of robots with elastic joints. In *Proceedings of the IEEE Intemational Conference on Robotics and Automation*.

Ma, J., Theiler, J., and Perkins, S. (2005). Accurate on-line support vector regression. *Neural Computation*, **15**, 2683–2703.

Maciejowski, J. M. (2002). *Predictive Control with Constraints*. Prentice Hall, New Jersey.

MacKay, D. J. (1992). A practical Bayesian framework for back-propagation networks. *Neural Computation*, **4**(3), 448–472.

McAllester, D. (1999). Some PAC-Bayesian theorems. *Machine Learning*, **37**.

Miyamoto, H., Kawato, M., Setoyama, T., and Suzuki, R. (1988). Feedback-error-learning neural network for trajectory control of a robotic manipulator. *Neural Networks*, **1**(3), 251–265.

Moore, A. (1992). Fast, robust adaptive control by learning only forward models. In *Advances in Neural Information Processing Systems*.

Moore, A. and Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. In *Proceedings of the 11th International Confonference on Machine Learning*.

Morimoto, J., Zeglin, G., and Atkeson, C. G. (2003). Minimax differential dynamic programming: Application to a biped walking robot. In *Proceedings of the 2009 IEEE International Conference on Intelligent Robots and Systems*.

Mosca, E., Zappa, G., and Lemos, J. M. (1989). Robustness of multipredictor adaptive regulators: MUSMAR. *Automatica*, **25**, 521–529.

Nakanishi, J. and Schaal, S. (2004). Feedback error learning and nonlinear adaptive control. *Neural Networks*, **17**(10).

Nakanishi, J., Farrell, J. A., and Schaal, S. (2005). Composite adaptive control with locally weighted statistical learning. *Neural Networks*, **18**(1), 71–90.

Nakanishi, J., Cory, R., Mistry, M., Peters, J., and Schaal, S. (2008). Operational space control: a theoretical and emprical comparison. *International Journal of Robotics Research*, **27**(6), 737–757.

Nakayama, H., Yun, Y., and Shirakawa, M. (2008). Multi-objective model predictive control. In *Proceedings of the 19th International Conference on Multiple Criteria Decision Making*.

Narendra, K. and Balakrishnan, J. (1997). Adaptive control using multiple models. *IEEE Transaction on Automatic Control*, **42**(2), 171–187.

Narendra, K., Balakrishnan, J., and Ciliz, M. (1995). Adaptation and learning using multiple models, switching and tuning. *IEEE Control System Magazin*, **15**(3), 37–51.

Narendra, K. S. and Annaswamy, A. M. (1987). Persistent excitation in adaptive systems. *International Journal of Control*, **45**, 127–160.

Narendra, K. S. and Annaswamy, A. M. (1989). *Stable Adaptive Systems*. Prentice Hall, New Jersey.

Neal, R. M. (1996). Bayesian learning for networks. *Lecture Notes in Statistics*.

Negenborn, R., Schutter, B. D., Wiering, M. A., and Hellendoorn, H. (2005). Learning-based model predictive control for markov decision processes. In *Proceedings of the 16th IFAC World Congress*.

Ng, A. Y. and Jordan, M. (2000). Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence*.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2004). Autonomous inverted helicopter flight via reinforcement learning. In *Proceedings of the 11th International Symposium on Experimental Robotics*.

Nguyen-Tuong, D. and Peters, J. (2009). Model learning with local gaussian process regression. *Advanced Robotics*, **23**(15), 2015–2034.

Nguyen-Tuong, D. and Peters, J. (2010a). Incremental sparsification for real-time online model learning. In *Proceedings of the 13th International Conference on ArtiÞcial Intelligence and Statistics*.

Nguyen-Tuong, D. and Peters, J. (2010b). Incremental sparsification for real-time online model learning. (in press). *Neurocomputing*.

Nguyen-Tuong, D. and Peters, J. (2010c). Using model knowledge for learning inverse dynamics. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*.

Nguyen-Tuong, D., Peters, J., and Seeger, M. (2008a). Computed torque control with nonparametric regression models. *Proceedings of the 2008 American Control Conference (ACC 2008)*.

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2008b). Local Gaussian process regression for real time online model learning and control. *Advances in Neural Information Processing Systems*.

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2008c). Local gaussian processes regression for real-time model-based robot control. In *International Conference on Intelligent Robots and Systems*, Nice, France.

Nguyen-Tuong, D., Schoelkopf, B., and Peters, J. (2009). Sparse online model learning for robot control with support vector regression. In *Proceedings of the 2009 IEEE International Conference on Intelligent Robots and Systems*.

Nicosia, S. and Tomei, P. (1984). Model reference adaptive control algorithms for industrial robots. *Automatica*, **20**, 635–644.

Nowlan, S. and Hinton, G. E. (1991). Evaluation of adaptive mixtures of competing experts. *Advances in Neural Information Processing Systems*.

Pan, S. J., Kwok, J. T., and Yang, Q. (2008). Transfer learning via dimensionality reduction. In *Association for the Advancement of Artificial Intelligence*.

Patino, H. D., Carelli, R., and Kuchen, B. R. (2002). Neural networks for advanced control of robot manipulators. *IEEE Transactions on Neural Networks*, **13**(2), 343–354.

Pelossof, R., Miller, A., Allen, P., and Jebara, T. (2004). An svm learning approach to robotic grasping. In *In IEEE International Conference on Robotics and Automation*.

Peters, J. and Schaal, S. (2008). Learning to control in operational space. *International Journal of Robotics Research*, **27**(2), 197–212.

Peters, J., Mistry, M., Udwadia, F. E., Nakanishi, J., and Schaal, S. (2008). A unifying methodology for robot control with redundant dofs. *Autonomous Robots*, **24**(1), 1–12.

Plagemann, C., Kersting, K., Pfaff, P., and Burgard, W. (2007). Heteroscedastic gaussian process regression for modeling range sensors in mobile robotics. *Snowbird learning workshop*.

Plagemann, C., Mischke, S., Prentice, S., Kersting, K., Roy, N., and Burgard, W. (2008). Learning predictive terrain models for legged robot locomotion. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*.

Rasmussen, C. E. (1996). Evaluation of gaussian processes and other methods for non-linear regression. *University of Toronto*.

Rasmussen, C. E. and Ghahramani, Z. (2002). Infinite mixtures of gaussian process experts. *Advances in Neural Information Processing Systems*.

Rasmussen, C. E. and Kuss, M. (2003). Gaussian processes in reinforcement learning. *Advances in Neural Information Processing Systems*.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. MIT-Press, Massachusetts Institute of Technology.

Reinhart, F. R. and Steil, J. J. (2008). Recurrent neural associative learning of forward and inverse kinematics for movement generation of the redundant pa-10 robot. In *Symposium on Learning and Adaptive Behavior in Robotic Systems*.

Reinhart, F. R. and Steil, J. J. (2009). Attractor-based computation with reservoirs for online learning of inverse kinematics. In *Proceedings of the European Symposium on Artificial Neural Networks*.

Rottmann, A. and Burgard, W. (2009). Adaptive autonomous control using online value iteration with gaussian processes. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Roweis, S. and Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, **290**.

Salaun, C., Padois, V., and Sigaud, O. (2009). Control of redundant robots using learned models: an operational space control approach. In *Proceedings of the 2009 IEEE International Conference on Intelligent Robots and Systems*.

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*.

Schaal, S. (2006). The SL simulation and real-time control software package. Technical report, university of southern california.

Schaal., S. and Atkeson, C. G. (1996). From isolation to cooperation: An alternative of a system of experts. *Advances in Neural Information Processing Systems*.

Schaal, S. and Atkeson, C. G. (2010). Learning control in robotics: Trajectory-based optimal control techniques. *IEEE Robotics and Automation Magazine*.

Schaal, S. and Sternad, D. (1998). Programmable pattern generators. In *International Conference on Computational Intelligence in Neuroscience*.

Schaal, S., Atkeson, C. G., and Vijayakumar, S. (2000). Real-time robot learning with locally weighted statistical learning. *International Conference on Robotics and Automation*.

Schaal, S., Atkeson, C. G., and Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics for real-time robot learning. *Applied Intelligence*, **17**(1), 49–60.

Schölkopf, B. and Smola, A. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT-Press, Cambridge, MA.

Schölkopf, B., Simard, P., Smola, A., and Vapnik, V. (1997). Prior knowledge in support vector kernel. In *Advances in Neural Information Processing Systems*, Denver,CO, USA.

Schölkopf, B., Mika, S., Burges, C. J. C., Knirsch, P., Müller, K.-R., Rätsch, G., and Smola, A. J. (1999). Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, **10**(5), 1000–1017.

Schölkopf, B., Smola, A., Williamson, R., and Bartlett, P. (2000). New support vector algorithms. *Neural Computation*.

Sciavicco, L. and Siciliano, B. (1996). *Modeling and Control of Robot Manipulators*. McGraw-Hill, New York.

Seeger, M. (2004). Gaussian processes for machine learning. *International Journal of Systems*.

Seeger, M. (2005). *Bayesian Gaussian Process Models: PAC-Bayesian Generalisation Error Bounds and Sparse Approximations*. Ph.D. thesis, University of Edinburgh.

Seeger, M. (2007a). *LHOTSE: Toolbox for Adaptive Statistical Model*.

Seeger, M. (2007b). Low rank update for the cholesky decomposition. Technical report, University of California at Berkeley.

Sentis, L. and Khatib, O. (2005). Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, **2**(4), 505–518.

Shen, Y., Ng, A. Y., and Seeger, M. (2005). Fast gaussian process regression using kd-trees. *Advances in Neural Information Processing Systems*.

Shibata, T. and Schaal, C. (2001). Biomimetic gaze stabilization based on feedback-error learning with nonparametric regression networks. *Neural Networks*, **14**(2), 201–216.

Skočaj, D., Kristan, M., Vrečko, A., Leonardis, A., Fritz, M., Stark, M., Schiele, B., Hongeng, S., and Wyatt, J. L. (2010). Multi-modal learning. *Cognitive Systems*, **8**, 265–309.

Slotine, J.-J. E. and Li, W. (1991). *Applied Nonlinear Control*. Prentice Hall, New Jersey.

Smith, O. J. (1959). A controller to overcome dead-time. *Instrument Society of America Journal*, **6**, 28–33.

Smola, A., Friess, T., and Schoelkopf, B. (1998). Semiparametric support vector and linear programming machines. In *Advances in Neural Information Processing Systems*, Denver,CO, USA.

Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, **14**(3), 199–222.

Snelson, E. and Ghahramani, Z. (2007). Local and global sparse gaussian process approximations. *Artificial Intelligence and Statistics*.

Spong, M. W., Hutchinson, S., and Vidyasagar, M. (2006). *Robot Dynamics and Control*. John Wiley and Sons, New York.

Steffen, J., Klanke, S., Vijayakumar, S., and Ritter, H. J. (2009). Realising dextrous manipulation with structured manifolds using unsupervised kernel regression with structural hints. In *ICRA 2009 Workshop: Approaches to Sensorimotor Learning on Humanoid Robots*, Kobe, Japan.

Steil, J. J. (2004). Backpropagation-decorrelation: online recurrent learning with o(n) complexity. In *Proceedings of the International Joint Conference on Neural Networks*.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, **2**(4), 160–163.

Suttorp, T. and Igel, C. (2008). Approximation of gaussian process models after training. *European Symposium on Artificial Networks*.

Swevers, J., Ganseman, C., Tükel, D., Schutter, J. D., and Brussel, H. V. (1997). Optimal robot excitation and identification. *IEEE Transaction on Robotics and Automation*, **13**, 730–740.

Tenenbaum, J., de Silva, V., and Langford, J. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, **290**.

Tevatia, G. and Schaal, S. (2008). Efficient inverse kinematics algorithms for high-dimensional movement systems. *University of Southern California*.

Thrun, S. and Mitchell, T. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*.

Ting, J., Kalakrishnan, M., Vijayakumar, S., and Schaal, S. (2008). Bayesian kernel shaping for learning control. *Advances in Neural Information Processing Systems*.

Ting, J., D'Souza, A., and Schaal, S. (2009). A bayesian approach to nonlinear parameter identification for rigid-body dynamics. *Neural Networks*.

Titsias, M. K. and Lawrence, N. D. (2010). Bayesian gaussian process latent variable model. In *Proceedings of the 13th International Conference on ArtiÞcial Intelligence and Statistics*.

Toussaint, M. and Vijayakumar, S. (2005). Learning discontinuities with products-of-sigmoids for switching between local models. In *Proceedings of the 22nd International Conference on Machine Learning*.

Townsend, W. (2007). *Inertial Data for the Whole-Arm-Manipulator (WAM) Arm*.

Treps, V. (2000). A bayesian committee machine. *Neural Computation*, **12**(11), 2719 – 2741.

Treps, V. (2001). Mixtures of gaussian process. *Advances in Neural Information Processing Systems*.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, **6**, 1453–1484.

Urtasun, R. and Darrell, T. (2008). Sparse probabilistic regression for activity-independent human pose inference. In *International Conference in Computer Vision and Pattern Recognition*, Anchorage, Alaska.

Vempaty, P., Cheok, K., and Loh, R. (2009). Model reference adaptive control for actuators of a biped robot locomotion. In *Proceedings of the World Congress on Engineering and Computer Science*.

Vijayakumar, S. and Schaal, S. (2000). Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional space. *International Conference on Machine Learning, Proceedings of the Sixteenth Conference*.

Vijayakumar, S. and Wu, S. (1999). Sequential support vector classifiers and regression. *International Conference on Soft Computing*.

Vijayakumar, S., D'Souza, A., and Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation*, **12**(11), 2602 – 2634.

Wan, E. A. and Bogdanov, A. A. (2001). Model predictive neural control with applications to a 6 dof helicopter model. In *Proceedings of the 2001 American Control Conference*.

Wolpert, D. M. and Kawato, M. (1998). Multiple paired forward and inverse models for motor control. *Neural Networks*, **11**, 1317–1329.

Wolpert, D. M., Miall, R. C., and Kawato, M. (1998). Internal models in the cerebellum. *Trends in Cognitive Sciences*, **2**.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software*.