Learning Dynamics for Robot Control under Varying Contexts

Georgios Petkos

Doctor of Philosophy
Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh
2008

Abstract

High fidelity, compliant robot control requires a sufficiently accurate dynamics model. Often though, it is not possible to obtain a dynamics model sufficiently accurately or at all using analytical methods. In such cases, an alternative is to learn the dynamics model from movement data. This thesis discusses the problems specific to dynamics learning for control under nonstationarity of the dynamics.

We refer to the cause of the nonstationarity as the context of the dynamics. Contexts are, typically, not directly observable. For instance, the dynamics of a robot manipulator changes as the robot manipulates different objects and the physical properties of the load – the context of the dynamics – are not directly known by the controller. Other examples of contexts that affect the dynamics are changing force fields or liquids with different viscosity in which a manipulator has to operate.

The learned dynamics model needs to be adapted whenever the context and therefore the dynamics changes. Inevitably, performance drops during the period of adaptation. The goal of this work, is to reuse and generalize the experience obtained by learning the dynamics of different contexts in order to adapt to changing contexts fast.

We first examine the case that the dynamics may switch between a discrete, finite set of contexts and use multiple models and switching between them to adapt the controller fast. A probabilistic formulation of multiple models is used, where a *discrete latent variable* is used to represent the unobserved context and index the models. In comparison to previous multiple model approaches, the developed method is able to learn multiple models of nonlinear dynamics, using an appropriately modified EM algorithm.

We also deal with the case when there exists a continuum of possible contexts that affect the dynamics and hence, it becomes essential to generalize from a set of experienced contexts to novel contexts. There is very little previous work on this direction and the developed methods are completely novel. We introduce a set of *continuous latent variables* to represent context and introduce a dynamics model that depends on this set of variables. We first examine learning and inference in such a model when there is strong prior knowledge on the relationship of these continuous latent variables to the modulation of the dynamics, e.g., when the load at the end effector changes. We also develop methods for the case that there is no such knowledge available.

Finally, we formulate a dynamics model whose input is *augmented with observed variables* that convey contextual information indirectly, e.g., the information from tactile sensors at the interface between the load and the arm. This approach also allows

generalization to not previously seen contexts and is applicable when the nature of the context is not known. In addition, we show that use of such a model is possible even when special sensory input is not available by using an instance of an autoregressive model.

The developed methods are tested on realistic, full physics simulations of robot arm systems including a simplistic 3 degree of freedom (DOF) arm and a simulation of the 7 DOF DLR light weight robot arm. In the experiments, varying contexts are different manipulated objects. Nevertheless, the developed methods (with the exception of the methods that require prior knowledge on the relationship of the context to the modulation of the dynamics) are more generally applicable and could be used to deal with different context variation scenarios.

Acknowledgements

First of all, I cannot find the words to express my gratitude and love to my family. Vasiliki and Ioanna, thank you for being next to me and helping me in any way that was needed.

I wish to thank my supervisor Dr. Sethu Vijayakumar for his support, patience and knowledgeable advice. His guidance and understanding during the most difficult periods of the very stressful process of the Ph.D. were very important to keep me focused and motivated.

I am grateful to the Greek State Scholarships Foundation (IKY) for funding my studies.

The help and support of Marc Toussaint has been invaluable. Apart from help with technical issues, he was a source of inspiration and encouragement. Heiko Hoffman and Stefan Klanke have been very supportive and provided much valued advice.

Furthermore, I want to thank my friends Stavros Parlalis, Giorgos Machtsiras, Evia Kainada, Angelos Lengeris, Claire Kydonaki, Evi Korakaki, Dimitris Klaroudas, Christophoros Christophoridis, Giorgos Delinikolas, Nikos Xanthopoulos, Ilias Koutis.

A big thanks also goes to my lab mates Narayanan Edakunni, Timothy Hospedales, Djordje Mitrovic, Matthew Howard, Adrian Haith, Sebastian Bitzer, Graham McNeill.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Georgios Petkos*)

To my mother Vasiliki

Table of Contents

List of Figures

Chapter 1

Introduction

Movement execution is a crucial skill in both biological (e.g. humans) and artificial (e.g. robots) embodied systems. The importance of the ability to generate movement is highlighted by the fact that movement is essentially the only means that we have to interact with the world (Wolpert et al., 2001). Humans and other biological systems exhibit high levels of robustness and flexibility in controlling their motor system under varying conditions. It is the holy grail of robotics to achieve similar levels of adaptability.

One of the key ingredients in fast, adaptive motor control is the ability to predict the consequences of one's actions and then adapt to novel or unexperienced dynamics. This work focuses on developing such ability in complex, anthropomorphic robotic manipulators. In particular, it examines the problem of *dynamics learning* for *control* of robot manipulators under *nonstationary* conditions. Three concepts are central to this study: *control*, *dynamics learning* and *nonstationarity*. While we defer in depth discussion of these topics till later, here, we attempt to briefly introduce these topics and motivate the key questions and contributions of this work.

Control, in the context of movement execution, can be defined as the problem of transforming a desired movement task into actual movement. This involves sending the appropriate signals to the actuators of the system (typically electric or hydraulic motors for robots) in order to realize the desired movement. Predictive control methods that can achieve fast and compliant movement require an accurate dynamics model of the system. In general, prior knowledge of the structure and the physical properties of the robot can be used to analytically derive a dynamic model. Nevertheless, in many cases, accurate analytical modeling of the dynamics may not be feasible. Possible reasons could be the complexity of the dynamics, i.e. the existence of unmodeled effects (e.g.

friction can be very hard to model) or inaccurate knowledge of some parameters or even complete lack of knowledge of the structure of the dynamics or the parameters. If acquiring an accurate dynamics model of the system is not possible analytically, then an alternative is to *learn* it, i.e. approach it using sensed movement data. Learning dynamics models for control is a much studied field (Baker and Farrell, 1992; Hunt et al., 1992; Atkeson et al., 1997; Vijayakumar et al., 2002).

This work focuses on the problems that *nonstationarity* of dynamics poses to dynamics learning for control. Nonstationarity can be caused either by interaction with different environments or by internal changes to the system itself. An example of interaction with a varying environment for a robot manipulator could be manipulation of different loads: different loads change the dynamics of the manipulator. An example of nonstationarity caused by an internal change to the system could be change of friction due to wear and tear or failure of parts. The cause of nonstationarity will be referred to as the *context* of the dynamics, i.e. the context is a varying factor that modulates the dynamics. In general, *the context is not directly observed*; only the effect of the context on the dynamics or some affected sensory input is observed. For example, the physical properties of the load are not known, however, the observed behaviour of the robot conveys information about the load. Other examples of unobserved contexts that affect the dynamics could be changing force fields or liquids with different viscosity in which a manipulator has to operate. In this work, experiments will be conducted for the case that modulation of the dynamics is due to manipulation of different objects. However, most of the methods that will be developed will be applicable to other context variations as well.

Nonstationarity affects control performance as well as the very process of learning of dynamics. The dynamics model that is learned and used for control needs to be adapted every time the dynamics changes to ensure accurate performance. For example, if a model of the robot that is not carrying a load is learned and consequently the robot grasps a heavy object, there will be significant tracking errors. Adapting the model online is possible, however, performance will inevitably be low during the period of adaptation. In this study, we examine ways to improve the control performance by reusing the knowledge obtained by learning previously experienced dynamics (contexts).

We consider two classes of nonstationary behaviour and treating these are the two main goals of this work.

- In the first, the dynamics may switch between a finite set of contexts.

- In the second, more complicated scenario, there is a continuum of possible contexts.

In the discrete context case, a solution is to learn a set of models, each of which is appropriate for a different context and switch between them as required. This allows for fast adaptation of the model that is used for control and greatly improves control performance. This multiple model approach is not new to motor learning and control (Gomi and Kawato, 1993; Cacciatore and Nowlan, 1994; Wolpert and Kawato, 1998; Haruno et al., 2001). Nevertheless, existing methods do not allow *learning of nonlinear* dynamics. We will formulate a multiple model paradigm that is able to learn models of nonlinear dynamics in a principled probabilistic way. The use of probabilistic methods will be useful in order to take into account uncertainties in the learned models properly. A graphical model representation of the multiple model system where a discrete hidden variable indexing the set of dynamics models will be used. Furthermore, we require to learn complicated nonlinear dynamics in high dimensional spaces and an appropriate regression algorithm needs to be used. Using probabilistic methods and a robust nonlinear regression algorithm, we will demonstrate that it is possible to learn multiple models of nonstationary nonlinear dynamics.

In the scenario of continuous range of potential contexts, the use of a finite set of models may not be viable. Nevertheless, the experience gained by learning the dynamics in previously seen contexts can be used to generalize to novel contexts. To the best of our knowledge, there is no previous work on principled ways of generalizing learned dynamics from an observed set of contexts to novel contexts. In some of the multiple model approaches (Wolpert and Kawato, 1998; Haruno et al., 2001), a linear combination of the predictions of the individual models (with weights given by the responsibility signal of each model) is used, however, no justification for doing this is provided. Two possible cases will be examined. In the first case *the nature of the context and its relationship to the modulation of the dynamics is known*. It is shown that when nonstationarity is caused by varying loads, it is possible to use such prior knowledge and generalize learned dynamics from experienced contexts to novel contexts as well. The method proposed for this scenario is related to classical load identification methods (Swevers et al., 2002, 2000; Dutkiewicz et al., 1993a). However, it differs in the fact that learned, instead of analytical, dynamics models are used. The probabilistic model that was proposed in the discrete case is reformulated in order to handle contexts that have a continuous rather than discrete range. A continuous latent variable is introduced to represent the context and the resulting model is similar to a linear state-

space model. Experiments indicate that very accurate estimates of the context can be obtained using this formulation and very high control performance can be achieved. In the second case, *knowledge on the nature of the context and the way it modulates the dynamics is not available*. A further reformulation of the aforementioned state-space model is suggested, which results in a *nonlinear state-space model*. Learning of such a model is considerably more difficult given that the relationship of the context to the dynamics may be arbitrary. Nevertheless, promising experimental results are obtained and satisfactory performance is achieved.

In both the scenarios of a finite and infinite number of contexts, the context needs to be inferred. Another completely novel approach that we suggest in this thesis is to use a dynamics model whose input is augmented with observed variables that convey information about the context indirectly. Then, no estimation of the context needs to be performed and also generalization to novel contexts can be achieved, with the additional advantage that knowledge of the nature of the context is not required. For example, in the scenario of different manipulated objects, tactile sensing at the interface between the arm and the load provides information about the properties of the object in an indirect way. It is demonstrated that this strategy is applicable even when special sensory input – like tactile – is not available, by using time delayed observed dynamics states and inputs as additional variables, similarly to an autoregressive model.

It is necessary to clarify the relevance of this work to biological motor control (Shadmehr and Wise, 2005), a field with similar goals as the field of robot control. (Schaal and Schweighofer, 2005) discusses some of the common concepts between robot control and biological motor control and points out that it is not always clear if the methods and notions of one field are applicable or meaningful in the other. In this work, the focus is on robot control rather than biological motor control. However, when there is a known analogy, it will be mentioned and we hope that the ideas and models developed in this thesis could be useful for biological motor control as well.

1.1 Outline of this thesis

The rest of the thesis is organized as described below. Details of relevant publications are included.

Chapter 2 gives a more detailed background on robot control, dynamics learning and previous approaches for dealing with nonstationarity in the fields of control and machine learning.

Chapter 3 discusses learning a dynamics model of a robot arm with stationary dynamics and using it for control. Also, the effects of nonstationarity on dynamics learning and control are briefly demonstrated.
Chapter 4 discusses the use of a set of models for dealing with a discrete number of contexts. Existing multiple model methods are discussed and a new formulation that deals with some of their problems is presented.

- International Conference on Artificial Neural Networks (ICANN), 2006 (Petkos et al., 2006)

Chapter 5 discusses the disadvantages of the multiple model scenario and introduces an alternative single model formulation augmented with unobserved continuous contextual variables.

- IEEE International Conference on Robotics and Automation (ICRA), 2007 (Petkos and Vijayakumar, 2007a)
- IEEE International Conference on Intelligent Robots and Systems (IROS), 2007 (Petkos and Vijayakumar, 2007b)
- International Conference on Informatics in Control, Automation and Robotics (ICINCO), 2007 (Hoffmann et al., 2007)

Chapter 6 discusses learning an augmented model that takes additional input sensed variables that convey contextual information indirectly.
Chapter 7 concludes and gives suggestions for further research.

Chapter 2

Background

This chapter gives some background on robot control, dynamics learning and the problem of nonstationarity from the perspectives of control (adaptive control) and machine learning.

2.1 Robot control

There are two classes of robot control methods, **direct** and **indirect** control. Indirect control, involves two independent stages:

- *Movement planning*, is the task of generating a representation of the desired motion of the system. Plans are expressed either in terms of joint angles or displacements (for revolute and prismatic joints respectively) or in terms of end-effector position and possibly orientation. The plan is represented either as a differential equation or a time indexed sequence of states θ_1^d, θ_2^d ... θ_T^d between the current state of the system and some desired end state. The desired end state or task could come from some higher level planning or "cognitive" process or from a human demonstrator. Movement planning is often performed by optimizing a cost criterion, e.g. kinematic smoothness of the movement; this is the field of optimal control (Jordan and Wolpert, 1999; Todorov, 2006; Stengel, 1994).

- *Plan execution*, is the task of realizing the planned motion by sending appropriate motor commands to the actuators of the system. Depending on the particular architecture or hardware, the actuators may be commanded in terms of e.g. voltage applied on a DC motor or in terms of the torque (if the joint is revolute) or force (if the joint is prismatic) that the actuator applies on the joint.

This two stage process is depicted in Fig. 2.1.

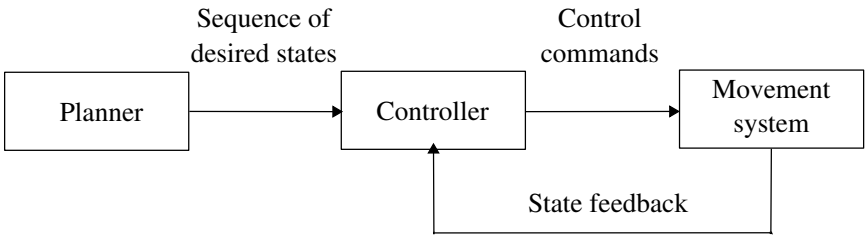


Figure 2.1: Indirect control: motor control involves two stages, planning and plan execution.

In direct control on the other hand, planning and computation of commands are not done independently. That is, no explicit plan is created independently to the computation of commands. A cost function that depends on both the state of the system and the applied commands is used and is optimized in order to obtain a sequence of states and commands. Since this approach utilizes optimization of a cost function for control problem, it also falls in the field of optimal control. The direct control approach is displayed in Fig. 2.2.

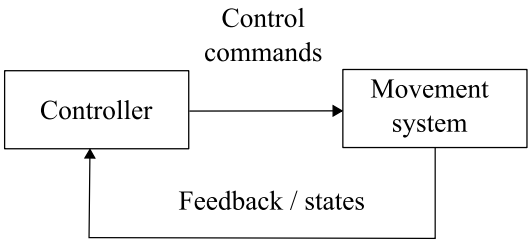


Figure 2.2: Direct control: motor control does not involve an independent planning stage. Planning rather happens in the controller as part of an optimization process.

Applied commands are always issued in terms of actuator coordinates. Nevertheless, plans can be either represented in joint coordinates or end-effector coordinates. If the desired trajectory is presented to the controller in terms of joint states, a **joint space controller** is used. If on the other hand, the desired trajectory is presented as a sequence of end-effector states, this is an **operational space control** problem, which is in general more difficult. The most common approach is to first solve the inverse kinematics problem, mapping the planned trajectory in operational space to another trajectory in joint space and then use a joint space controller. This is depicted in Fig. 2.3.

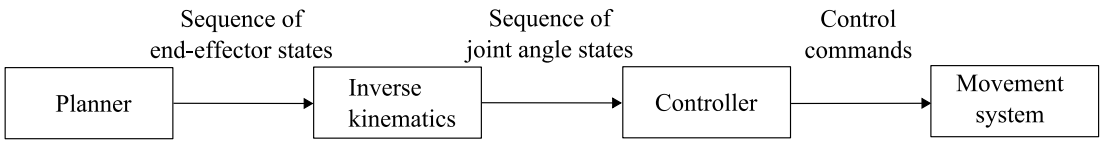


Figure 2.3: Transforming a planned trajectory from end-effector states to joint angle states.

Another option is to use a control law that directly maps plans in operational space to actuator commands, e.g. using the inverse of the Jacobian matrix (Sciavicco and Siciliano, 2000). It must also be stressed that redundant degrees of freedom is an issue that needs to be taken into account when transforming a plan from operational space to joint angle space or when using a pure operational space controller. That is, when there are more degrees of freedom to execute a required task in operational space, the task can be completed in different ways and it may be desired to resolve between these.

Depending on the application requirements, representation of the desired trajectory, knowledge of the dynamics of the robot and hardware, the control problem varies significantly. For example, if the application goal is just to set the manipulator to some fixed posture, this is a **position control** problem. If the task is to execute some movement, without any interaction between the robot and it's environment, this is a **motion control** problem. If it is required to interact with the environment and the task is e.g. to apply some specific force on some surface or object, this is a **force control** problem.

Most importantly, controllers are characterized as **open loop** or **closed loop**, which are also known as **feedback controllers** (Ogata, 2001). In open loop control (Fig. 2.4) there is no feedback from the movement system to the controller: the controller assumes that the system is in its desired state and computes the command that will drive the system to the next desired state. Open loop controllers are not robust to disturbances or system uncertainties. A closed loop controller uses some measurement of the state of the system. In practice some form of feedback is almost always required as in most cases there will be disturbances or uncertainties. One of the most common closed loop controllers is the non-model based feedback controller that is displayed in Fig. 2.5. This uses the sensors of the system to measure or estimate the actual state of the robot θ_t and compares it with the desired state of the robot θ_t^d. It then uses the error e_t and computes the next command to the robot as a simple function of the error. The most common feedback controller is the **Proportional Integral Derivative** (PID)

controller that computes the next control command τ_t as:

$$\tau_t = K_p e_t + K_i \int e_t dt + K_d \frac{de}{dt} \tag{2.1}$$

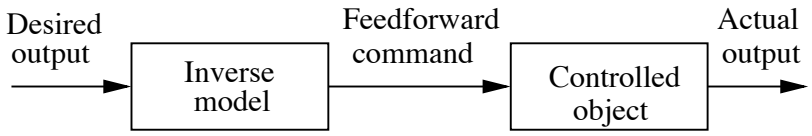


Figure 2.4: An open loop feedforward controller.

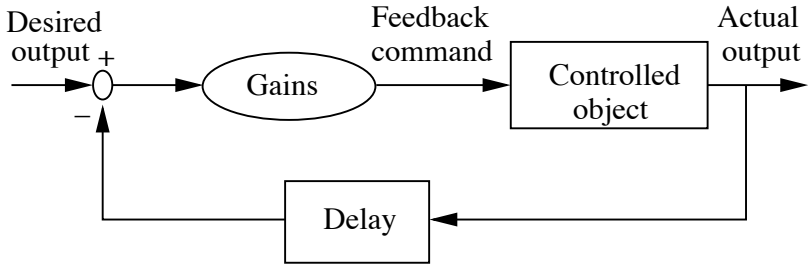


Figure 2.5: A pure feedback controller.

K_p, K_i and K_d are the proportional, integral and derivative gains respectively. Other variants, like the P, PI or PD controller, using only the corresponding terms, are also common. The advantage of these controllers is that they do not require a model of the system's dynamics. However, a PID controller will not provide a command unless there is some error to correct and therefore tracking of the desired trajectory will not be perfect. In addition, there are in many cases feedback delays that may lead to inaccurate computation of the error. Then, the applied feedback command may not be appropriate for controlling the system and could reduce tracking performance, instead of improving it. Feedback delays are usually quite low in most artificial systems, but can play a significant role in biological motor control systems. Furthermore, in many cases, in order to achieve relatively good tracking performance, the gains have to be set to high values, something that results in potentially dangerous non-compliant movement, that could cause the manipulator to damage its environment if it comes in contact with it (Wolpert and Ghahramani, 2000).

High fidelity, compliant robot control requires a sufficiently accurate dynamics model. The dynamics of a robot manipulator is expressed as a second order nonlinear differential equation. The inputs to the system are generally the torques or forces

applied at the joints and the outputs are most commonly the joint angles or displacements. An operational space formulation of the dynamics model where the output of the differential equation is the position and orientation of the end-effector is possible but not common. In the common joint space form, assuming only rotational joints, the dynamics model is:

$$\tau = B(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) \tag{2.2}$$

Here, τ is the vector of torques applied at the joints, q, $\dot{q}$ and $\ddot{q}$ are the vectors of joint angles, velocities and accelerations respectively, B is called the inertia matrix, C is a matrix that accounts for Coriolis and centrifugal forces and g is a vector of gravitational forces. All B, C and g are nonlinear functions of the joint states. For example, the expressions for B and g involve the Jacobian of each link – which depend nonlinearly on the joint angles up to that link – and the expression for C involves the derivatives of elements of B with respect to the joint angles. Friction effects can also be easily included in the model. Static friction can be modeled by including the term $-F_s \operatorname{sgn}(\dot{q})$ and viscous friction can be included by including the term $-F_u\dot{q}$ on the left hand side of (2.2). F_s and F_u are diagonal matrices with the static and viscous friction coefficients respectively. More details about the analytical dynamics of a robot manipulator can be found in (Sciavicco and Siciliano, 2000; Craig, 2005; Spong et al., 2006; Kelly et al., 2005)

Typically, discrete time approximations are used to compute with the dynamics model (e.g. with a small time step $t = \alpha$) as:

$$\tau_t = \bar{B}(q_t)\ddot{q}_{t+\alpha} + \bar{C}(q_t,\dot{q}_t)\dot{q}_t + \bar{g}(q_t) \tag{2.3}$$

(2.3) represents the **inverse dynamics model**, i.e. given that the manipulator is in the state represented by the vectors of joint angles and velocities $q_t, \dot{q}_t$, the inverse dynamics model gives the torque that needs to be applied from time t until $t+\alpha$ so that an acceleration of $\ddot{q}_{t+\alpha}$ is achieved. The next state $q_{t+\alpha}$ and $\dot{q}_{t+\alpha}$ can be obtained by q_t,$\dot{q}_t$ and $\ddot{q}_{t+\alpha}$ by integrating. The matrices $\bar{B}, \bar{C}$ and $\bar{g}$ are the discrete time approximations of B, C and g respectively.

Grouping the state variables at time t, q_t and $\dot{q}_t$ in a single state variable θ_t, the inverse model can be represented in a more compact form as:

$$\tau_t = g(\theta_t, \theta_{t+1}) \tag{2.4}$$

where θ_{t+1} denotes the at time $t+1$. This state-space representation will often be used for brevity.

The inverse dynamics model is most commonly useful for control but can be useful for planning also. The dynamics of the system is also commonly represented in terms of a **forward model**. The forward dynamics model maps the state of the system and applied command (input to the dynamical system) to the state that it will end up in at the next time step. Formally, the forward model is:

$$\theta_{t+1} = g(\theta_t, \tau_t) \tag{2.5}$$

The forward dynamics model can be retrieved from (2.2) by solving for $\ddot{q}$ as:

$$\ddot{q} = B(q)^{-1}(\tau - C(q,\dot{q})\dot{q} - g(q)) \tag{2.6}$$

A graphical model representation (Bishop, 2006) of the forward and inverse model can be seen in Fig. 2.6 (a) and (b) respectively.

Figure 2.6: Graphical model representation of (a) the forward model (b) the inverse model.

Each node in a graphical model represents a random variable. Each variable has a probability distribution assigned to it. If the node for a variable has parents (nodes pointing to it), then the probability distribution is conditioned on the parent variables. For example, in Fig. 2.6 (a), the distribution of θ_{t+1} is conditioned on θ_t and τ_t, i.e. it is $p(\theta_{t+1}|\theta_t, \tau_t)$. The graphical model represents the joint probability distribution of the variables as the product of the probability distribution of the individual variables. Furthermore, shaded nodes represent variables that are observed, whereas nodes that are not shaded represent variables that are not observed. These graphical models have only observed variables but we will later develop and use instances of graphical models with hidden variables.

The inverse model can be used in many control settings. The simplest is as part of an open loop controller, ignoring the actual state of the system and applying a pure "feedforward" command as displayed in Fig. 2.4, i.e. the following control law is used:

$$\tau_t = B(q_t^d)\ddot{q}_{t+1}^d + C(q_t^d, \dot{q}_t^d)\dot{q}_t^d + g(q_t^d) \tag{2.7}$$

or equivalently in the state-space representation:

$$\tau_t = g(\theta_t^d, \theta_{t+1}^d) \tag{2.8}$$

As discussed, open loop control is in almost all cases not an option as it is very sensitive to disturbances and system uncertainties.

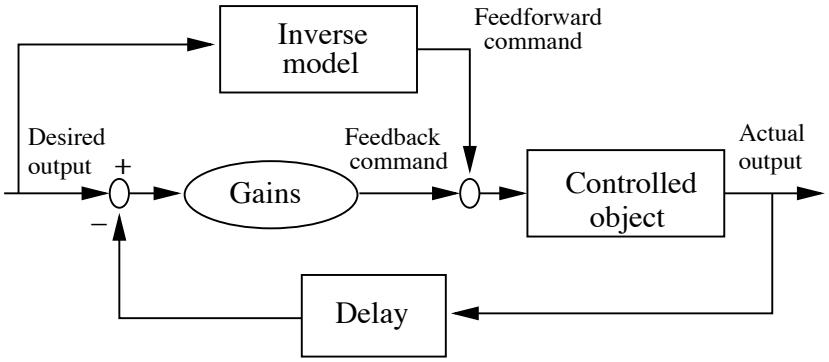


Figure 2.7: A composite controller.

A common type of model based closed loop controller is the **composite controller** (Fig. 2.7) (Jordan, 1996), which is also known as "PD plus feedforward" (Kelly et al., 2005). A composite controller consists of a feedforward part based on the inverse model of the system and a feedback part, a PD controller. If the inverse model of the system is accurate and there are no disturbances, then the errors will be small and the error-correcting feedback part of the controller will be mostly inactive.

Given a discrete time plan $\theta_1^d, \theta_2^d ... \theta_T^d$, the composite control law is expressed as:

$$\tau_t = g(\theta_t^d, \theta_{t+1}^d) + K_p e_t + K_d \frac{de}{dt} \tag{2.9}$$

which is simply the open loop controller of (2.8) with additional PD terms. The more accurate the inverse model is, the lower the feedback component of the command will be, i.e., the magnitude of the feedback command can be used as a measure of the accuracy of the inverse model. Furthermore, good predictive models allow the use of low feedback gains, resulting in a highly compliant system without sacrificing the speed and accuracy of the movements.

There are many more controllers for motion control of robot manipulators, with one of the most important being the **computed torque** control law (for a comprehensive study see (Kelly et al., 2005)). The computed torque control law is given by:

$$\tau = B(q)[\ddot{q}^d + K_p(q^d - q) + K_d(\dot{q}^q - \dot{q})] + C(q, \dot{q})\dot{q} + g(q) \tag{2.10}$$

The composite controller will be used in this thesis because of its favorable properties that were mentioned. Furthermore, it is one of the few controllers that allow use of a learned instead of an analytically derived model. Controllers that do not allow the use of a learned dynamics model are the ones that require computation of the individual terms $B(q), C(q,\dot{q})$ or $g(q)$, as a learned dynamics model does not provide these. The computed torque controller also allows use of a learned dynamics model, however, it does not have the error-correcting properties that the composite controller has since the error is not directly corrected by the feedback loop. Furthermore, the computed torque law is very sensitive to modeling errors that will inevitably be present at least in the initial phases of training.

Reference should also be made to classic control theory (Franklin et al., 1993). The subject of classical control theory is how to determine the behaviour of dynamical systems by setting its inputs (commands). An important requirement for a control system is stability. There are various definitions of stability of a dynamical system (Slotine and Li, 1991; Kelly et al., 2005). For example, it is desired that the controlled system exhibits asymptotic stability (Slotine and Li, 1991; Kelly et al., 2005). This means that the differential equation that describes the dynamics of the tracking error $e_t = \theta_t^d - \theta_t$ is computed and it is required that this differential equation converges to a single stable point which is the origin of the error state-space i.e. zero error and zero error velocity. A complete treatment of any control system should in principle involve stability analysis. However, stability analysis is often very difficult, in particular in cases of learning nonlinear dynamics with complicated regression algorithms. When possible, we will make empirical comments on the stability of the methods that will be developed, based on experiments. Thus, we will rather use a qualitative definition of stability: a control system is called stable if starting from small tracking error, the error stays small.

2.2 Motor learning

It is often the case that an accurate model of the robots's dynamics is not easy or possible to obtain analytically. A reason may be that the dynamics of the system is too complicated to derive analytically and important factors are omitted. For example, the dynamics of robot manipulators is usually based on the assumption of rigid links and joints. However, there may be non-rigid body effects due to elasticity of the links. Even though non-rigid body effects may be important, they are usually ignored since

they are hard to model. Another reason may be that some parameters of the system are not known accurately or at all. In the case of robot manipulators, this could include kinematic (e.g. link lengths or angles of rotation axes), dynamic (e.g. link masses, positions of centers of mass etc.), friction or elasticity parameters. When accurate derivation is not possible, learning the dynamics from movement data is an attractive alternative. The problem of learning dynamics, is essentially a problem of function approximation and a multitude of regression algorithms can be used. However, it is necessary to use an algorithm that is able to learn *nonlinear* models, since the dynamics of manipulators is in general highly nonlinear and it is also desirable to use an *online* algorithm.

It should also be noted that in the engineering language, learning the dynamics of a system from data is called system identification. The term learning instead of identification of dynamics will be used in order to make a connection with the modern statistical machine learning techniques that will be used.

There are three main approaches for learning dynamics: direct inverse learning, feedback error learning and distal supervised learning (Jordan and Wolpert, 1999). These are discussed next.

2.2.1 Direct inverse learning

Typically, in robotic systems with proprioceptive and torque sensing, at each time step t a state transition and an applied torque signal summarized in the triplet $(\theta_t, \theta_{t+1}, \tau_t)$ are observed, i.e., we have access to the true applied control command and the state transition. In *direct inverse learning* different test commands are applied to the actuators of the robot, the actual state transitions are observed and are subsequently used as training data for learning the inverse dynamics model.

Data collection is an issue with direct inverse learning. Exhaustive data generation is only possible for low-dimensional nonlinear regression problems and is thus not a realistic option for any interesting robotic system. Another option is to use an incompletely or coarsely trained model to control the system and collect training data. Nevertheless, it is possible that the data collection process stays in a limited area of the input space that may not be relevant for a specific task.

2.2.2 Feedback error learning

Another approach is *feedback error learning* (Kawato et al., 1987). Contrary to direct inverse learning, the actual observed movement data are not used for training the dynamics model in feedback error learning. Feedback error learning requires a composite controller and the training data for the dynamics model are the desired state transitions and the applied composite command. However, the error-correcting properties of the composite controller should tend to bring the model to the desired states and eventually, the commands produced by the composite controller will be close to the commands that are actually required to produce the desired movement.

The benefit of feedback error control is that – contrary to direct inverse learning – it is goal directed. The error-correcting feedback part of the controller will guide the system to eventually sample data in the area that is required to execute some specific movement. However, given that incorrect training data is at least initially used for training the inverse model, the control command applied by the inverse model will be inaccurate and thus convergence to the actual desired trajectory may be slow. Convergence would depend on the form of the learned model, the complexity of the actual dynamics that needs to be learned and the selection of feedback gains.

The use of feedback error learning will not be considered further in this work. However, – as already discussed – given the benefits of the error-correcting composite controller, a composite controller will be used, but the actual movement data will be used for training instead. That is, direct inverse learning with a composite controller will be utilized.

2.2.3 Distal supervised learning

Another approach for inverse dynamics learning is distal supervised learning (Jordan and Rumelhart, 1992). Distal supervised learning is useful for problems where explicit training data is not easy to obtain and also for learning ill-posed inverse problems. An example of such a problem would be a redundantly actuated arm, i.e. an arm that has more that one actuator at some joint. The actuators could actually be commanded in an infinite number of different ways, so that the same net torque is produced, however, the average of different solutions may not be a solution, i.e. the solution space may not be convex. In a case like this, direct inverse learning would learn the average of solutions and would probably result in no proper solution. Distal supervised learning deals with this problem by first learning a forward model – which is in any case not

ill-defined – and using this in conjunction with the learned inverse model to obtain an identity mapping and resolve between inverse solutions.

2.3 Nonstationarity

The dynamics of the system to be controlled does not usually change through time, i.e. it is stationary. For learning purposes, this means that the learned dynamics model remains roughly valid (as long as it has been sufficiently well learned). However, in many real world applications the dynamics changes, it exhibits nonstationarity. This could be the result of interaction with different environments or objects. For example, the dynamics of a robot manipulator changes as it manipulates objects with different physical properties. Another source of nonstationarity is natural wear and tear, for example joint friction may change after long periods of time.

Since an accurate model of the system's dynamics is important for control, nonstationarity should be taken into account in a learning dynamics for control scenario. In principle, any online learning algorithm can deal with nonstationarity. However, the adaptation period can be long. At this point we need to distinguish between two cases of nonstationarity. One is when the change in dynamics is not reversible, this is usually the case of nonstationarity due to wear and tear. The other is when the change is reversible and the dynamics may switch back and forth, e.g. in the case of a manipulator handling a series of tools to execute different tasks. We focus on the latter case: readapting every time the dynamics changes is an inefficient and suboptimal strategy, since the same models may be learned and unlearned all the time. It is possible to learn the new dynamics every time it occurs, however it is clear that the knowledge obtained previously can be reused, instead of discarded and that this could increase the control performance considerably.

The factors that affect the dynamics of the system are as mentioned before the *context* of the dynamics. In general, the context is not directly observed, e.g. in the example of varying loads, the mass and the inertial properties of the manipulated object are not known. If they were known, say if a measurement had taken place before the manipulation of the load, the problem would be relatively straightforward. In most cases even the nature of the context may not be known, i.e. nonstationary may be observed but the nature of the factor that affects the dynamics may not be known. In the rest of this section, existing approaches for dealing with nonstationarity in the fields of control and machine learning will be reviewed.

2.3.1 Adaptive control

The field of *adaptive control* (Narendra and Annaswamy, 1989; Åström and Wittenmark, 1994; Dumont and Huzmezan, 2002) studies the control of systems with unknown or changing dynamics, i.e. systems with uncertain dynamics.

There are two main classes of adaptive controllers: *Model Reference Adaptive Controllers (MRAC)* and *Self-tuning controllers (ST)*.

In MRAC (Fig. 2.8), a reference model that describes the desired response of the controlled system to the input is defined. The controller is updated in order to best match the actual system behaviour to the output of the reference model. Update is performed using say gradient descent on the controller parameters, e.g. the PD gains and requires knowledge of the structure of the model. No explicit modeling of the dynamics is performed. MRAC is not appropriate for this study, since no knowledge of the structure of the model of the system is assumed to be available, the dynamics is learned from a naive state.

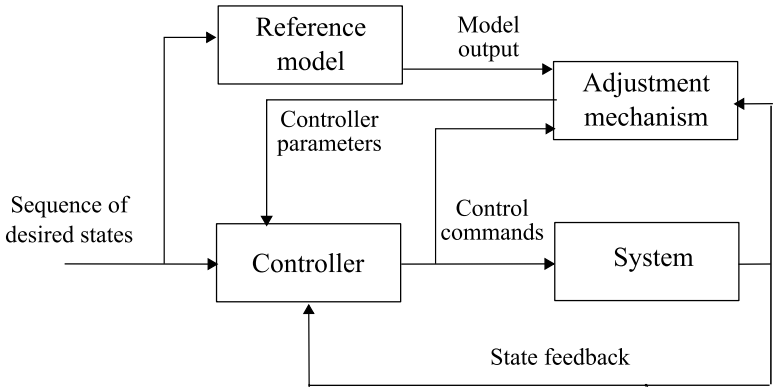


Figure 2.8: Model reference adaptive control.

MRAC is characterized as a direct adaptive control method, as it directly adapts the controller parameters.

Contrary to MRAC, in ST (Fig. 2.9) a model of the system is explicitly estimated. The parameters of the controller are set according to the model estimate. The controller uses the model estimate as if it represents the actual system. This is called the "certainty equivalence principle". Essentially any identification method can be used to obtain a model of the dynamics. There is a multitude of different ST models, according to the method used to identify the dynamics and adjusting the controller according to the identified model.

In the most common case, the identified system model is linear and it is used in

conjunction with linear control techniques. For example, an estimate of the linear dynamics of the system can be obtained by recursive least squares and using the estimated model one could set the gains of a PID controller, using say pole placement (Ogata, 2001) or use the estimated model as part of a composite controller. There are studies on the convergence and stability properties of ST control of linear systems using different estimation and control techniques (Åström and Wittenmark, 1994).

In this work the dynamics models will be explicitly estimated and the learned dynamics model will be used as part of a composite controller. Therefore, this learning control approach can also be classified as a ST control method.

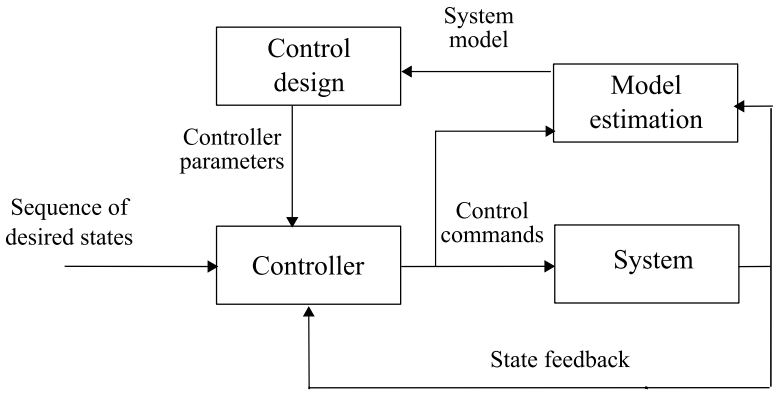


Figure 2.9: Self-tuning control.

Gain scheduling is another control technique that should be mentioned. Gain scheduling is sometimes classified as an adaptive control technique and sometimes not. In gain scheduling, a set of simple – typically linear – controllers is maintained and the system is controlled by switching or by mixing the parameters of the set of controllers. Mixing or switching is performed based on a set of measured variables that are called scheduling variables. Gain scheduling has been extensively used in flight control and is usually highly engineered, i.e. a lot of effort and system dependent knowledge is put into determining the scheduling variables and the way these determine the controller parameters. The use of a set of models (or controller parameters) is reminiscent of the multiple model approach that will be discussed in Chapter 4. The difference is that the multiple model approach does not rely on the existence of explicit scheduling variables, instead it tries to identify the most appropriate model or models using the observed dynamics. Also gain scheduling, does not typically attempt to deal with nonstationary dynamics but with different operating regions.

2.3.2 Supervised learning approaches to nonstationarity

There are various approaches for dealing with nonstationary problems in supervised learning. It should first be noted that most online regression algorithms should be appropriate for nonstationary regression problems, as one of the motivations for online learning is dealing with nonstationarity (the other being to discard bulky data). Thus, any online learning algorithm could in general be used to identify the system model in a self-tuning regulator and could deal with nonstationary dynamics. For example, one could use a multilayer perceptron neural network that is trained with online gradient descent. However, online learning with multilayer perceptrons is prone to the problem of negative interference, i.e. presenting data in a new part of the input space (something that is quite common in the dynamics learning scenario) can degrade the performance of the learner in another part of the space where data has been seen and well fitted previously. Local learning algorithms – i.e. algorithms that use a set of local models that account for different areas of the input space – do not have this problem and are thus suitable for learning dynamics.

Furthermore, there are ways to improve the performance of online learning algorithms under nonstationary contexts (Murata et al., 1996): once nonstationarity is detected, the learning or forgetting rate of the specific learning algorithm can be tuned in order to more rapidly forget the previously seen data and learn the new. This essentially increases the learning rate when the error is large and decreases it when the error is small. When local learning algorithms are used, adapting learning rates would help to adapt each local model faster. However, in order to adapt the whole model (the set of local models), we would have to explore all areas in the input space model.

There is a large amount of work on "switching regression" problems, i.e. regression problems where the data generator switches between a set of different modes. This also related to the problem of multimodal regression, i.e. regression problems with more than one response, a common phenomenon in the so called inverse problems. An example of such a problem is inverse kinematics of redundant arms: a manipulator may be able to reach some end-effector position in more than one configuration. The problem is essentially to learn to predict the different possible responses to the same input. Next we look at some popular approaches to switching regression and techniques used to perform inference with them.

Conditional mixture models

An approach to achieve multiple responses is to use a conditional mixture model

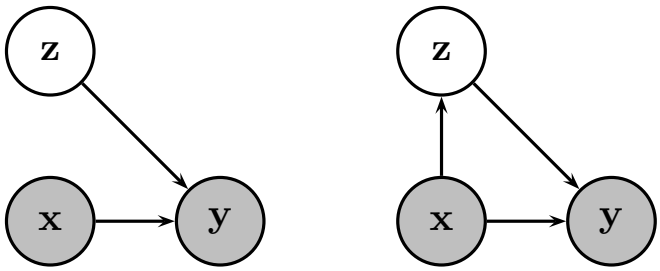


Figure 2.10: Graphical models depicting a conditional mixture model (left) and a mixture of experts model (right).

(Bishop, 2006), which is the supervised learning equivalent of a mixture model for modeling probability densities in unsupervised learning. That is, as a Gaussian mixture model provides a multimodal probability distribution for some datapoint, a conditional mixture model provides a multimodal probability distribution for the output of a regression problem.

A graphical model representation of a conditional mixture model is displayed in Fig. 2.10 (left). The output variable is y and the input variable is x. The unobserved (and hence not shaded) variable z is discrete and indexes different regression modes. The distribution $p(z)$ (the prior of each mode) is a multinomial distribution. If z ranges from 1 to K, then there are K different modes in the probability distribution.

Given that the graphical model represents the joint probability distribution as the product of the probability distributions of each variable, the joint distribution for the graphical model in Fig. 2.10 (left) is:

$$p(x,y,z) = p(x)p(y|x,z)p(z) \tag{2.11}$$

What we are interested in is to obtain the (multimodal) probability distribution of the output given some input, i.e. we are interested in $p(y|x)$. Using the two fundamental rules of probability (the sum and product rule) we find from (2.11) that $p(y|x)$ is given indeed by a mixture distribution as required:

$$p(y|x) = \sum_{k=1}^{K} p(z=k)p(y|x,z=k) \tag{2.12}$$

Mixture of experts

When a dependency of the hidden discrete variable on the input variable x is introduced, the model is called a mixture of experts model (Jacobs et al., 1991; Xu et al., 1995). The graphical model representation of a mixture of experts model is displayed in 2.10 (right). This represents the following joint probability distribution.

$$p(x,y,z) = p(x)p(y|x,z)p(z|x) \tag{2.13}$$

Again, if we use the basic rules of probability, we find that the conditional density of the output given the input is given by:

$$p(y|x) = \sum_{k=1}^{K} p(z=k|x)p(y|x,z=k) \tag{2.14}$$

A mixture of experts model is useful not only for multimodal problems but also for complex problems where it is easier to decompose the problem in local areas in input space (by defining $p(z|x)$ to give probability mass in local areas of x). For example, one could use a set of simple models for $p(y|x,z=k)$, e.g. linear Gaussian and still be able to model a complicated nonlinear predictive distribution $p(y|x)$. In the first mixture of experts model (Jacobs et al., 1991), the mixing coefficients $p(z=k|x)$ were modeled using a neural network, which was termed the gating network. A further development of the mixture of experts model came in the form of the hierarchical mixture of experts model, which further divides areas of locality (Jordan and Jacobs, 1994). It is worth noting that the mixture of experts approaches may be more useful for decomposing a complex learning problem in input space for stationary problems rather than learning a nonstationary or multimodal regression problem. Conditional mixture models are usually more suitable for dealing with nonstationary problems and the focus will be on these from now on.

Expectation Maximization for conditional mixture models

As these models are graphical models with hidden variables, the Expectation Maximization (EM) algorithm is typically used to train them (gradient descent on a likelihood function was used for the initial model in (Jacobs et al., 1991)). It is worth going in some detail about the EM algorithm for the conditional mixture model, as this is related to the multiple model approach that will be the focus of Chapter 4.

The EM algorithm (Dempster et al., 1977; Bilmes, 1997) tries to maximize the likelihood of the observed data, i.e. in the case of the conditional mixture model the inputs $X = [x_i]_{1...N}$ and the outputs $Y = [y_i]_{1...N}$, where N is the number of datapoints. Maximizing the likelihood is equivalent to maximizing the log-likelihood as the logarithm is a monotonically increasing function. Thus, for mathematical convenience, the log-likelihood is typically maximized instead of the likelihood. The log-likelihood is given by the expression:

$$\mathcal{L}(X,Y) = \sum_{i=1}^{N} \log \sum_{k=1}^{K} p(z_i=k)p(y_i|z_i=k,x_i) \tag{2.15}$$

The goal is to maximize this log-likelihood with respect to $p(z=k)$ and the parameters of $p(y_i|z_i=k,x_i)$ (the distribution of the inputs $p(x)$ that should in principle

be included in the likelihood function is not of interest in general – as it is not useful for predicting the output for some input – and is not modeled). Because of the sum inside of the logarithm, this maximization is not easy to perform exactly using nonlinear optimization techniques. Instead, of maximizing the log-likelihood directly, the EM algorithm maximizes the expected (with respect to the hidden variables) complete data log-likelihood (i.e. the log-likelihood if the hidden variables were also observed):

$$\mathcal{Q} = \sum_{i=1}^{N} \sum_{k=1}^{K} p(z_i = k|y_i, x_i)[\log p(z_i = k) + \log p(y_i|z_i = k, x_i)] \tag{2.16}$$

The expected complete log-likelihood is a much easier quantity to maximize with respect to the parameters. The EM algorithm is an iterative procedure that alternates between computing the expected complete log-likelihood, i.e. computing the posterior $p(z_i = k|y_i, x_i)$ (E-step) and maximizing the parameters of the expected complete log-likelihood (M-step). In the E-step, the parameters estimated in the previous M-step are used and kept fixed.

In the conditional mixture model scenario, the posterior $p(z_i = k|y_i, x_i)$ is easily computed using Bayes law as:

$$p(z_i = k|y_i, x_i) = \frac{p(y_i|, z_i = k, x_i)p(z_i = k)}{\sum_{k=1}^{K} p(y_i|, z_i = k, x_i)p(z_i = k)} \tag{2.17}$$

Maximizing the expected complete log-likelihood with respect to $p(z = k)$ is also straightforward. It can be easily shown that it can be exactly maximized by setting:

$$p(z = k) = \frac{1}{N} \sum_{i=1}^{N} p(z_i = k|y_i, x_i) \tag{2.18}$$

Maximizing with respect to the parameters of $p(y_i|, z_i = k, x_i)$ can be more tricky, depending on the form of the distribution chosen. If it is chosen to be a Gaussian centered at the prediction of some parametric model $g_k(x)$ with some variance Σ_k, i.e.

$$p(y_i|z_i = k, x_i) = \mathcal{N}(g_k(x_i), \Sigma_k) \tag{2.19}$$

then maximizing with respect to Σ_k is achieved by:

$$\Sigma_k = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p(z_i = k|y_i, x_i)(y_i - g_k(x_i))(y_i - g_k(x_i))^T \tag{2.20}$$

Maximization with respect to $g_k(x)$ can be more complicated depending on the specific form of g chosen.

Annealed competition of experts and related methods

There is more work on learning switching and multimodal problems (Kohlmorgen et al., 1994; Müller et al., 1995). In (Kohlmorgen et al., 1994), a set of predictors $f_k(x_i)$ give some prediction that is compared to the actual observed output y_i to give the error term:

$$e_k(x_i) = |f_k(x_i) - y_i| \tag{2.21}$$

Then, a moving average filter is applied to the error terms to get a new error term as:

$$E_k(i) = \frac{\sum_{j=-\eta}^{\eta} e_k(x_{i+j})}{2\eta + 1} \tag{2.22}$$

The i^{th} datapoint is then used to train only the predictor f_k with the smallest $E_k(i)$. The algorithm alternates between computing the errors and training the predictors. This approach resembles the EM used for training a conditional mixture model. It essentially differs in the fact that it computes errors only instead of posterior probabilities of a hidden variable and that hard assignment of data to models is used instead of maximization of the expected complete log-likelihood which amounts to taking into account all the data for training all the models, weighted of course by the respective posterior probabilities.

In (Müller et al., 1995), the squared prediction error for each predictor is computed and is used to compute weightings as:

$$\gamma_k^i = \frac{e^{-\beta e_k(x_i)}}{\sum_j e^{-\beta e_j(x_i)}} \tag{2.23}$$

This is equivalent to inferring the posterior of the hidden variable in a conditional mixture model, with β being the inverse variance of the Gaussian noise and the difference that the noise variance is the same for the different predictors and all predictors have equal prior probabilities. All the datapoints are then used to train all models and the γ_k^t are used to weigh the contribution of each datapoint, i.e. this approach is closer to the EM procedure for training conditional mixture models. The parameter β controls the competition between the predictors for data. When β has a small value, the γ_k^t will tend to have similar values and the predictors will be trained with similar data, i.e. there is a low competition between the predictors for data. As β grows larger, the competition increases and at the limit $\beta = \infty$, the competition is hard and only one model is trained with each datapoint as in (Kohlmorgen et al., 1994). An annealing of β is suggested, from a low to a high value. If this is done carefully, i.e. increasing

the competition only when the overall prediction error stops decreasing, the data can be separated between experts successfully. The resulting algorithm was termed the Annealed Competition of Experts (ACE). Also, a similar as before moving average procedure on the errors can be used and can improve the separation of data between models when the predictions of the different predictors cross.

Furthermore, in (Liehr et al., 1999) an extension of (Müller et al., 1995) is presented, where a moving average of the errors is replaced by a Hidden Markov Model to model the temporal smoothness of the unobserved switching process. This uses an input dependent transition matrix $A(x)$, which is modeled by a neural network. The competition factor β is still annealed and the model is trained using gradient descent on a likelihood function. This is a step towards the proper probabilistic approach that will be used and discussed later, with the difference that gradient descent is used for training instead of the more efficient EM and that the competition factor (noise estimate) is annealed instead for learned.

Mixture density networks

Mixture density networks (Bishop, 1994) is another approach that has been proposed for multimodal regression problems. Mixture density networks are supervised neural networks – in (Bishop, 1994) multilayer perceptrons are used – whose outputs represent the parameters of a mixture model. Say the prediction is given by a Gaussian mixture model of the form:

$$p(y|x) = \sum_{k=1}^{K} p(z=k)\mathcal{N}\left(g_k(x), \Sigma_k(x)\right) \tag{2.24}$$

The parameters of this distribution are the mixing proportions $p(z=k)$, the means $g_k(x)$ of the Gaussians and the variances $\Sigma_k(x)$ of the Gaussians. If there are K different modes and l outputs and assuming independent noise between outputs, i.e. diagonal $\Sigma_i(x)$, then the mixture density network will have $k(1+2l)$ outputs. The model is trained using gradient descent on a likelihood cost function. A more efficient approach, using radial basis function networks and an Expectation Maximization algorithm has been suggested in (Vlassis and Kröse, 1999). Also, regularization in mixture density networks is discussed in (Hjorth, 1999).

Other methods

Another approach for multimodal and nonstationary regression has been proposed in (Ghahramani and Jordan, 1994; Ghahramani, 1993). Ghahramani proposes to learn the joint probability density of the input and output variables using a mixture model. That is, learn the probability density $p(y,x)$. Then, the regression problem is to com-

pute the conditional density $p(y|x)$. The probability density $p(y,x)$ is modeled as a mixture of Gaussians and thus the conditional density $p(y|x)$ is also a mixture of Gaussians with as many mixture components as the joint density $p(y,x)$. This approach requires to know the appropriate number of mixture components (although there are ways to add mixture components incrementally on an as needed basis) and in most cases a very large number of mixture components may be required to model the input-output relationship accurately.

A nonparametric technique has also been suggested for multimodal regression (Einbeck and Stutz, 2006b,a). In this approach, a kernel density estimator is used to obtain an estimate of $p(y|x)$ and then an iterative search procedure is used to find the modes of this density. Nevertheless, this approach relies on proper setting of the kernel widths and this can be tricky.

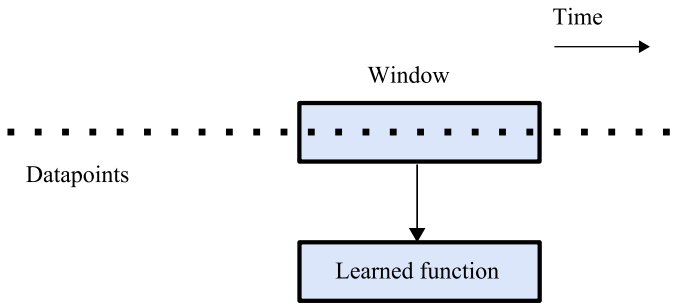


Figure 2.11: Learning from a moving window of data for dealing with nonstationarity. Only the data inside the sliding window are used to train the learner at each time.

Furthermore, a short survey of methods for dealing with nonstationarity in classification and concept learning is presented in (Kubat, 2004). Although not directly relevant to the regression problem that is of interest for a learning dynamics task, it is worth mentioning some of the approaches that are used. The most common approach is to learn using a limited, moving window, set of data (Fig. 2.11). This approach has been treated in the FLORA systems, including techniques for detecting context change (using some heuristic that depends on the prediction accuracy) and automatically adapting the window size and for storing identified contexts (Widmer and Kubat, 1993) (using another heuristic for defining concepts as stable and then storing them). Similar approaches for dealing with nonstationarity in concept learning are presented in (Harries et al., 1998; Harries and Horn, 1996; Kubat and Widmer, 1995). The approach of a moving window of training data, although seems fairly reasonable, is not really applicable since it would require – even with a relatively small window – re-

learning the model very frequently, something that is clearly prohibitive for a nonlinear control problem. However, the approach of storing learned definitions of concepts and using them as they recur is related to the approach of using a set of models that will be discussed later.

2.3.3 Nonstationarity in reinforcement learning

Reinforcement learning (Sutton and Barto, 1998) (Kaelbling et al., 1996) is the subfield of machine learning that deals with the problem of *learning to act* in order to maximize received reward. The results of actions can be stochastic and there may be delayed rewards, complicating the problem significantly. Nonstationarity in reinforcement learning has been treated in (da Silva et al., 2006a,b; Doya et al., 2002). (Doya et al., 2002) is of particular interest as it is essentially a direct extension of the multiple model paradigm that has been proposed in (Wolpert and Kawato, 1998) and will be considered later. All of the mentioned work relies on the use of a set of models, each of which is appropriate for a different context. The approach of Silva compared to the approach of Doya, has the advantage that it can deal, after tuning of the relevant parameters, with an unknown number of contexts, i.e. it allows for dynamic addition of models. It learns a transition and a reward model for each context, from which the policy for each context is derived and consequently switches as appropriate. The work of Doya on the other hand allows the use of local models for the control of both nonlinear and nonstationary systems by allowing different local models to decompose the control problem in space (the state-space of the system) and time (the different contexts). However, Doya'a work does not deal with an unknown number of contexts, i.e. the correct number of contexts needs to be known and also the correct number of local models needed to deal with each context needs to be known also. As mentioned, its architecture is similar to the architecture proposed in (Wolpert and Kawato, 1998), i.e. it uses a pair of predictor (forward models) and controller (inverse models) modules, instead of transition (the forward model essentially) and reward models.

2.4 Transfer learning

One of the goals of this work is to generalize the knowledge obtained by experiencing a set of contexts to other contexts as well. This goal is related to a subfield of machine learning that comes under a few different titles: lifelong learning, learning to learn,

multi-task learning, transfer learning (Thrun and Pratt, 1997). Transfer learning is a topic that has attracted a lot of attention in the machine learning community and studies how learning one task could help improve learning another related task. Transfer learning methods are relevant for our problem since the dynamics of the individual contexts are related and thus, the use of such methods could improve the speed and accuracy of learning in novel contexts, in particular regression methods like (Bonilla et al., 2008; Evgeniou et al., 2005). Transfer learning has been applied to accelerate learning of new environments from autonomous robots (Thrun and Mitchell, 1995) and there has been some recent work on applying transfer learning techniques for learning inverse robot dynamics (Chai et al., 2008).

Chapter 3

Motor learning and control using a single dynamics model

In this chapter learning a single inverse dynamics model and using it for control under stationary and nonstationary conditions is discussed. As was mentioned in the previous chapter, the problem of dynamics learning for control is essentially a problem of nonlinear regression. A robust nonlinear regression algorithm that is suitable for motor learning tasks is presented first and experiments using this algorithm for learning dynamics and control of two robot arms with stationary dynamics are performed. The chapter concludes by experimentally examining the effects of nonstationarity on motor learning.

3.1 Locally Weighted Projection Regression

Locally Weighted Projection Regression (LWPR) (Vijayakumar et al., 2005) – an algorithm which is extremely robust and efficient for incremental learning of nonlinear models in high dimensions is used as our regression tool of choice. An LWPR model consists of a set of linear models, each of which is accompanied by a locality kernel that defines the area of validity of the linear model. The kernel has a Gaussian form. That is, for some input vector $\boldsymbol{x}$ the activation of the k^{th} locality kernel is given by:

$$w_k = \exp\big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c}_k)^T D_k(\boldsymbol{x}-\boldsymbol{c}_k)\big) \tag{3.1}$$

where $\boldsymbol{c}_k$ is the center of the kernel and D_k, the distance metric, defines the shape and size of the kernel. The k^{th} local model gives an output prediction $y_k(\boldsymbol{x})$ using Partial Least Squared (PLS). PLS projects the input on a small number of directions in

input space (the directions of maximum correlation with the output) and then performs linear regression on the projected inputs. The use of PLS makes LWPR suitable for high dimensional input spaces. The combined prediction of the LWPR model, $\hat{y}$, is

$$\hat{y}(\boldsymbol{x}) = \frac{1}{W} \sum_k w_k(\boldsymbol{x})\, y_k(\boldsymbol{x}) \;, \quad W = \sum_k w_k(\boldsymbol{x}) \;. \tag{3.2}$$

The schematic of the LWPR framework is displayed in Fig. 3.1.

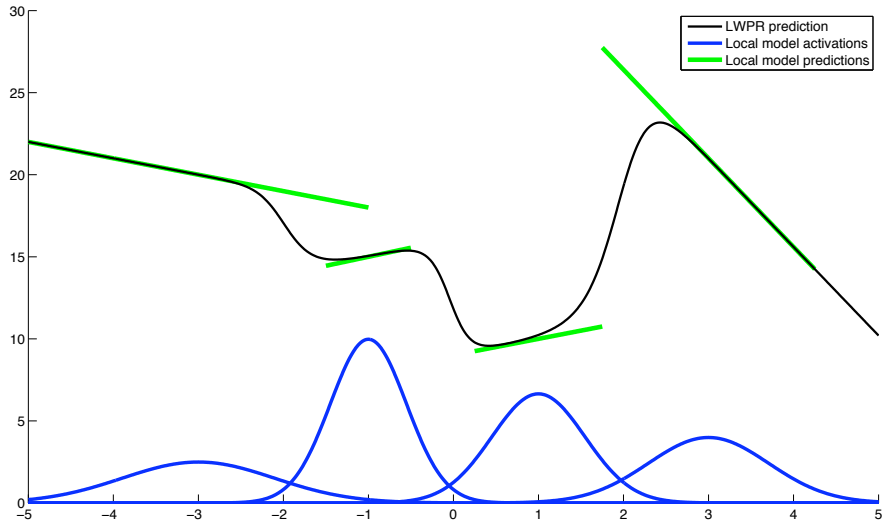


Figure 3.1: Structure of an LWPR model. Each local linear model gives some prediction (green line) and has some activation (blue line) that defines the area of validity of the local model. The total model prediction is a weighted sum of the predictions of the local models.

The parameters of the local linear models, i.e. the projection directions and the regression coefficients are learned online using a recursive version of locally weighted PLS. Projections are added on an as needed basis: one additional projection direction is always kept and is included in the regression only if the error of the additional projection is smaller than some percentage of the error of the previous projection. The distance metric D_k is also learned online using gradient descent on the following cost function:

$$J = \frac{1}{\sum_{i=1}^{M} w_i} \sum_{i=1}^{M} w_i (y_i - \hat{y}_{i,-i})^2 + \frac{\gamma}{N} \sum_{i,j=1}^{N} D_{ij}^2 \tag{3.3}$$

The first term in (3.3) is the leave one out cross validation error (hence the subscript $i,-i$) on a set of M datapoints and can be reformulated in terms of the PRESS residual

error. This allows online incremental computation of the cost function. The second term (whose relative contribution is determined by the factor γ) penalizes too narrow kernel widths (small values in D_k) and prevents overshrinking of local models. The distance metric is updated using a stochastic approximation of the gradient $\frac{dJ}{dM}$ as:

$$M^{n+1} = M^n - a\frac{dJ}{dM} \quad \text{where} \quad D = M^T M \quad \text{(for positive definiteness)} \tag{3.4}$$

The complete set of equations for updating the local model regression and projection parameters as well as the distance metric can be found in (Vijayakumar et al., 2005).

Furthermore, local models are added on an as needed basis, i.e. when no kernel is activated above some threshold for some new datapoint, a new local model is created centered at the new datapoint. Pseudocode for learning with LWPR is given in Table 3.1

Table 3.1: Pseudocode for learning with LWPR. Adopted from (Vijayakumar et al., 2005)

```
- Initialize the LWPR with no local models
- For every new training sample (x,y):
        - For k=1: to K (# of local models):
                - calculate the activation w_k
                - update projections, regression coefficient and distance metric D_k
                - check if number of projections needs to be increased
        - If no local model was activated by more than w_gen:
                - create a new local model centered at x
```

LWPR provides statistically sound input dependent confidence bounds on its predictions. To derive the confidence bounds, the following data-generating process is postulated for the k^{th} local model prediction $y_{q,k}$ for some new input x_q:

$$y_{q,k} = y_q + \varepsilon_1 + \varepsilon_{2,k} \tag{3.5}$$

This corresponds to the weighted linear regression model in (Gelman et al., 1995) with two separate noise processes $\varepsilon_1 \sim N(0, \sigma^2/w_k)$ and $\varepsilon_{2,k} \sim N(0, \sigma^2_{pred,k}/w_k)$. The first noise process is independent of the local model and accounts for the differences between the predictions of the local models and the second is the noise process of the individual local models. Then, the predictive variance over all models can be approxi-

mated as:

$$\sigma^2_{pred} = \frac{\sum_k w_k \sigma^2}{(\sum_k w_k)^2} + \frac{\sum_k w_k \sigma^2_{pred,k}}{(\sum_k w_k)^2} \tag{3.6}$$

This expression, requires estimates for σ^2 and $\sigma^2_{pred,k}$. In (Vijayakumar et al., 2005), σ^2 is approximated as $\sum_k w_k (\hat{y}_q - \hat{y}_{k,q})^2 / \sum_k w_k$ and an incremental update is derived for $\sigma^2_{pred,k}$.

The role of LWPR in acquiring the probabilistic inverse model of Fig. 2.6 (b) can be summarized in the equation:

$$P(\tau \,|\, \theta_{t+1}, \theta_t) = \mathcal{N}\left(g(\theta_{t+1}, \theta_t),\ \sigma(\theta_{t+1}, \theta_t)\right), \tag{3.7}$$

where $g(\theta_{t+1}, \theta_t)$ is now a learned LWPR regression, mapping state transitions to torques. Here, we have two options for choosing the variance: (1) we can assume a fixed noise level independent of the context and the input; (2) we can use the confidence bounds provided by each LWPR model which also depends on the current input (θ_{t+1}, θ_t). We will test both cases in our experiments.

There are other learning algorithms that have been applied for learning dynamics (Nguyen-Tuong et al., 2008), e.g. Gaussian Process Regression (GPR) (Rasmussen and Williams, 2006) or Support Vector Regression (SVR) (Müller et al., 2001). In (Nguyen-Tuong et al., 2008), GPR, SVR and LWPR models are trained offline and tested on data gathered from a simulation of the SARCOS arm and the real SARCOS arm. Accuracy of GPR and SVR on the test data is higher than the accuracy of LWPR, both for the simulation and the real arm data. GPR and SVR have the additional advantage that they do not depend on a large number of parameters – like LWPR – and are therefore easier to tune. The drawback is that training of GPR and SVR has taken roughly twice the time than LPWR. In addition, lookup during online control was slower for GPR and SVR than LWPR and could not be performed in the same frequency as LWPR. In addition, GPR and SVR can not be used in an online learning scenario. In conclusion, currently no regression algorithm can match the computational efficiency and online learning ability of LWPR for learning dynamics in high dimensional scenarios. LWPR is particularly suitable when the methodology needs to be scaled up to deal with large amounts of training data as well as deal with high dimensional movement systems.

3.2 Experiments

We verify the ability to learn the inverse model online with LWPR and show that the learnt model can successfully be used for control. We demonstrate our results on two physically realistic simulations of robotic arms, a 3 DOF artificial arm and a 7 DOF DLR LWR III simulation.

3.2.1 Experimental setup

A 3 DOF robot arm (see Fig. 3.2) was used. This resembles a 2 DOF planar arm, with the third DOF (the first joint) allowing up and down movement. Simulations were performed using the Open Dynamics Engine library (ODE) and OpenGL.

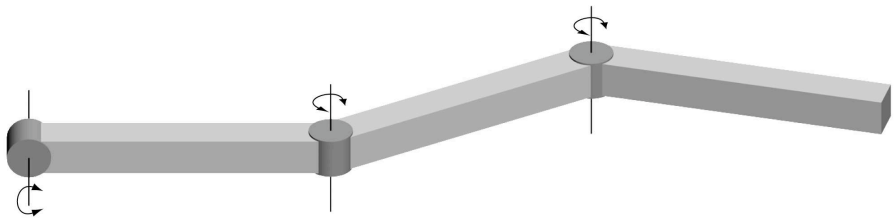


Figure 3.2: Simulated 3 DOF arm.

Simulations with the Open Dynamics Engine

ODE is an open source library for simulating articulated rigid body dynamics. Rigid bodies are the primary building blocks of an ODE based simulation. By rigid we mean that elasticity of objects is not modeled (although no real object is infinitely stiff). In a typical ODE simulation, a set of rigid bodies is defined (boxes, cylinders, spheres) along with a set of joints between them. These joints, constrain the relative movement between the bodies that are connected to the joint. For example, in Fig. 3.3, we can see two boxes connected with a hinge joint. A hinge joint constrains the movement between two bodies so that they can only rotate around a single axis relative to each other. To define such a structure in ODE, we need to define the initial position and orientation of the two bodies as well as the rotation of axis of the joint in some global coordinate system. Other joints like the two-axis, the universal (three-axis) and the slider joints are provided. An articulated object, like a robot arm, can be created in ODE by setting up a sequence of bodies, connected by joints. Each body is characterized by a shape that is used for collision detection and some inertial parameters (mass, position of center of mass and inertia tensor) that are used for the movement simulation.

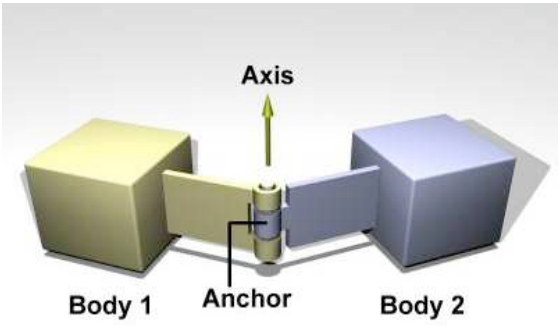


Figure 3.3: Hinge ODE joint. Adopted from the ODE user guide (Smith, 2006).

At each iteration cycle, a set of forces or torques are applied at the bodies or joints and ODE returns the next state of the system (i.e. bodies positions, rotations, linear and angular velocities, joint angles and velocities). Furthermore, ODE performs collision detection and applies the appropriate forces to the bodies when a collision is detected. The pseudocode in Table 3.2 describes a typical ODE simulation.

In addition, we use the shapes, positions and orientations of the bodies provided by ODE to visualize the movement of the system using OpenGL.

Table 3.2: Pseudocode for a typical ODE simulation loop. Partly adopted from the ODE user guide (Smith, 2006).

```
1. Create a world and set world parameters.
2. Create bodies in the world.
3. Set the state (position, rotation etc.) of all bodies.
4. Create joints in the world.
5. Attach the joints to the bodies.
6. Set the parameters of all joints.
7. Loop:
   a. Apply forces to the bodies as necessary.
   b. Adjust joint parameters as necessary.
   c. Call collision detection.
   d. Take a simulation setup.
8. Destroy the dynamics world.
```

The structure and physical properties of the 3 DOF arm are given in Table 3.3 and sample C++ code for setting up the simulation of the arm is given in Appendix A.

In the second setup, the DLR light-weight arm (Fig. 3.4 (left)) was simulated using the Matlab robotics toolbox (Corke, 1996). The DLR light-weight arm is a high

Table 3.3: Structure of the artificial 3 DOF simulated arm

Links				
Link	Length	Width	Depth	Mass
1,2,3	1	0.2	0.2	1 Kg
Joints				
Body 1	Body 2	Axis of rotation		
Pole	1	[1 0 0]		
1	2	[0 0 1]		
2	3	[0 0 1]		

performance manipulator developed at the German Aerospace Center. It has 7 DOFs and it's structure is represented in Fig. 3.4 (right). The exact physical parameters of the DLR arm are protected and cannot be included here.

Simulation using the Matlab Robotics Toolbox

The Matlab Robotics Toolbox (MRT) is a set of tools for kinematics and dynamics problems of generic manipulators. The dynamics tools of the MRT are also based on the assumption of rigid bodies. To run a simulation in the MRT, one again needs to define a set of links and joints between the links, then loop applying a set of forces or torques at the joints, computing the next state and so on, in a similar way as in an ODE simulation.

3 DOF arm task

The task of the 3 DOF arm was to follow a simple trajectory planned in joint angle space, consisting of a superposition of sinusoids with different phase shifts for each joint:

$$\theta_i^* = a_i \cos(\alpha_i \frac{2\pi}{T} t) + b_i \cos(\beta_i \frac{2\pi}{T} t) \ , \tag{3.8}$$

where $T = 4000$ is the total length of the target trajectory (which is 40 seconds, since control happens at 100 Hz), $a_i, b_i \in [-1, 1]$ are different amplitudes and $\alpha_i, \beta_i \in \{1, .., 15\}$ parameterize different frequencies. This task was selected so that a sufficiently rich movement will be executed. As it will explained in Chapter 5, a sinusoidal task can be useful for exciting specific components of the dynamics that are of interest. This task will be used for all experiments with the 3 DOF arm in this thesis.

20 iterations of the trajectory were repeated: during the first four iterations, pure feedback PD control (2.1) was used to control the arm, while at the next 16 iterations, a composite controller (2.9) using the inverse model being learned was used. Every

second datapoint was used for training whereas the rest was used for testing. The gains were lowered as training proceeded. The procedure was executed ten times and repeated for ten different stationary contexts for accumulating statistics. Different contexts are simulated by attaching an object with different mass at the last link of the arm. The mass of the load ranged from 0.1 Kg to 1.9 Kg, in increments of 0.2 Kg.

DLR arm task

The task for the experiments with DLR arm simulation was to follow a figure of eight trajectory in task space. The inverse kinematics for the task space trajectory were computed to obtain a smooth trajectory in joint angle space. The trajectories of individual joints have enough variation so that again a sufficiently rich movement will be executed. 30 iterations of the trajectory were repeated. This time, the composite controller was used from the beginning of the simulation and again every second datapoint was used for training whereas the rest was used for testing. The procedure was repeated six times for six different contexts. The mass of the load ranged from 0.1 Kg to 0.6 Kg, in increments of 0.1 Kg.

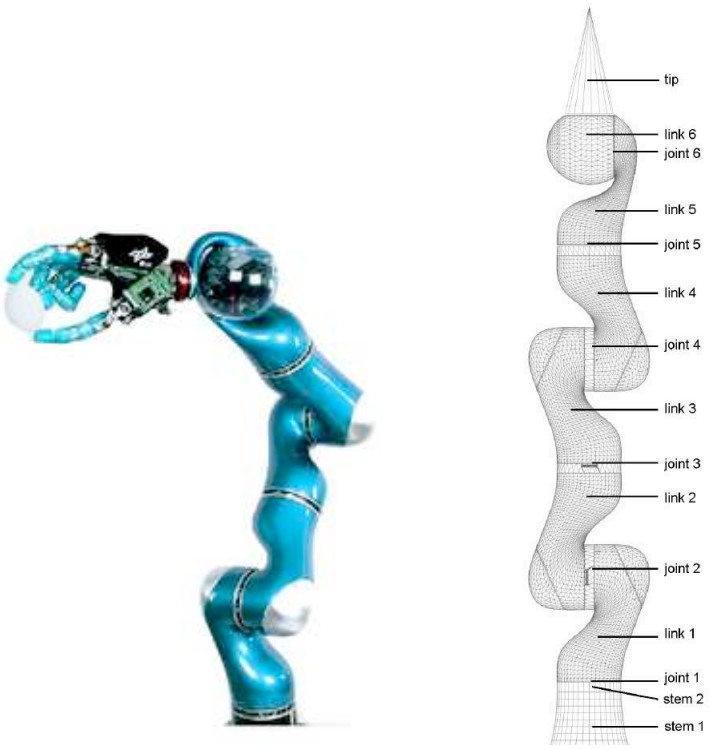


Figure 3.4: The DLR light-weight arm (left). Structure of the DLR arm (right).

Performance measures Three performance measures are used:

1. The normalized mean squared error (nMSE) between the torques predicted by

the LWPR model and the true torques experienced on the test data (i.e. the data that was held out from the training). Denoting the prediction of the model as $\hat{\tau}_t$ and the actual torque experienced as τ_t, the nMSE is given by:

$$\text{nMSE} = \frac{1}{\text{var}(\tau)T}\sum_{t=1}^{T}(\hat{\tau}_t - \tau_t)^2 \tag{3.9}$$

2. The relative contribution of the feedback command to the composite command. Let the feedforward component of the applied command be τ_t^{ff} and the feedback component of the command be τ_t^{fb}. The relative contribution of the feedback command to the composite command is given by the ratio:

$$\frac{|\tau_t^{fb}|}{|\tau_t^{fb}| + |\tau_t^{ff}|} \tag{3.10}$$

We compute the mean of this quantity over a whole iteration. The more accurate the learned model is, the smallest will the error-correcting feedback be and therefore, the smaller the ratio will be. This gives an indirect measurement of how well the inverse model used approximates the actual dynamics.

3. The absolute tracking error. That is, if the desired joint angle at time t is θ_t^* and the actual joint angle is θ_t, the absolute tracking error is $|\theta_t^* - \theta_t|$. We also compute the mean of this quantity over a whole iteration.

3.2.2 Results

The results for the 3 DOF robot are given in Fig. 3.5. Results are averaged over the ten trials and error bars indicate the standard deviation between the trials. The left plot shows how the nMSE on the test data drops as training proceeds and settles at a very low value. The contribution of the error-correcting feedback command to the feedforward command (see Fig. 3.5 (middle)) is low, vouching for the accuracy of the learnt model while being used for control. Furthermore, the tracking error (Fig. 3.5 (right)) is very low and improves significantly when we switch to composite control. For all performance measures, the variance between trials is low, indicating that they have a similar behaviour in all trials.

The results for the DLR arm are presented in Fig. 3.6. Again, on Fig. 3.6 (left) it can be seen that the normalized mean squared error between the torques predicted by the LWPR model and the true torques experienced on the test data is very low over

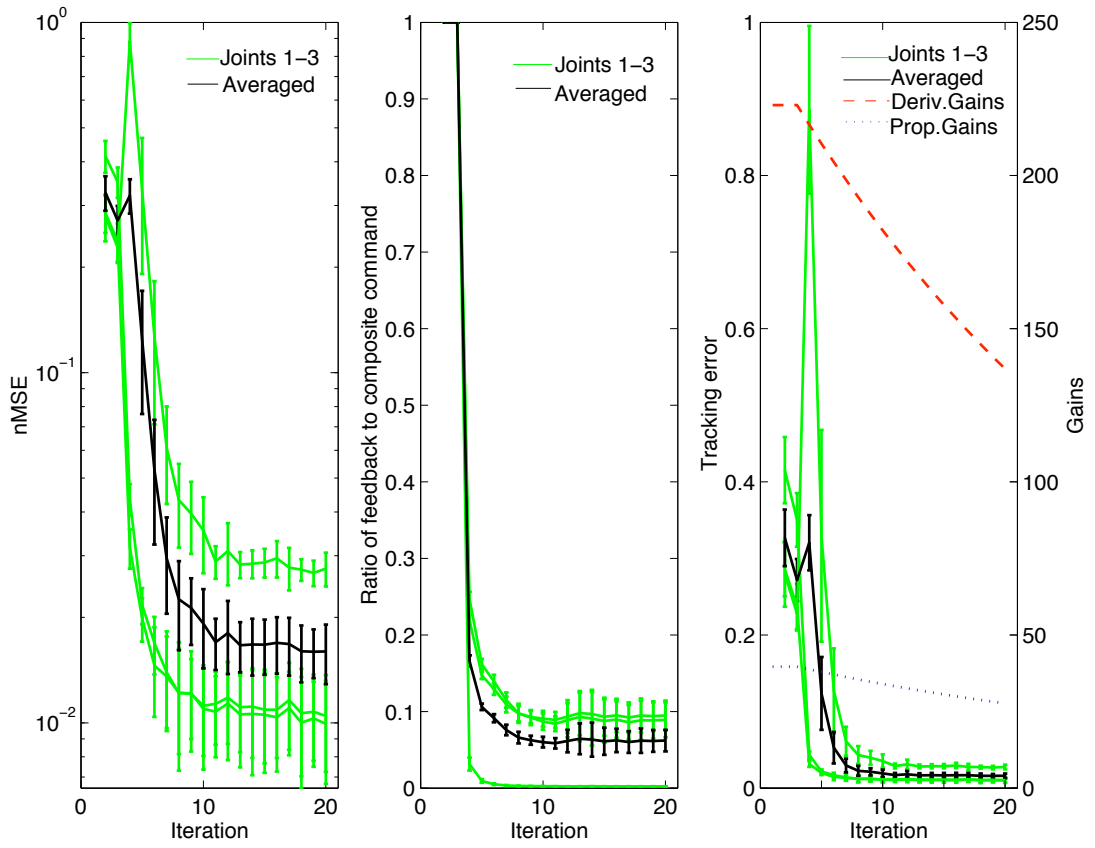


Figure 3.5: Results on learning stationary dynamics of the 3 DOF arm. Results are averaged over ten trials (ten different loads) and the bars indicate the standard deviation between the trials. Left: test error. Middle: contribution of error-correcting feedback command. Right: tracking error.

all trials. Furthermore, as displayed in Fig. 3.5 (middle), the contribution of the error-correcting feedback command to the feedforward command is low while at the same time the tracking error (Fig. 3.5 (right)) is very low. Again, the variance between trials is low for all performance measures. These results indicate that the dynamics model of the real DLR arm has been well approximated by LWPR and that this model can be used for control with excellent performance.

In the described experiments, the exact state of the system was known and the commanded torque was exactly applied. Nevertheless, in most realistic scenarios, applications on real hardware, there will be noise on the sensed states of the system and the commanded torque will not be exactly applied, possibly due to unmodeled actuator dynamics or noise. Noise is an important issue in control and learning. In general it can be expected that with a large amount of data and zero mean and low variance noise, it will be easy to learn the actual dynamics and use it effectively for control. It

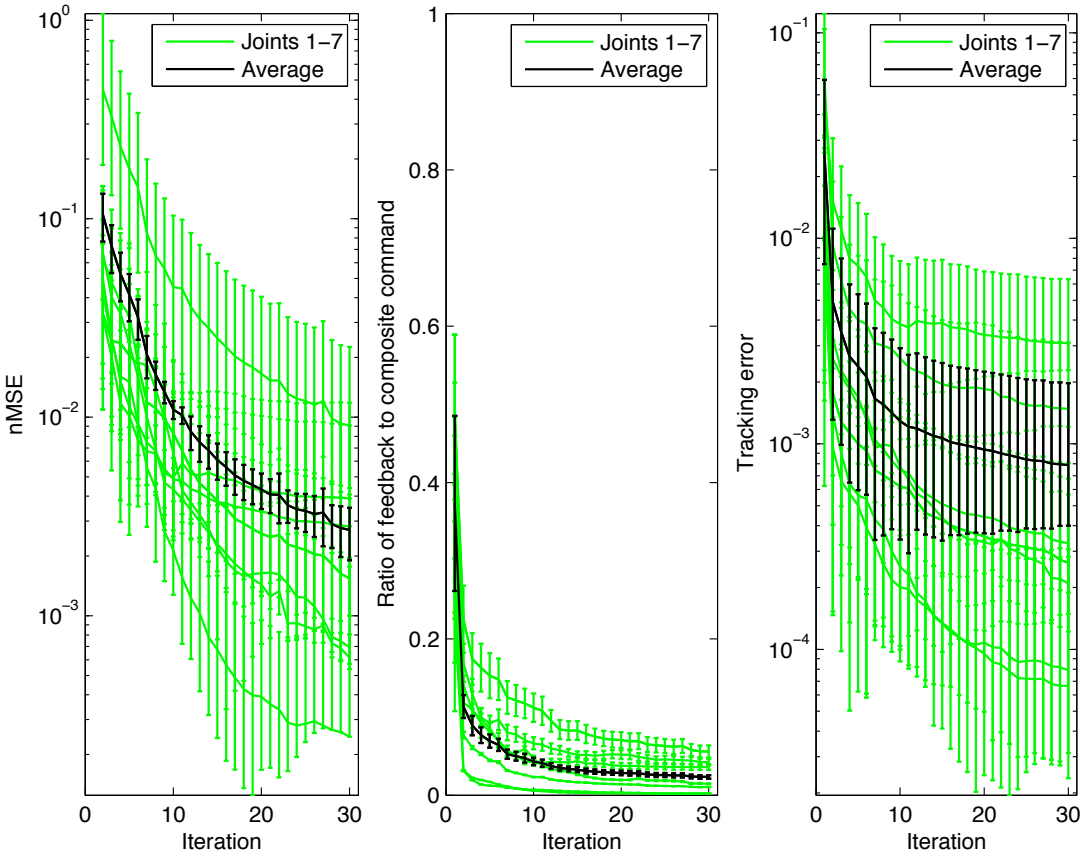


Figure 3.6: Results on learning stationary dynamics of the DLR arm. Results are averaged over six trials (six different loads) and the bars indicate the standard deviation between the trials. Left: test error. Middle: contribution of error-correcting feedback command. Right: Tracking error.

can also be expected that both learning and control performance would degrade as the noise level increases.

Furthermore, it should be noted that the actual benefits of learning dynamics for control cannot be fully appreciated on simulation, as the simulation simply implements rigid body dynamics. The dynamics of a real system will be significantly more complex than the simulated dynamics as there are important nonlinear effects that are not simulated such as complicated actuator dynamics or elasticity and the potential benefits of learning will indeed be more apparent in such real world scenarios. Indeed, it has been shown previously that learning of dynamics using LWPR on real world high DOF robotic hardware works very efficiently (Vijayakumar et al., 2005).

3.3 Learning under nonstationarity

In the previous section, it was demonstrated that an inverse dynamics model of a robotic system operating under stationary conditions can be learned with a robust nonlinear regression algorithm and can be used for control. Nevertheless, the focus of this work is on the problems caused by nonstationarity. One approach for dealing with nonstationarity is to use online learning and adapt the model to the novel dynamics. In general, any online learning algorithm can deal with nonstationarity and there are ways to speed up adaptation of online learning methods to nonstationary data by tuning learning rates (Murata et al., 1996). LWPR uses a forgetting factor λ which can be used to tune how much older data contribute to learning of both the local PLS models and the locality kernels. The forgetting factor ranges from zero to one and – counterintuitively – the smaller the forgetting factor is, the more older data is forgotten. The forgetting factor is used in parameter updates of the form:

$$\hat{\theta}^{new} = \lambda \hat{\theta}^{old} + \hat{\theta}^{update} \tag{3.11}$$

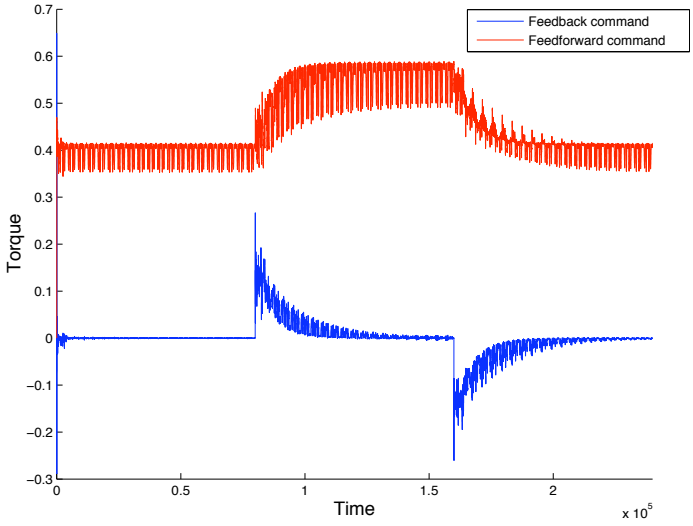
Therefore, the smaller the λ is, the smaller the contribution of the old value of the parameter is and the smaller the effect of older data is. For the exact LWPR equations please see (Vijayakumar et al., 2005).

In this section the effect of nonstationarity is demonstrated on the 3 DOF arm simulation. 20 iterations of the same task are first executed, with the load of the arm being 0.1 kg. Then, for another 20 iterations, the weight of the load is doubled and then finally, the weight of the load becomes 0.1 kg. again. The forgetting factor is set to 0.999. The result is displayed in Fig. 3.7.

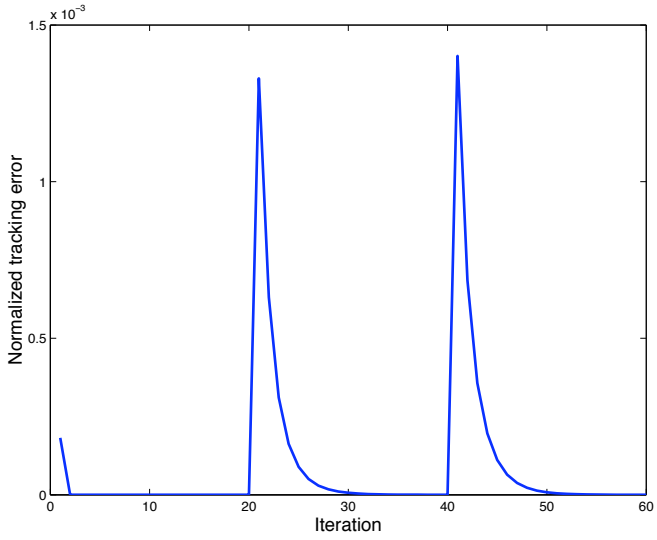
The feedback command is very small initially but when the context changes and the learned model becomes invalid, the feedback part of the controller has to provide significant corrective action. With time, the learned inverse model adapts to the new dynamics and the contribution of the feedback command drops again. However, when the load switches back to the original, the learned model becomes invalid again for a long period of time until the model relearns the initially learned dynamics.

The same experiment was repeated with the forgetting factor set to 0.9999. The result can be seen in Fig. 3.8.

Comparing Fig. 3.7 and Fig. 3.8 it can be seen that setting the forgetting factor to a lower value helps to adapt faster to nonstationary dynamics. It is reasonable to assume that the forgetting factor can be further reduced in order to further accelerate



(a)



(b)

Figure 3.7: (a) Feedforward and feedback command and (b) tracking error while manipulating a nonstationary load using a single dynamics model (the forgetting factor is set to 0.999). The context switches at datapoint 80000 and then switches back to the initial context at datapoint 160000. There is a lag until the feedback command drops to zero after a switch, meaning that the learned model takes some time to adapt.

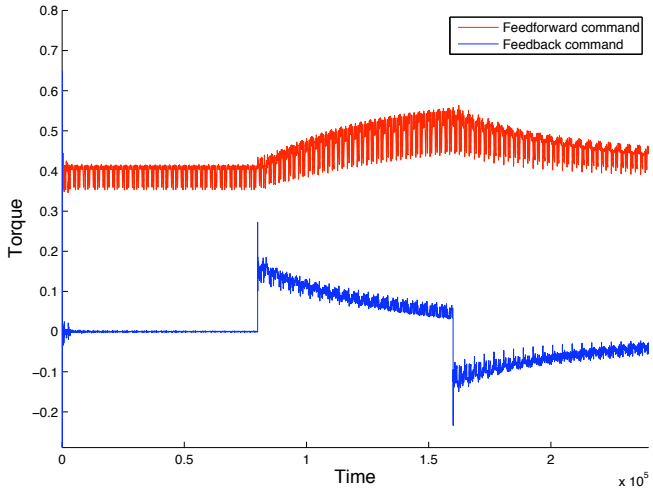


Figure 3.8: Feedforward and feedback command while manipulating a nonstationary load using a single dynamics model (the forgetting factor is set to 0.9999). The context switches at datapoint 80000 and then switches back to the initial context at datapoint 160000. The feedback command decreases very slowly after a switch, meaning that the learned model adapts too slowly.

the adaptation. The same experiment was repeated with the forgetting factor set to 0.99 and the result can be seen in Fig. 3.9. The decrease of the learning rate made learning of the inverse model, as indicated by the feedforward command, unstable. After the middle of the simulation learning fails completely and the feedforward command (i.e. the predictions of the inverse model) becomes too jiggly. This indicates that online adaptation of the model cannot be performed arbitrarily fast using lower forgetting factors.

3.4 Conclusions

In this chapter, it was demonstrated how an inverse dynamics model can be learned from sensed movement data and used for control. It was also demonstrated that when there is a varying context and the learned model needs to be updated to reflect the changing dynamics, there is a drop in performance during the period of adaptation, as reflected in the contribution of the error-correcting feedback commands and the increase in tracking error. A possible solution would be to detect the nonstationarity and reinitialize the model, once the experienced dynamics becomes significantly different from the learned dynamics e.g. by comparing the model predictions with the observed

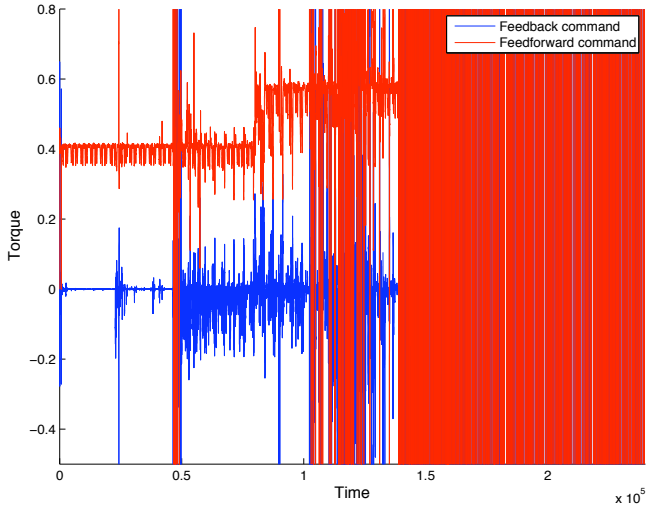


Figure 3.9: Feedforward and feedback command while manipulating a nonstationary load using a single dynamics model (the forgetting factor is set to 0.99). The context switches at datapoint 80000 and then switches back to the initial context at datapoint 160000. Learning of the inverse model is not stable due to the low forgetting factor.

system behaviour or examining the contribution of the feedback command. Nevertheless, this is still not very efficient, as it may take a potentially long period to relearn the previously learned dynamics. In the next chapters, we will be looking at ways to speed up adaptation of the dynamics model by *reusing* the knowledge obtained by learning the dynamics of previously contexts. We start in the next chapter by examining the case that the dynamics may switch between a set of previously seen contexts.

Chapter 4

Learning multiple models for discrete hidden context

As already presented, nonstationarity of dynamics due to interaction with varying environments (or contexts) poses substantial difficulties to the problem of motor learning: the dynamics model needs to be adapted everytime the context switches. Adaptation to varying dynamics may be a lengthy process, especially in the case that local models are used for modeling the dynamics. The performance of the controller will be reduced during the period of readaptation. If different contexts recur, a far more effective strategy is to learn a set of models, each of which is appropriate for a different mode of the nonstationary dynamics and switch between them appropriately.

It should be noted that the idea of multiple models has been considered in human motion studies and there is experimental evidence (Flanagan et al., 1999) indicating that the Central Nervous System (CNS) uses a set of models and switches between them accordingly. There are also studies using functional magnetic resonance imaging (fMRI) that indicate that indeed, models of different tools are spatially separated in areas of the brain (Imamizu et al., 2003). There is, however, another study that provides experimental evidence that the CNS does not use multiple models but instead uses a single model under different perturbations (Karniel and Mussa-Ivaldi, 2000).

The use of multiple models for tackling switching or multimodal problems in machine learning has already been presented in Chapter 2. This chapter builds on that material and examines the use of switching multiple models for control. Previous approaches on the use of multiple models for motor control are presented first. Subsequently, a probabilistic formulation of the multiple model scenario is proposed and based on that, issues like context estimation and model learning are discussed. Finally,

the proposed formulation is tested experimentally.

4.1 Multiple model approaches for control

4.1.1 Early approaches

One of the first approaches for using a set of models for control in robotics comes from (Gomi and Kawato, 1993). A composite controller is used, i.e. the applied command of the system is the sum of a feedforward u_{ff} and a feedback u_{fb} command. The feedforward command is given by a mixture of experts model as:

$$u_{ff} = \sum_{i=1}^{K} g_i u_i \tag{4.1}$$

where u_i is the output of the i^{th} inverse model and g_i is the output of the gating network that determines the responsibility of each model to the current dynamics. The gating network can either take as input time delayed state transitions and commands or other sensory information that may allow to determine which model is appropriate for each case, e.g. visual information. In practice, visual information may be useful but sometimes deceiving, as environments or objects with different physical properties may provide similar or the same visual clues. The difference of this gating network compared to the original mixture of experts formulation, is the addition of inputs that provide more information about the context and not only the same input that is fed to the individual expert networks (inverse models). The model (gating network and inverse models) is trained by gradient descent on a likelihood function.

A very similar approach has been proposed in (Cacciatore and Nowlan, 1994). The system is termed "mixtures of controllers" and again is a mixtures of experts variation. The difference with the model of (Gomi and Kawato, 1993) is that the gating network does not take as input time delayed state transitions and commands or other sensory information. Instead, the gating network is a function of the inverse models' input, (state transitions) and its own previous output. That is, the gating network is:

$$g^t = f(\theta_t, \theta_{t+1}, g^{t-1}) \tag{4.2}$$

The model is again trained by gradient descent on a likelihood function.

4.1.2 Multiple models switching and tuning

More work on the use of multiple models for nonstationary control tasks has been done by Narendra: (Ciliz and Narendra, 1996; Narendra and Balakrishnan, 1997; Narendra and Xiang, 2000). Narendra terms his system Multiple Model Switching and Tuning (MMST). MMST consists of a set of pairs of models and controllers (inverse models or non-model based controllers) corresponding to different modes of operation. The structure of the models is assumed to be known and the parameters – that are assumed to appear linearly in the dynamics – are appropriately placed in the space of possible dynamics. The nonstationary system is controlled by computing a cost function for each context and switching to using the one with the smallest cost. There are various possible choices for the cost. Let the prediction of the i^{th} forward model be y_i, then using the error $e_i = y - y^*$, some cost functions that have been used are:

$$J_i(t) = e_i^2(t) \tag{4.3}$$

$$J_i(t) = \int_0^t e_i^2(\tau)d\tau \tag{4.4}$$

$$J_i(t) = \alpha e_i^2(t) + \beta \int_0^t e_i^2(\tau)d\tau \tag{4.5}$$

$$J_i(t) = \alpha e_i^2(t) + \beta \int_0^t e^{-\lambda(t-\tau)} e_i^2(\tau)d\tau \tag{4.6}$$

$$\tag{4.7}$$

The cost function determines the importance of newer versus older measurements and different cost functions are more appropriate for different cases. For example, the cost function (4.3) takes into account only the current accuracy of the models to predict the current mode of the system and is more appropriate for systems that may switch faster. However, it is not suitable when there is noise in the measurements or inaccuracies in the models. On the other hand, the cost function (4.4) takes into account the long term errors and is more appropriate for systems that may involve noisy measurements than (4.3). Nevertheless, this cost function would detect a switch with some lag and would not be suitable for systems that switch fast. It would be suitable for systems with noisy measurements that may switch slower. A combination of the cost functions (4.5) and (4.6) are the cost functions (4.5) and (4.6), which weigh the short and long term contributions by corresponding factors α and β. Thus, they can be tuned to better deal with situations with different levels of measurement noise and frequency of switching. The cost function (4.6) also involves a forgetting factor λ which determines the length of the memory of the long term error contribution to the

cost and therefore provides another way to tune the system according to how fast the dynamics switch.

The controlled system's dynamics may be linear or nonlinear, the adaptive parameters though, always appear in the models linearly and are adapted by a linear adaptive control law. Models in MMST are grouped in two different classes: adaptive and fixed. Depending on the class of the models, there are different kinds of MMST systems:

- All N models fixed.
- All N models adaptive.
- $N-1$ models fixed and one model adaptive.
- $N-2$ models fixed and two models adaptive.

MMST has been shown to result in a stable controller and has been shown to greatly improve control performance under nonstationary dynamics. However, there is no learning of the initial models (i.e. the models' parameters, since the form of the models is assumed to be known) and prior knowledge of the possible dynamics is required: to achieve good performance the models must be placed in areas of the context space that dynamics appears frequently or otherwise properly optimized, especially in the case that all models are fixed. This can still be an important issue in the case that all or at least one model are adaptive but not as much as when all models are kept fixed. The motivation for MMST is to improve control performance whereas the motivation for this thesis is not only to improve control performance, but the focus is also on learning (knowledge of the structure of the dynamics is not assumed) and as mentioned there is essentially no learning in MMST, or little if linear parameter adaptation can be referred to as learning. An example of the application of MMST to a varying load manipulation task is presented in (Ciliz and Narendra, 1994). Furthermore some experimental results on tuning a MMST system can be found on (Karimi et al., 2001).

4.1.3 Multiple paired forward and inverse models

Another approach is the Multiple Paired Forward and Inverse Models (MPFIM) (Wolpert and Kawato, 1998). As the name suggests, MPFIM uses pairs of forward and inverse models. Inverse models are used for control, whereas forward models are used for context estimation. Let $\hat{\theta}_k^t$ be the prediction of the k^{th} forward model for the

state of the system at time t and θ^t be the true state of the system at time t. Then, a responsibility signal for each context is calculated using a softmax function as:

$$\lambda_k^t = \frac{e^{-|\theta^t - \theta_k^t|^2}/2\sigma^2}{\sum_{j=1}^{K} e^{-|\theta^t - \theta_j^t|^2}/2\sigma^2} \tag{4.8}$$

Here, σ is a scaling constant that plays a similar role as the competition coefficient in Annealed Competition of Experts and the noise variance in conditional mixture models. The responsibilities sum to one and the closest the responsibility is to one the more likely it is that the corresponding context is the current.

Learning in MPFIM is controlled by the responsibility signals. All models, both forward and inverse are trained with all data using gradient updates weighted by the respective responsibilities. That is, the change of parameters w_k of the k^{th} forward model is given by:

$$\Delta w_k^t = \varepsilon \frac{d\hat{x}_k^t}{dw_k^t} \lambda_k^t (\theta^t - \hat{\theta}_k^t) \tag{4.9}$$

A similar update is used for learning the inverse models that are used for control. Models that predict better will be updated, whereas models with poor predictive performance will learn less.

The responsibility signals are also used for control. The applied feedforward command u^t is the weighted average of the predictions given by the individual inverse models $\hat{u}_k^t$ as:

$$u^t = \sum_{k=1}^{N} \lambda_k^t \hat{u}_k^t \tag{4.10}$$

A mechanism for detecting contexts prior to actual movement execution is also proposed. Assume the context is reflected to some other sensory input, e.g. vision. Then another regressor $\hat{\lambda}_k^t$ is trained with the visual data as input and the responsibility predictions as output. Gradually, $\hat{\lambda}_k^t$ should learn to predict the responsibilities λ_k^t prior to movement execution and could be used to initiate movement execution.

Since the estimated responsibilities $\hat{\lambda}_k^t$ are available prior to movement execution and sum to 1, they could be interpreted as the prior probabilities of each context. Also, the responsibilities λ_k^t could be interpreted as the likelihood of each context, since they are available after movement execution. These could be then combined using Bayes rule to obtain posterior responsibilities for the contexts $\bar{\lambda}_k^t$ as:

$$\bar{\lambda}_k^t = \frac{\hat{\lambda}_k^t \lambda_k^t}{\sum_{j=1}^{N} \hat{\lambda}_j^t \lambda_j^t} \tag{4.11}$$

These posterior responsibilities can then be used for learning and control instead of λ_k^t.

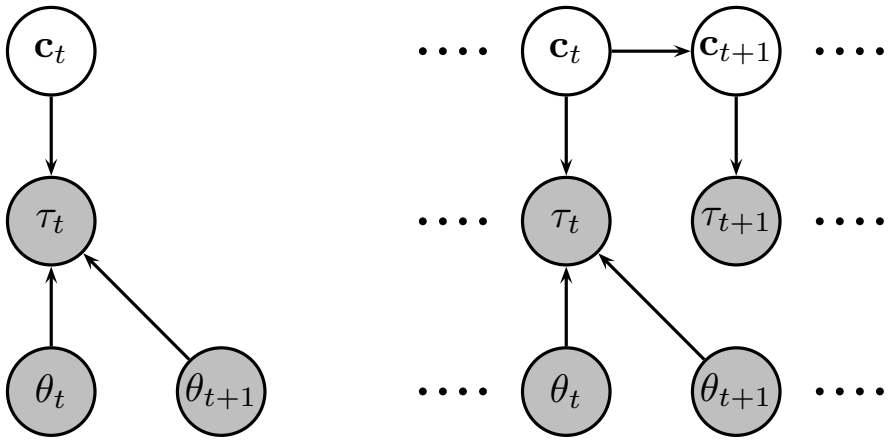
MPFIM resembles the system in (Gomi and Kawato, 1993), the difference being the absence of the gating network. MPFIM allows mixing (instead of switching as in MMST) between models and is thus supposed to be able to handle a whole range of contexts, instead of just a set of contexts. Nevertheless, no justification why the predictions of individual models can in general be weighted in a linear way in order to deal with novel contexts is provided. The first experimental results for MPFIM were presented in (Haruno et al., 1998), where models of a system with nonstationary but linear dynamics are learned and used for control. The ability to generalize to new dynamics is also displayed; however, this is fairly trivial since the dynamics is linear.

4.1.4 Modular selection and identification for control

Modular Selection and Identification for Control (MOSAIC) (Haruno et al., 2001) is an extension of MPFIM. It tackles one of the major difficulties in MPFIM, which is setting of the competition parameter σ. MOSAIC also introduces a temporal dependency on the context evolution and forms a Hidden Markov Model in order to improve context estimation and learning of the models. This is in accordance to findings (Vetter and Wolpert, 2000) in humans that demonstrate that the CNS bases its context predictions both on the accuracy of the predictions of the internal models and on prior knowledge about the time evolution of the context. Also, a likelihood cost function is introduced and the system is learned with the Expectation Maximization algorithm that estimates the dynamics models, the transition probabilities and the noise estimate σ. This model presents an important improvement over MPFIM since it incorporates context dynamics and uses a far more efficient learning method. MOSAIC is very similar to the approach taken in this thesis. Nevertheless, MOSAIC has again only been tested with a system with linear dynamics. It should also be noted that there are fMRI studies (Imamizu et al., 2004), indicating that the human CNS utilizes a MPFIM / MOSAIC-like model, rather than a mixture of experts like model. That is, different context models are separated but cooperate to guide the selection of model to be used by the CNS, rather than utilizing a separate single gating network that would provide the context estimates.

4.2 Formulating multiple models probabilistically

In what follows, a probabilistic formulation, based on the conditional mixture model will be presented. The probabilistic formulation is chosen so that uncertainty (inherent in modeling) is taken into account. The probabilistic approach also has the benefit that the EM algorithm, which is usually far more efficient than gradient methods, can be used for learning.



(a) (b)

Figure 4.1: (a) A graphical model representing a set of multiple models. The discrete hidden variable c_t represents the context and indexes the set of models. (b) Introducing a temporal relationship $p(c_{t+1}|c_t)$ on the hidden variable.

The graphical model in Fig. 4.1(a) represents a set of inverse models corresponding to a specific number of contexts and is an instance of the conditional mixture model with the input being the state transition. The hidden contextual variable c_t is discrete and indexes the different models. A similar graphical model can be formulated for the forward model. When learning a joint space inverse model for non-redundantly actuated arms, the inverse dynamics has a unique solution, i.e. is not multivalued and can thus be used for detecting the context. In the case that there is redundant actuation or the inverse model is learned in operational space, a forward model also needs to be learned and context estimation will have to be based on that. In this study, this scenario will not be considered.

The inverse model in this formulation can be written as:

$$P(\tau \,|\, \theta_{t+1}, \theta_t, c_t = k) = \mathcal{N}\left(g_k(\theta_{t+1}, \theta_t),\ \sigma_k(\theta_{t+1}, \theta_t)\right), \tag{4.12}$$

where g_k is the command predicted by the LWPR model corresponding to the k^{th} context and σ_k is some estimate of the variance, which can be either set to a maximum likelihood estimate or based upon the input dependent confidence bounds provided by LWPR. Also, if there is no knowledge about the prior probability of contexts, we can assume a non-informative prior.

4.2.1 Context estimation

Under this probabilistic formulation, context estimation is just using Bayes rule to infer the posterior of c_t given a state transition and the command that resulted in this transition:

$$P(c_t = k \,|\, \theta_t, \theta_{t+1}, \tau_t) \propto P(\tau_t \,|\, c_t = k, \theta_t, \theta_{t+1}) P(c_t = k). \tag{4.13}$$

4.2.2 Introducing temporal dependency of contextual variables

Context estimates are very sensitive to the accuracy of the inverse models and the possibly noisy states and commands. They can be improved by acknowledging that contexts do not change too frequently. Similarly to the MOSAIC model, a temporal dependency between contexts $p(c_{t+1} \,|\, c_t)$ with an appropriate transition probability between contexts can be introduced to achieve much more robust context estimation. The graphical model can be reformulated as the Dynamic Bayesian Network shown in Fig. 4.1(b) to achieve this. A low transition probability penalizes too frequent transitions and using filtering, smoothing or Viterbi alignment produces more stable context estimates. Application of standard Hidden Markov Model (HMM) techniques is straightforward by using (4.12) as the observation likelihood in the HMM, given the hidden state $c_t = k$. The first estimate of interest is the filtered estimate $p(c_t | \theta_{1:t+1}, \tau_{1:t})$. It can be computed from $\alpha(c_t) = p(c_t, \theta_{1:t+1}, \tau_{1:t})$ as

$$p(c_t | \theta_{1:t+1}, \tau_{1:t}) = \frac{p(c_t, \theta_{1:t+1}, \tau_{1:t})}{\sum_{c_t} p(c_t, \theta_{1:t+1}, \tau_{1:t})}. \tag{4.14}$$

The filtered estimate can in turn be computed recursively in a forward recursion, using the next observation θ_{t+2} (state) and τ_{t+1} (command) to obtain the estimate at the next time step $\alpha(c_{t+1}) = p(c_{t+2}, \theta_{1:t+1}, \tau_{1:t+1})$ as:

$$\alpha(c_{t+1}) = \sum_{c_t} \alpha(c_t) p(c_{t+1} | c_t) p(\tau_{t+1} | \theta_{t+1}, \theta_{t+2}, c_{t+1}) \tag{4.15}$$

Given a full control sequence $\theta_{1:T}$ and $\tau_{1:T}$, the smoothed estimated $p(c_t | \theta_{1:T}, \tau_{1:T})$ can also be computed. These estimates are more accurate than the filtered estimates

as they take into account information from the future as well. A backward pass where the distribution $\beta(c_t) = p(\theta_{t+1:T}, \tau_{t+1:T}|c_t)$ is recursively computed from $\beta(c_{t+1}) = p(\theta_{t+2:T}, \tau_{t+2:T}|c_{t+1})$ is first performed as:

$$\beta(c_t) = \sum_{c_{t+1}} \beta(c_{t+1}) p(c_{t+1}|c_t) p(\tau_t|\theta_t, \theta_{t+1}, c_t) \tag{4.16}$$

The quantity $\beta(c_T) = p(\theta_{T+1:T}, \tau_{T+1:T}|c_T)$ that is needed to initialize the recursion is meaningless (since there is no observation at time $T+1$). It can be simply set to 1, which will provide the correct results.

Finally, the forward and backward estimates are combined to provide the smoothed estimates as:

$$p(c_t|\theta_{1:T}, \tau_{1:T}) = \frac{\alpha(c_t)\beta(c_t)}{\sum_{c_t} \alpha(c_t)\beta(c_t)} \tag{4.17}$$

For online control purposes, only filtered estimates can be used, since only data until the current time step will be available. As it will be discussed in the next section, smoothed estimates will be useful for offline learning of models, since the whole sequence of data will be available. It also needs to be mentioned, that for numerical stability reasons, the recursive estimates are computed somewhat differently than the equations given here. Details can be found in Appendix B.

4.2.3 Multiple model learning

In the MPFIM, separation of data for learning the inverse models happens either online or offline. In the online case, the predictions of the models are compared with the observed behaviour of the system to give context estimates and the models are trained using weighted gradient descent, where the context estimates are used as weights. This approach worked well when the learned models were linear but has not been shown to work for nonlinear models. In the offline case, which we are going to discuss, an Expectation Maximization (EM) algorithm is used. This offline procedure, will give relatively accurate (initial) models to bootstrap the context estimation procedure, after which further online data separation can be performed. However, the offline EM proposed by MPFIM and MOSAIC has been shown to work also only for linear models. We will show how it is possible to use the EM algorithm for the described conditional mixture model to do data separation and learn a set of nonlinear models.

The EM algorithm for the conditional mixture model has actually already been developed in Chapter 2. One thing that needs to be clarified is that in the case of the temporal model, the smoothed estimates $p(c_t|\theta_{1:T}, \tau_{1:T})$ are to be used in place

of $p(z_i = k|y_i, x_i)$ in (2.20). Furthermore, since it was decided to use LWPR as the function approximator for the inverse model, a way must be found so that different data that are used to train LWPR contribute differently to training according to their posterior probability. It turns out that simply multiplying the kernel activations by the posteriors yields the required weighing.

Furthermore, learning the transition probabilities from a sequence of observations in the temporal model is straightforward using EM. In particular, the probabilities $p(c_t, c_{t+1} \,|\, \theta_{1:T}, \tau_{1:T})$ for $t = 1...T-1$ need to be calculated (E-step) as:

$$p(c_t, c_{t+1} \,|\, \theta_{1:T}, \tau_{1:T}) = \frac{a(c_t)p(\tau_{t+1}|\theta_{t+1}, \theta_{t+2}, c_{t+1})\beta(c_{t+1})p(c_{t+1}|c_t)}{\sum_{c_t} \alpha(c_t)\beta(c_{t+1})} \tag{4.18}$$

Again, for numerical stability reasons, we compute $p(c_t, c_{t+1} \,|\, \theta_{1:T}, \tau_{1:T})$ using the equations in Appendix B. The relative frequencies $p(c_{t+1} \,|\, c_t)$ for any t can be easily estimated (M-step) as:

$$p(c_{t+1} = i|c_t = j) = \frac{\sum_{t=0}^{T-1} p(c_t = i, c_{t+1} = j \,|\, \theta_{1:T}, \tau_{1:T})}{\sum_{k=1}^{K} \sum_{t=0}^{T-1} p(c_t = i, c_{t+1} = k \,|\, \theta_{1:T}, \tau_{1:T})} \tag{4.19}$$

As usual, the procedure is iterated a few times: i.e. the posteriors $p(c_t|\theta_{1:T}, \tau_{1:T})$ and $p(c_t, c_{t+1} \,|\, \Theta_{1:T}, \tau_{1:T})$ are computed using some values for the transition probabilities (one could initially set all transitions to be equally probable), then $p(c_{t+1} \,|\, c_t)$, the models and the noise are estimated, then $p(c_t|\theta_{1:T}, \tau_{1:T})$ and $p(c_t, c_{t+1} \,|\, \Theta_{1:T}, \tau_{1:T})$ are computed again using the estimated transition probabilities, models and noise and so on until either a maximum number of iterations is reached or some other criterion is met, e.g. the likelihood of the observed data stops increasing.

Our approach differs from MOSAIC (which also used an EM procedure and a HMM) in that it is based on the inverse models only (rather than combined forward and inverse models), accounts for the possibility of nonlinear models and uses a robust local learning algorithm. The use of a local regression method capable of learning nonlinear models poses some important issues. These issues and their solutions will be presented in the experiments section.

4.3 Experiments

4.3.1 Context estimation

The context estimation methods suggested earlier were tested on the simulated 3 DOF arm and the 7 DOF DLR arm. Different contexts are assumed to be manipulated

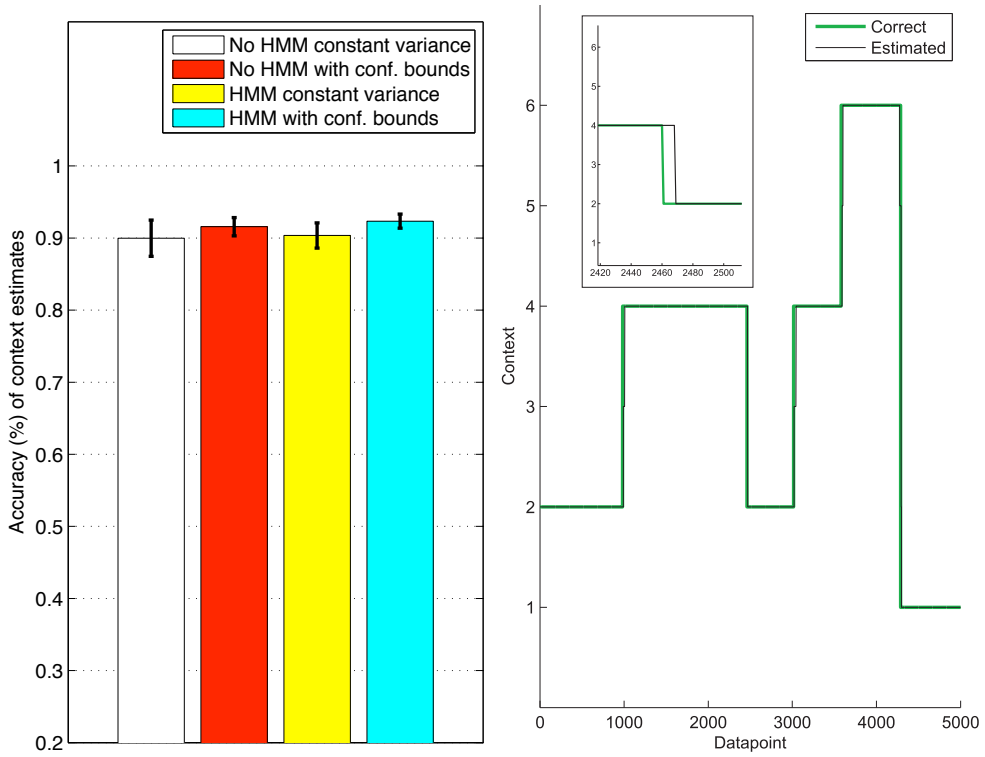


Figure 4.2: Online context estimation without using the context estimates for control for the 3 DOF arm. Left: context estimation accuracy using different estimation methods. Right: example of random context switches and its estimate using HMM filtering over time. A detail of a switch happening at datapoint 2460 is displayed in the inset, where we see that the estimate switch lags a few time steps behind the actual context switch.

objects with different mass. These are simulated by fixing an object with variable mass on the last link of the manipulator.

For the simple 3 DOF arm, random switches between six contexts were performed in the simulation, where at every time step we switch to a random context with probability 0.001 and stay in the current context otherwise. Since control happens at 100 Hz, this means that the context changes every ten seconds on average. This is neither a too fast or too slow switching frequency. We do not want to have too frequent switches, so that the system has time to recover from the inevitable error after a switch. We also want not too sparse frequent switches, so that we observe better how the system detects switches and handles them.

There are some options for the probabilistic model used in the experiments:

- Structure of probabilistic model:

    1. Temporal relationship of the hidden context used (HMM filtering, for the

moment we assume that we know the correct transition. probabilities)

2. Temporal relationship of the hidden context not used.

- Noise model:

1. Using a maximum likelihood estimate for the noise variance.
2. Use the confidence bounds provided by LWPR to estimate the noise variance.

These give four different combinations of options. All four will be tested and for each of them there will be five different trials in order to collect statistics. In each of the trials, we randomly switch between six of the models that were learned in Section 3.2. Each trial's length is ten iterations of the trajectory. We start by examining the accuracy of our probabilistic context estimation methods while not using the context estimates for control (a PD controller was used). The percentage of accurate online context estimates for the four cases, averaged over the five trials, can be seen in Fig. 4.2 (left) (error bars are obtained from the five different trials). The difference in accuracy between methods is not significant. Nevertheless, the methods that use the confidence bounds perform on average slightly better that the methods that use the maximum likelihood estimate for the noise.

Fig. 4.2 (right) gives an example of how the HMM filtering using LWPR's confidence bounds performs when used for online context estimation. As displayed in the inset frame, the estimation of a switch lags behind a few time steps when there are context switches. The number of timesteps before correctly detecting a switch is typically around 10, which corresponds to 0.1 seconds (since control is performed at 100 Hz). Thus, the context switch is detected quite fast after it happens. The presence of this lag is a natural effect of online filtering (as opposed to retrospect smoothing). However, the HMM based methods give more stable estimates between switches compared to the non-HMM based methods.

The context estimates were then used online for selecting the model that will provide the feedforward commands. Fig. 4.3 (left) shows the percentage of accurate online context estimates for the four cases and Fig. 4.3 (right) shows the corresponding contribution of the feedback command. Results are again averaged over the five trials. There is a significant improvement on estimation accuracy when using the HMM based methods over the non-HMM based methods and when using the confidence bounds over the maximum likelihood estimates. The ratio of feedback to composite command is also

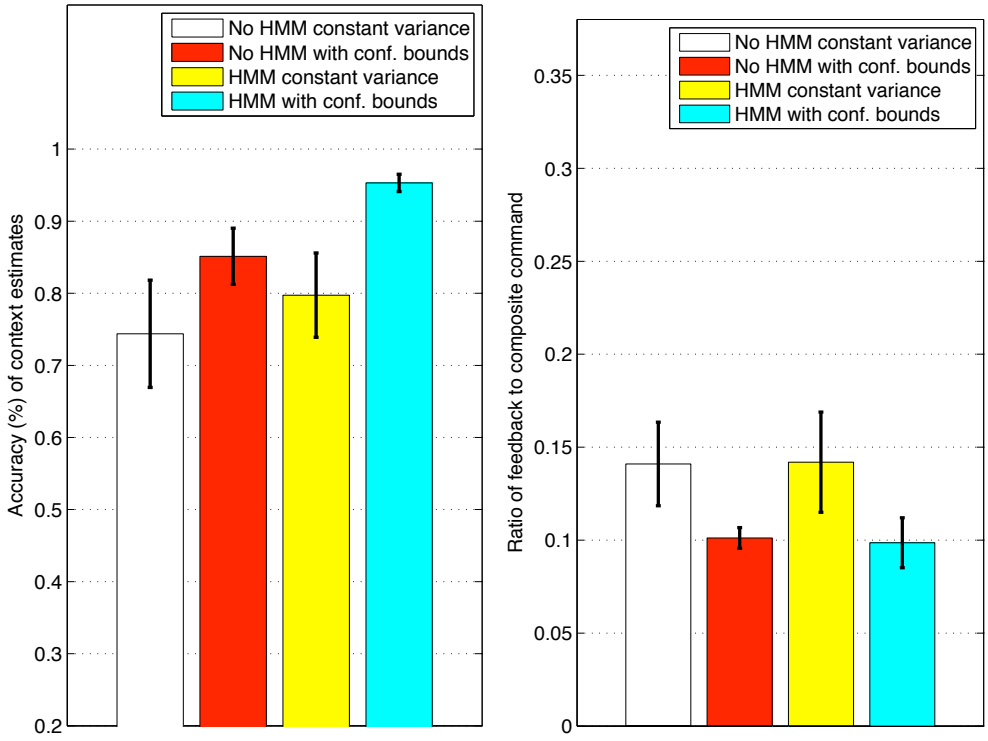


Figure 4.3: Online context estimation using the context estimates for control for the 3 DOF arm. Left: Context estimation accuracy using different estimation methods. Right: Ratio of feedback to composite command using different estimation methods.

comparable to the ratio for the single context displayed in Fig. 3.5, when the confidence bounds are used.

In comparison to the performance of the different methods when the estimates are not used for control, in this more complicated and of practical interest scenario, we can clearly see the benefit of exploiting the temporal relationship of the models and using the input dependent confidence bounds for the noise model.

The reason why the temporal model is beneficial to context estimation has already been discussed. It is also useful to discuss how the use of the confidence bounds improves the context estimation accuracy. In contrast to the maximum likelihood estimate, we expect the input dependent confidence bounds to give larger estimates for the noise variance in areas of the input space that are not well learned than in well learned areas. Thus, $p(\tau \,|\, \theta_{t+1}, \theta_t, c_t = k)$ will be more similar for different k in (4.12) and in combination with the filtered estimates can prevent inaccurate context switching when the predictions of models are not accurate and come close to each other.

In the previous experiments, the learned models were quite accurate. The nMSE of the individual models was about 0.005. The last experiments, where the context

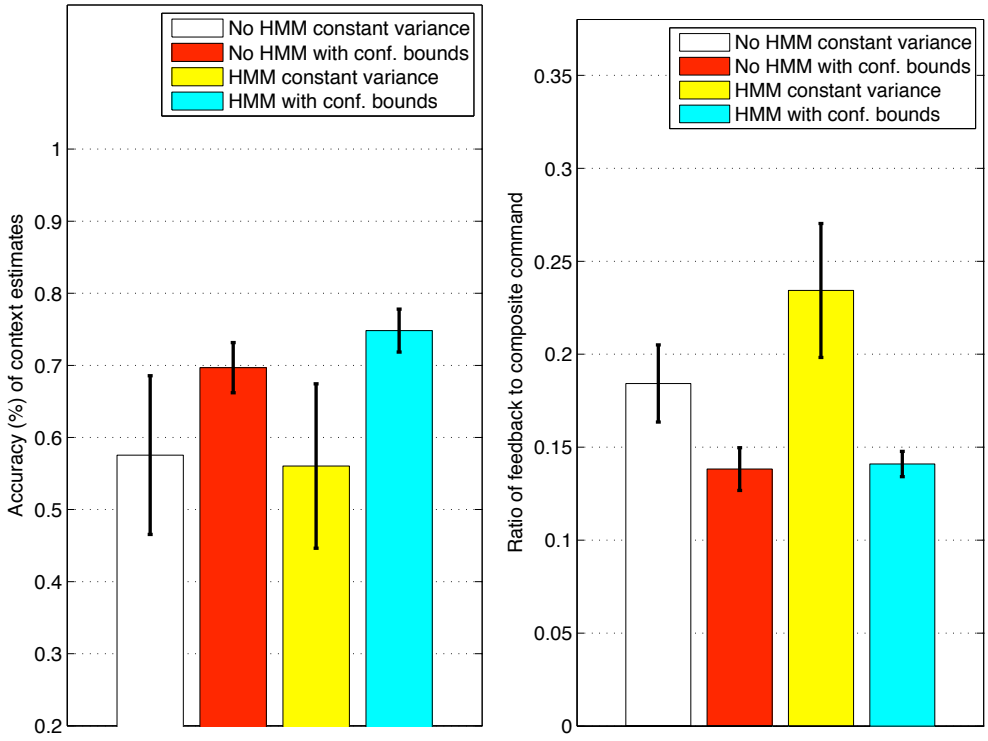


Figure 4.4: Online context estimation using models with reduced accuracy. The context estimates are used to control the 3 DOF arm. Left: Context estimation accuracy using different estimation methods. Right: Ratio of feedback to composite command using different estimation methods.

estimates were used for control, were repeated with less accurate models. This time, the nMSE of the learned models was about 0.05. The results can be seen in Fig. 4.4. Fig. 4.4 (left) shows the accuracy of context estimates and Fig. 4.4 (right) shows the ratio of feedback to composite command for the different context estimation methods. In comparison to the case where more accurate models were used (Fig. 4.3), the performance has clearly dropped: the accuracy of context estimates is clearly lower and the ratio of feedback to composite command is higher for all context estimation methods. However, the benefits of using the confidence bounds and the temporal model are still evident when less accurate models are used, as there is significant improvement in performance when they are used over the other methods.

Context estimation experiments were also performed with the DLR arm simulation. Again, all four options – using the temporal or non-temporal models and a maximum likelihood or confidence bounds based estimate for the noise variance – were tested. For each option we run five different trials to collect statistics. In each of the trials, we randomly switch between five of the models that were learned in Section 3.2 and each

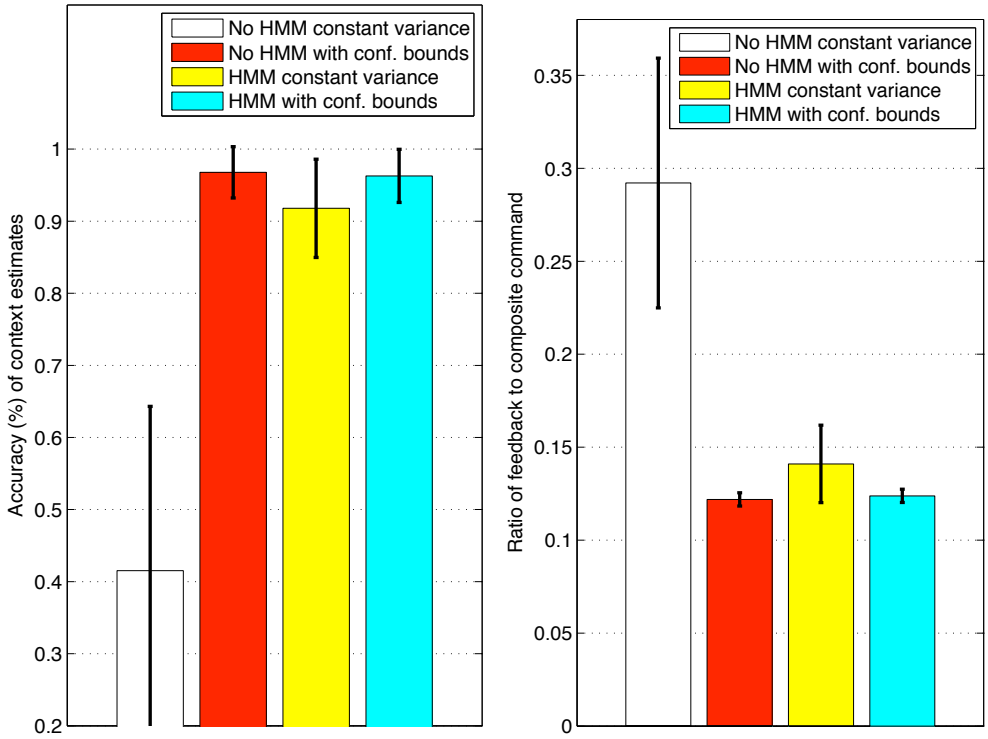


Figure 4.5: Online context estimation using the context estimates for control for the DLR arm. Left: Context estimation accuracy using different estimation methods. Right: Ratio of feedback to composite command using different estimation methods.

trial's length is 20 iterations of the trajectory. Results are presented only for the more interesting case where the context estimates are used for control. As it can be seen in Fig. 4.5 (left) the use of the confidence bounds greatly improves context estimation accuracy. The benefit of using the temporal model is apparent when the maximum likelihood estimate is used for the noise variance, whereas it does not significantly affect the performance when the confidence bounds are used. The estimation accuracy is almost perfect when the confidence bounds are used. These results are also reflected on Fig. 4.5 (right), that displays the ratio of feedback to composite command for the four methods. The lowest ratio is given for the methods using the confidence bounds, again with no significant difference between the non-temporal and temporal models. It is again evident that the performance of the temporal model is far superior to the performance of the non-temporal model when a constant noise variance is used.

It should also be noted that on all experiments, performance depended heavily on the choice of the PD gains. When the PD gains were relatively high, the feedback part of the command would be higher than required and would push the system far from the desired trajectory right after switching. Given that the training data for the

models come from a relatively small area in the dynamics space, the predictions of the LWPR models may not be accurate in novel areas and the context estimates may switch between contexts arbitrarily. When the gains are properly set though, the system will not enter areas where it has not seen data and will thus be able to detect context changes correctly, as happens in the experiments. Possible solutions to this problem could be the use of global learning algorithms and more extensive exploration of the input space. Furthermore, the use of the temporal model and the confidence bounds seems to alleviate this problem.

In addition, not only the gains have to be chosen carefully, the models also have to be sufficiently accurate for the performance of the multiple model approach to be satisfactory. We purposefully tried to approximate the nonlinear dynamics of the DLR arm with too smooth dynamics (i.e. we used a single very wide LWPR kernel). The learned dynamics would roughly approximate the true dynamics (nMSE 0.09) and could potentially be used for control under stationary conditions, but would be useless for control under switching dynamics: the learned dynamics of one context could be similar to the true and more accurately learned dynamics of other contexts and could thus result in too frequent (and incorrect) switching between models. These have knock-on effects during control including increased tracking error, increased feedback correction and consequently, overshoot into regions of the state space that were previously unexplored. As it has been shown, the use of the temporal model and the confidence bounds does indeed help with these issues but may be unable to compensate for models that are highly inaccurate.

It is also necessary to note that the similarity of contexts is important. In general, the more similar the dynamics of different contexts are, the more likely it will be that the predictions of models become entangled and will therefore be difficult to distinguish. It is also clear that the more the contexts differ, the less likely it will be that inaccuracies of the learned models will be significant for context estimation. In the previous experiments, the contexts and the corresponding learned models were fairly different. In the experiments with the 3 DOF arm, the loads had at least 0.2 Kg difference in mass and in the experiments with the DLR arm the difference was at least 0.1 Kg (the models from Section 3.2 have been used).

In conclusion, the use of multiple models can improve performance under non-stationarity but should be done with care. A bad choice of gains and very inaccurate models can cause increase rather decrease of tracking errors for long periods. Thus, there is an issue with stability and in order to benefit from the use of multiple models,

there should be sufficient experimentation with gains and learning of models should be as accurate as possible.

4.3.2 Data separation and learning

Next, we investigate bootstrapping of data separation (and model learning). We will use the temporal model as it has been shown to give far superior results for context estimation and control and also investigate learning of transition probabilities.

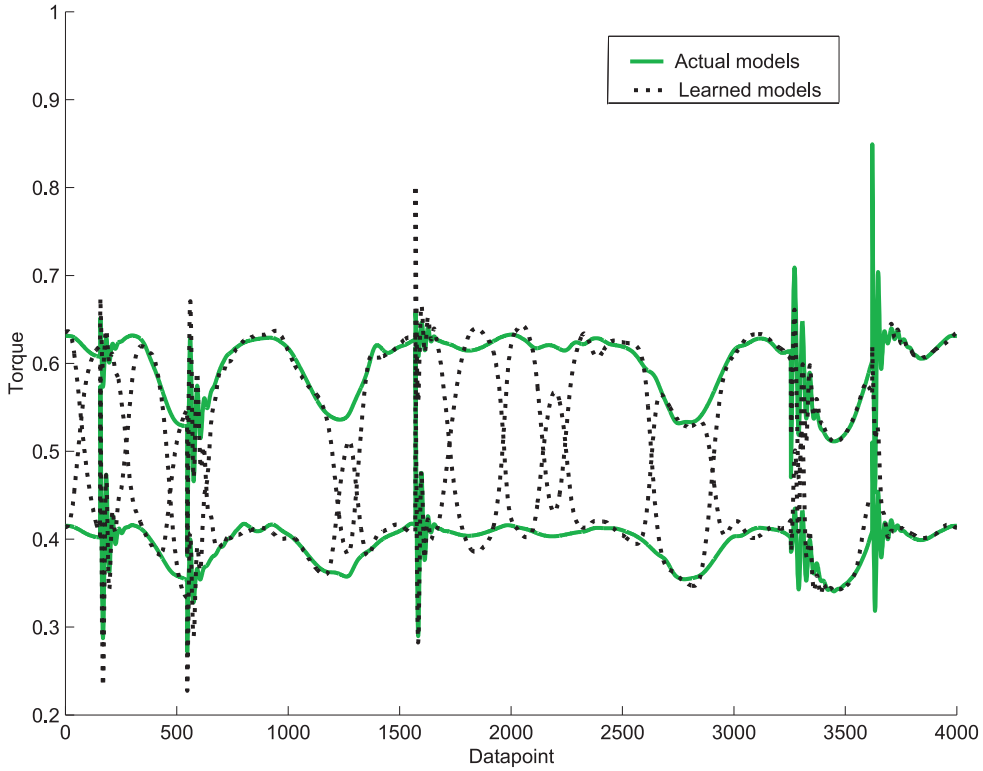


Figure 4.6: The solid lines show the predictions of the inverse models for the first joint on the training data, if the models had been trained with perfectly separated data. The dotted lines show the predictions of the models generated by the automated separation procedure. Data separation seems to work locally but not in the whole input space. That is, in small areas of the input space, for which a single local linear model from each LWPR model is responsible, data can be separated well between the models and each LWPR model seems to specialize in one context. Nevertheless, the local models from one LWPR model may specialize in different contexts across the input space.

Experiments were performed on the 3 DOF arm. Again, when generating the data, we switched between two different contexts with probability 0.001 at each time step (since, as mentioned earlier, we do not want too frequent or too sparse context

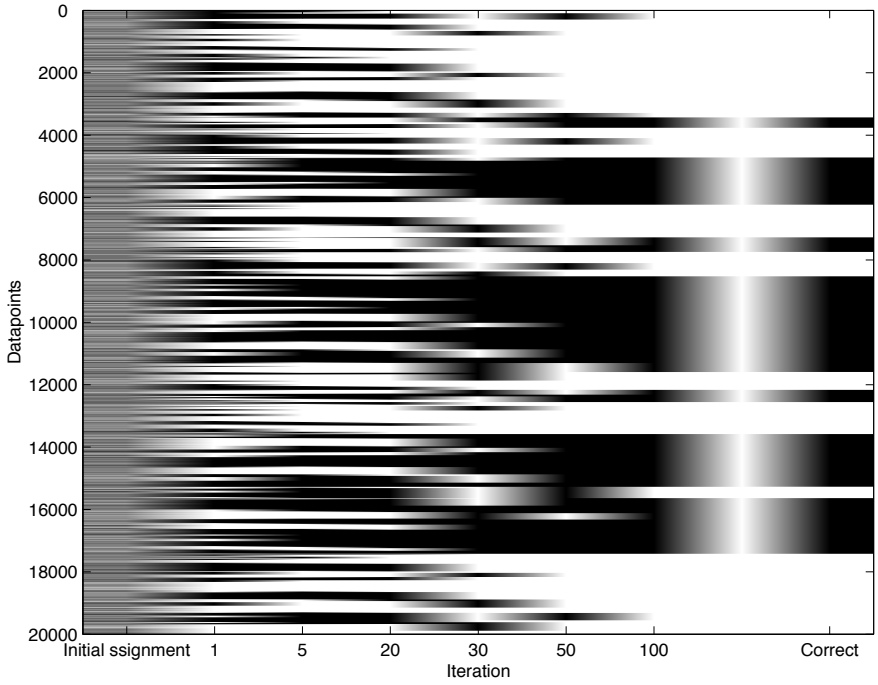


Figure 4.7: The evolution of the data separation from unlabeled data over some iterations of the EM procedure. The first column displays the initial random assignment of datapoints to contexts. The last column displays the correct context for each datapoint. The columns in between display the most likely context for each datapoint according to the currently learned models for some iterations of the EM procedure.

switches); however, we now do not use the correct transition probabilities in either inference (E-step) or learning (M-step). We first collected a batch of context unlabeled data from five cycles through the target trajectory where the arm was controlled by pure feedback PD control. The EM procedure for data separation and learning of transition probabilities (Section 4.2.3) was applied. Using the confidence bounds for the noise estimates of the observation (inverse) model from the beginning of the EM procedure did not work and made all models collapse into the same model. This is because the confidence bounds gave very large values for the noise variance. Increased noise variances make the posteriors of all models roughly equal, resulting in all data contributing equally to the training of the models. Eventually, after some iterations, all posteriors become equal and all models end up being the same. What is needed is harder division of data between models. This can be achieved by having smaller noise variance. Indeed, using the maximum likelihood estimates works much better. However, this still does not give very good results. The problem lies in the fact that a local learning method is used: data seem to be separated correctly locally but not globally,

across the input space. This is demonstrated in Fig. 4.6. In small areas of the input space, where a single local linear model from each LWPR model is mostly responsible for, data separation works fine. However, the grouping between local models is almost random and depends mostly on the initial random assignment of datapoints to contexts. In some cases, data coming from relatively larger areas of the input space are separated correctly. That is, the local models in these areas are by chance grouped correctly, but it is rather unlikely that this will happen across the input space. A possibility would be to try to regroup the local models to maximize the smoothness of the learned model, however, we show here that it is possible to achieve perfect data separation by modifying the EM algorithm slightly.

Improving data separation by using the confidence bounds

We note that if we manage to increase the volatility in the areas where there is mixing between local models that actually belong in different contexts, the posterior of the datapoints in that area will switch from one context to the other slower and therefore, will reduce the mixing effect. This increase of volatility can be achieved using LWPR's confidence bounds: the confidence bounds increase when there is a sudden change in the model's prediction. Thus, we run the EM procedure using a maximum likelihood estimate for the inverse model noise until the data is well separated locally and then switch to using the confidence bounds. This amendment has been sufficient to solve the problem but with some exceptions. If at the point that we start to use the confidence bounds, there are very large or too many areas that need to be swapped between LWPR models, then the procedure may still get stuck.

Shrinking areas that have been incorrectly separated

The way to solve this problem is to switch from using smoothed estimates in the E-step to using filtered estimates. This effectively, together with the increased noise levels on the edges of the areas that need to be swapped, makes the areas that are not grouped correctly narrower and narrower in each iteration of EM. This effect is due to the fact that, as discussed earlier, detection of context switches with filtered estimates, lags a few time steps behind. Therefore, the posteriors near the unnecessary switches, that are due to inaccurate grouping of local models, will reflect the correct grouping for a while and help to train the models with the correct data.

This modified EM procedure was tried with perfect data separation always being achieved. Fig. 4.7 displays a typical evolution of the data separation. Switching to using the confidence bounds and the filtered instead of the smoothed estimates happens at iteration 20. The final algorithm is summarized in Table 4.1.

The transition probabilities are also estimated during the EM procedure. The estimates were very close to the actual ones: the estimated probability is very close to the actual ones, i.e. 0.999 staying in the same context and 0.001 switching.

Furthermore, the effect of using the wrong number of models has been examined. The same experiment with the same two switching contexts was executed; however, three models were used instead of two. It was found that the redundant model ruined the data separation procedure. As it can be seen in Fig. 4.8 (a) the data has been separated between all three models instead of two. As a first solution, it was attempted to do the competition between the models harder by setting a lower limit on the posterior for training the models. That is, if the posterior for some model was lower than the threshold, the datapoint would not be used for training the model. Setting the threshold to 0.05, the result in Fig. 4.8 (b) was obtained. Adding the threshold yielded the desired result and after some point, the redundant model was not trained with any data and eventually disappeared.

This could lead to the conclusion that it is reasonable to underestimate the number of required models and use this thresholding or switch to hard assignment after some point to deal with the problem of lack of knowledge of the number of contexts. Another approach could be to use some principled model selection procedure. Model selection works in general by computing the likelihood of the observed data D, $p(D|\theta)$ where θ are the parameters of the model for the different models under consideration, including a model specific penalty term. Examples include the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC) and Maximum a Posterior selection (MAP). For example, in MAP a prior on the model parameters $p(\theta)$ can be set and used to find the value for the parameters θ^{MAP} that maximize the posterior distribution of the parameters:

$$\theta^{MAP} = \arg\max_{\theta} \log p(D|\theta) + \log p(\theta) \tag{4.20}$$

The prior should be set accordingly and in general, it should give higher probability to simpler models, reflecting the prior belief that simpler models are more likely to explain data.

Table 4.1: Pseudocode for separating offline a set of n datapoints coming from m contexts

```
set up simulation environment;
for t=1:T
        control arm using feedback control;
        collect t^th movement datum;
        switch context with some low probability;
end

for t=1:T
        assign randomly the t^th datapoint to a random model and train the model;
end
set the maximum likelihood noise estimate to some small value for all models;

//First part of EM
for iteration=1:em_iterations_part_1
        compute the smoothed posterior p(c_t=k|θ_1:T,τ_1:T) of each datapoint t
        using the maximum likelihood noise estimate;
        train each model k with each datapoint t,
        weighing its contribution by p(c_t=k|θ_1:T,τ_1:T);
        compute the maximum likelihood noise estimate;
end

//Second part of EM
for iteration=1:em_iterations_part_2
        compute the filtered posterior p(c_t=k|θ_1:t+1,τ_1:t) of each datapoint t
        using the confidence bounds for the noise variance;
        train each model k with each datapoint t,
        weighing its contribution by  p(c_t=k|θ_1:t+1,τ_1:t);
end
```

4.4 Discussion

An approach for dealing with recurring, switching nonstationary dynamics has been presented. It differs from previous multiple model approaches in a few points:

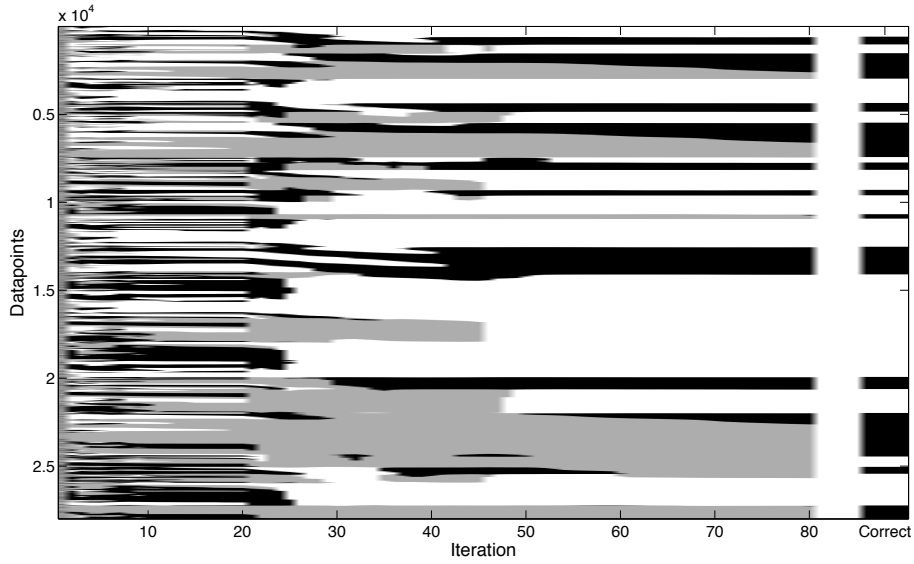
1. Comparing it to the approach of (Gomi and Kawato, 1993) and (Cacciatore and Nowlan, 1994), a fully probabilistic formulation and an EM algorithm for learning has been used. Also, no gating network is used but context estimation is based on the predictions of the models like in MPFIM and MOSAIC.

2. Comparing it to MMST, no prior knowledge on the structure of the dynamics is used. MMST assumes dynamics whose structure is known and whose unknown

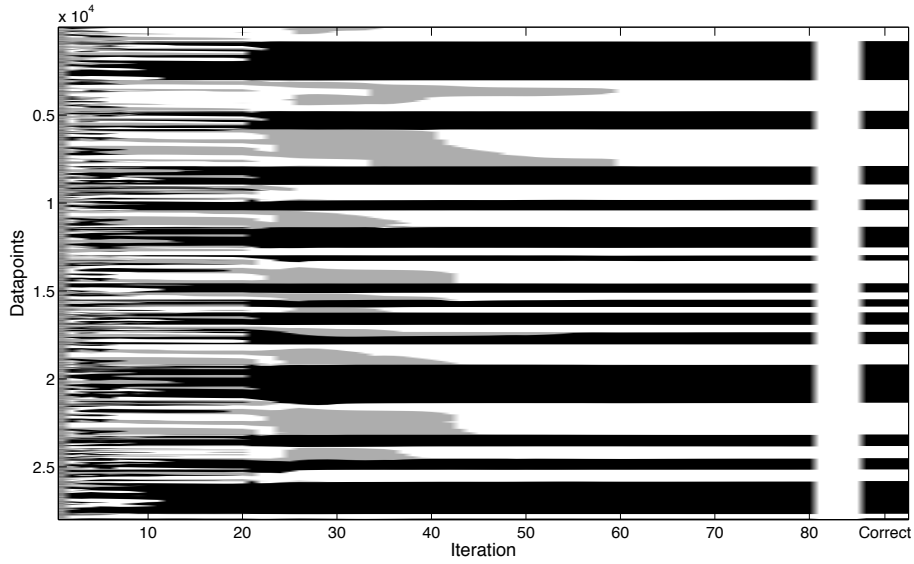
parameters appear linearly in the equations. In our approach, no knowledge of the functional form of the dynamics is assumed to be available.

3. The system is most closely related to MPFIM and MOSAIC. The differences are that no pairs of forward and inverse models are used, only inverse models are learned and context estimation is performed only based on that. Furthermore, nonlinear dynamics models are learned and a local regression algorithm is used for learning the models.

In short, it has been shown in this chapter that it is possible to learn multiple models of *nonlinear* dynamics using *no prior knowledge on the structure of the dynamics*. Furthermore, it was shown that this can be achieved without needing to learn a pair of forward and inverse models for each context (when the inverse model is uniquely defined). In addition, learning of the models was performed using LWPR, a state of the art local learning algorithm that is suitable for learning dynamics models of complicated systems. The use of a local learning method has been shown to pose some important difficulties in learning the models from context unlabeled data: the data is easily separated between the local models of different LWPR models that are responsible for some area of the input space, but the local models are not grouped correctly in LWPR models. This problem has been tackled by modifying the EM algorithm slightly. The confidence bounds have been used to increase competition between LWPR models in areas where they are not smooth (the areas where local models are incorrectly grouped) and filtered instead of smoothed estimates where used in the E-step in order to shrink the incorrectly grouped areas.



(a)



(b)

Figure 4.8: The evolution of the data separation from unlabeled data over 80 iterations of the EM-procedure using the wrong number of models: two contexts are present but three models are used. The first column displays the initial random assignment of datapoints to contexts. The last column displays the correct context for each datapoint. The columns in between display the most likely context for each datapoint according to the currently learned models for some iterations of the EM procedure. In (a) the redundant model destroys the data separation procedure, whereas in (b) a threshold has been set on the posterior probabilities for training a model and the redundant model eventually disappears after some iterations.

Chapter 5

Continuous hidden context

The multiple model paradigm is an option for dealing with nonstationary dynamics in motor control. Using a set of models and switching between them as the contexts change provides much faster adaptation to nonstationary dynamics and much higher control performance than using a single dynamics model and adapting it online. However, the multiple model paradigm has many limitations. In this chapter, some of these shortcomings are first discussed and then, a reformulation of the probabilistic multiple model approach that was presented in Chapter 4 is proposed. Scenarios when prior knowledge on the relationship between the context and modulation of dynamics is known are discussed as well as solutions to cases when this is not readily available.

5.1 Disadvantages of the multiple model paradigm

The multiple model paradigm has several limitations. First, the right number of models needs to be known or estimated. Estimating the number of contexts only from data (using some model selection procedure) may be tricky. Realistically, novel contexts may appear quite often and to cope with this, a novelty detection mechanism is needed. However, even with a very robust novelty detection mechanism, we may end up with a very large number of models, since there may be a continuum rather than a set of possible contexts. Moreover, we would like to generalize between contexts and most multiple model paradigms do not cope well with this. For example, Gomi's model (Gomi and Kawato, 1993), MPFIM (Wolpert and Kawato, 1998) and MOSAIC (Haruno et al., 2001) assume that the context responsibilities can be used for weighing the predictions of individual models. Apart from the fact that this is clearly not justified for systems with nonlinear dynamics, it also limits the generalization to the convex space between the set of models.

5.2 Generalizing between experienced contexts: continuous hidden context

In the multiple model paradigm, a discrete variable represents the context. If the context is represented by a set of continuous variables there is no need to consider how many models to use. Furthermore, assuming that there is a smooth relationship that relates the change of the dynamics to the change of such continuous contextual variables, we can hope that we will be able to learn this relationship and thus be able to achieve generalization from a set of contexts to novel contexts. Therefore, we can attempt to circumvent the issues with the multiple model paradigm by replacing the set of models with a single model that takes as additional input appropriate *continuous* hidden contextual variables, i.e., instead of a set of g_is corresponding to different contexts, a single inverse model G is used:

$$\tau_t = G(\theta_t, \theta_{t+1}, c_t) \; . \tag{5.1}$$

Here, c_t (c.f. Fig. 4.1) is not a discrete variable that indexes different models but a set of continuous variables that provides information about the context. The probabilistic model of the inverse dynamics would then be:

$$P(\tau \,|\, \theta_t, \theta_{t+1}, c_t) = \mathcal{N}\left(G(\theta_t, \theta_{t+1}, c_t),\; \sigma(\theta_t, \theta_{t+1}, c_t)\right) . \tag{5.2}$$

It needs to be clear that the use of a set of continuous hidden contextual variables allows us to deal with a continuum of contexts, regardless of how the context changes may occur: it could deal both with a continuous change of context (e.g. a leaking bottle) and an abrupt change of context (e.g. manipulating different tools).

The temporal version of the model (5.2) – where a dependency $p(c_{t+1}|c_t)$ is introduced – is equivalent to a state-space model. This is a class of models that is well studied in the graphical models and statistical learning literature. An issue with state-space models, but also with any latent variable model, is that the learned representation of the hidden variable is not unique. The same likelihood can be achieved for the same data $\theta_{1:T}$ and $\tau_{1:T}$ for different representations of the hidden variable $c^a_{1:T}$ and $c^b_{1:T}$, just by changing the observation model from $p^a(\tau \,|\, \theta_t, \theta_{t+1}, c_t)$ to $p^b(\tau \,|\, \theta_t, \theta_{t+1}, c_t)$. In fact, for models with continuous latent variables there are infinite different possible representations of the latent variable that could result in the same likelihood. Especially in the case of a nonlinear relationship of the hidden variable to the observed, the nature

of the representation could be completely arbitrary. Usually, some form of regularization is used to encourage finding representations of some particular form. In general though, learning in such models from a completely naive state is a very difficult problem. Discussion will be split in two parts. In the first (Section 5.3 and Section 5.4) it will be assumed that prior knowledge on the nature of the contextual variables is known and in the second (Section 5.5) that it is not.

5.3 Manipulation of objects: linearly modified dynamics

A possibility for learning the augmented model is to follow the same procedure as in the discrete case for learning the models, i.e., use EM. However, since learning in such a model is a very difficult task in general, it is useful to exploit any prior knowledge about the relationship of the inverse model to appropriate contextual variables. For the case of manipulation of objects with a robot arm, this is possible. In particular, we can take advantage of the fact that the dynamics of a robot arm has a linear relationship to the inertial properties of the links. In other words, the inverse dynamics can be written in the form:

$$\sum_{j=1}^{J} y_{ij}(q,\dot{q},\ddot{q})^T \pi_j = \tau_i \tag{5.3}$$

where τ_i is the torque applied at the i^{th} joint, $y_{ij}(q,\dot{q},\ddot{q})$ is a function $\mathbb{R}^{3J} \mapsto \mathbb{R}^{10}$ and $\pi_j \in \mathbb{R}^{10}$ includes the inertial parameters of the j^{th} link

$$\pi_j = [m_j, m_j l_{jx}, m_j l_{jy}, m_j l_{jz}, I_{jxx}, I_{jxy}, I_{jxz}, I_{jyy}, I_{jyz}, I_{jzz}] \tag{5.4}$$

m_j is the mass, l_j is the position of center of mass in the link's reference frame and I_j is the inertia tensor of the link. It is important to note that the term $y_{ij}(q,\dot{q},\ddot{q})$ depends only on kinematics quantities, i.e. joint angles, velocities, accelerations, link lengths, rotation axes etc. The above relationship can be derived based on fundamentals of robot dynamics (Sciavicco and Siciliano, 2000; Craig, 2005) as shown in Appendix C.

5.3.1 Learning the augmented model

Now, let's examine how this can be used to acquire the augmented model for the scenario of changing loads. Indexing the dynamics in (5.3) in terms of the context we

have for the m^{th} context:

$$\sum_{j=1}^{J} y_{ij}^{m}(q,\dot{q},\ddot{q})^{T}\pi_{j}^{m} = \tau_{i}^{m} \tag{5.5}$$

The first thing to note is that the kinematics dependent terms $y_{ij}^{m}(q,\dot{q},\ddot{q})$ do not change as different objects are manipulated. Only the inertial parameters of the last link of the arm change as different loads are manipulated, i.e. the vector π_{J}^{m} is different between contexts, whereas the vectors π_{j}^{m} are common between all contexts $1...M$ for all links $1...J-1$. Thus, we can define:

$$y_{ij}(q,\dot{q},\ddot{q}) = y_{ij}^{1}(q,\dot{q},\ddot{q}) = y_{ij}^{2}(q,\dot{q},\ddot{q}) = ... = y_{ij}^{M}(q,\dot{q},\ddot{q}), \forall i,j \tag{5.6}$$

and also

$$\pi_{j} = \pi_{j}^{1} = \pi_{j}^{2} = ... = \pi_{j}^{M}, \text{ for } j = 1,...,J-1 \tag{5.7}$$

Then, (5.5) becomes:

$$(\sum_{j=1}^{J-1} y_{ij}(q,\dot{q},\ddot{q})^{T}\pi_{j}) + y_{iJ}(q,\dot{q},\ddot{q})^{T}\pi_{J}^{m} = \tau_{i}^{m} \tag{5.8}$$

The sum $\sum_{j=1}^{J-1} y_{ij}(q,\dot{q},\ddot{q})^{T}\pi_{j}$ can be precomputed and replaced with a single function $h_{i}(q,\dot{q},\ddot{q})$ to obtain:

$$h_{i}(q,\dot{q},\ddot{q}) + y_{iJ}(q,\dot{q},\ddot{q})^{T}\pi_{J}^{m} = \tau_{i}^{m} \tag{5.9}$$

Now, compiling the $J-1$ equations for the different joints we have:

$$\begin{bmatrix} \tau_{1}^{m} \\ \tau_{2}^{m} \\ \vdots \\ \tau_{J}-1^{m} \end{bmatrix} = \begin{bmatrix} h_{1}(q,\dot{q},\ddot{q}) \\ h_{2}(q,\dot{q},\ddot{q}) \\ \vdots \\ h_{J}(q,\dot{q},\ddot{q}) \end{bmatrix} + \begin{bmatrix} y_{1J}(q,\dot{q},\ddot{q})^{T} \\ y_{2J}(q,\dot{q},\ddot{q})^{T} \\ \vdots \\ y_{JJ}(q,\dot{q},\ddot{q})^{T} \end{bmatrix} \pi_{J}^{m}, \tag{5.10}$$

which will be compactly expressed as:

$$\tau^{m} = A(q,\dot{q},\ddot{q}) + B(q,\dot{q},\ddot{q})\pi_{J}^{m} \tag{5.11}$$

where $\tau^{m} \in \mathbb{R}^{J}$, $A : \mathbb{R}^{3J} \mapsto \mathbb{R}^{J}$ and $B : \mathbb{R}^{3J} \mapsto \mathbb{R}^{J\times 10}$.

It is clear that the inertial parameters π^{m} can be used as the latent contextual variables in the augmented model $G(\theta_{t},\theta_{t+1},c_{t})$ which can then be written as:

$$\tau_{t} = G(\theta_{t},\theta_{t+1},c_{t}) = A(q,\dot{q},\ddot{q}) + B(q,\dot{q},\ddot{q})\pi_{J}^{m} \tag{5.12}$$

Note that state transitions have been appropriately replaced by joint angles, velocities and accelerations. This is more compactly written as:

$$G(\theta_t, \theta_{t+1}, c_t) = \tilde{Y}(q, \dot{q}, \ddot{q})\tilde{\pi}_J^m = \tau \tag{5.13}$$

where $\tilde{\pi}_J^m$ denotes the vector $[1\ \pi_J^m]$ and $\tilde{Y}(q, \dot{q}, \ddot{q})$ denotes the $J \times 11$ matrix $[A(q, \dot{q}, \ddot{q})\ B(q, \dot{q}, \ddot{q})]$.

To *acquire* the model, essentially means to estimate $\tilde{Y}(q, \dot{q}, \ddot{q})$. If we have learned appropriate number of models M (that is, at least as many as the cardinality of $\tilde{\pi}^m + 1 = 11$) and the corresponding π^m labels, we can estimate $\tilde{Y}(q, \dot{q}, \ddot{q})$ as described below. Say, we have learned a set of *learned reference* models $\tau^1(q, \dot{q}, \ddot{q}), \tau^2(q, \dot{q}, \ddot{q}) ... \tau^M(q, \dot{q}, \ddot{q})$ corresponding to manipulation of objects which result in the last link of the arm having known inertial parameters $\pi_J^1, \pi_J^2 ... \pi_J^M$, then since $\tilde{Y}(q, \dot{q}, \ddot{q})$ is shared between models we can write:

$$\left[\ \tau^1 \tau^2 ... \tau^M\ \right] = \tilde{Y} \left[\ \tilde{\pi}_J^1 \tilde{\pi}_J^2 ... \tilde{\pi}_J^M\ \right] \tag{5.14}$$

or more compactly:

$$R = \tilde{Y}\tilde{\Pi} \tag{5.15}$$

Where, $R : \mathbb{R}^{3J} \mapsto \mathbb{R}^{J \times M}$ is a matrix with the predictions of the reference models as its columns and $\tilde{\Pi} \in \mathbb{R}^{11 \times M}$ is a matrix with the inertial parameters of the reference models as its columns. Given that R and $\tilde{\Pi}$ are known, this is essentially a multiple linear regression problem and thus, $\tilde{Y}(q, \dot{q}, \ddot{q})$ can be estimated as:

$$\tilde{Y}(q, \dot{q}, \ddot{q}) = R\tilde{\Pi}^T(\tilde{\Pi}\tilde{\Pi}^T)^{-1} = R\tilde{\Pi}^+ \tag{5.16}$$

where $\tilde{\Pi}^+$ denotes the pseudoinverse of $\tilde{\Pi}$.

Acquisition of the augmented model is displayed in Fig. 5.1. The dots are data belonging to different contexts. A model is fit to the data belonging to each of the contexts (the solid lines) and then, we can use the predictions of the learned models together with the known corresponding inertial parameters to perform multiple linear regression and acquire the augmented inverse model for any point of the input space (the dotted lines). Computing the augmented model for any point of the input space gives the dynamics model of any other context (dashed lines).

It is important to note that to acquire the augmented inverse model, the regression coefficient matrix $\tilde{Y}(q, \dot{q}, \ddot{q})$ has to be evaluated (which implies the least squares problem has to be solved) at all points in the input space. However, this is not as computationally expensive as it might seem at first. All that is needed is to reevaluate the

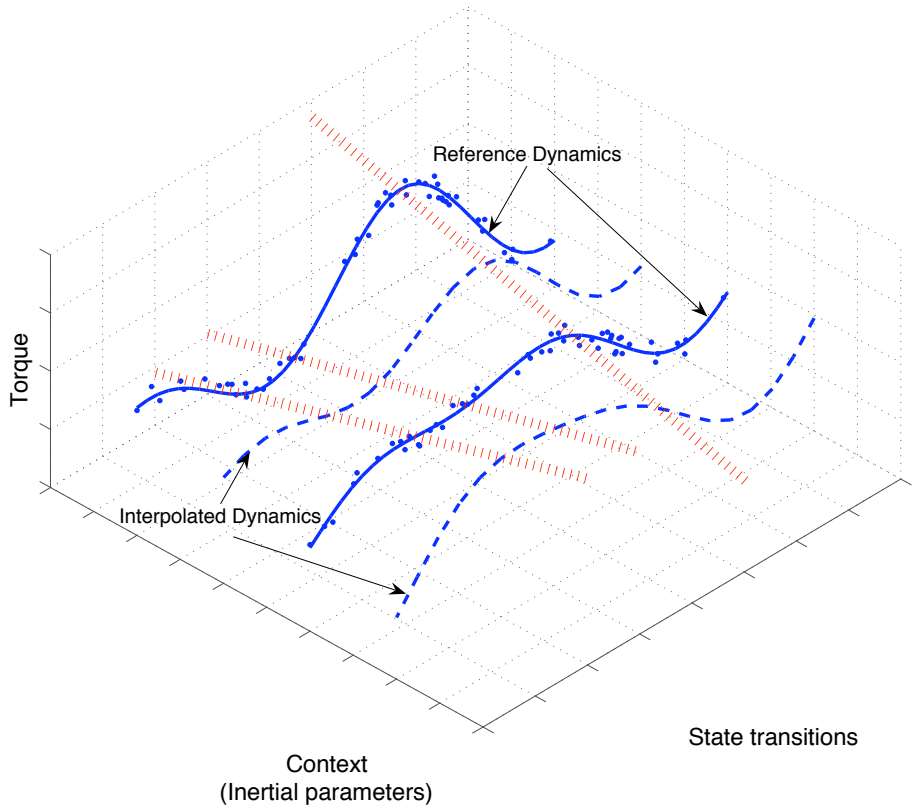


Figure 5.1: Learning the augmented inverse dynamics model using a set of learned reference models and their corresponding inertial parameters. The dots are training data for the different contexts. The solid lines are the learned models for each context, the red dotted lines show the augmented model derived by the set of learned models for some state transition and the dashed lines show the augmented model for some new context.

predictions of the reference inverse models at each point in input space and multiply by the pseudoinverse of the reference inertial parameters matrix $\tilde{\Pi}^{+}$. This pseudoinverse needs to be evaluated once and no further matrix inversion is needed to estimate the augmented model at any point in the input space.

The previous discussion implies that, ideally, if we have the prerequisite number of 'labeled' reference models, then, one can deal with manipulation of any object. In practice, however, since learned dynamic models will not be perfect and due to the presence of noise in the sensor measurements, a larger number of reference models may be necessary to give accurate estimates and achieve robust control performance.

Furthermore, it was assumed that the predictions of all models are taken equally into account for acquiring the augmented inverse model. Nevertheless, there may be higher confidence to the predictions of some of the reference models. In cases like that, it would be more appropriate to use weighted least squares instead of simple least squares as:

$$\tilde{Y}(q,\dot{q},\ddot{q}) = WR\tilde{\Pi}^{T}(\tilde{\Pi}W\tilde{\Pi}^{T})^{-1} \tag{5.17}$$

where W gives the weighting of individual models and could come from the LWPR confidence bounds for example. For the purposes of this study, it will be assumed that all learned models are learned well and no weighting of the individual models' predictions is required. Nevertheless, it will be useful to keep in mind that this may be required for other scenarios, where some reference models may not have been learned accurately.

5.3.2 Context estimation

The augmented model can be used both for control and context estimation purposes. For *control* purposes, say we have an estimate of π^{m} at time t, given the desired transition for the next time step, we can easily compute $\tilde{Y}(q^{d},\dot{q}^{d},\ddot{q}^{d})$ using (5.16) and hence, the feedforward command. For robust context estimation, we can use temporal dependencies, similar to the principles used in the multiple model scenario. However, since we now have a set of continuous hidden variables as opposed to a single discrete context variable, the inference is slightly more involved.

With reference to (5.12), the probabilistic probabilistic formulation of the augmented inverse model is

$$\tau_t = G(\theta_t,\theta_{t+1},c_t) = A(\theta_t,\theta_{t+1}) + B(\theta_t,\theta_{t+1})c_t + \eta \tag{5.18}$$

where $\eta = \mathcal{N}(0, \Sigma_{obs})$. Σ_{obs} is estimated from the confidence bounds of the inverse models that form the augmented model. Please note that for brevity we will use states θ_t rather than joint angles q_t and velocities $\dot{q}_t$.

Also, the transition model for the context needs to be defined. Since we believe that the context does not change too fast, this is set to:

$$c_{t+1} = c_t + \zeta \tag{5.19}$$

where $\zeta = \mathcal{N}(0, \Sigma_{tr})$ with Σ_{tr} set to a very small value.

Based on the defined model, we can write down the inference for the temporal Bayesian network using the augmented inverse model. For control, only filtered estimates (a la Kalman filtering) can be used.
We want to compute $p(c_t \,|\, \tau_{1:t+1}, \theta_{1:t+1})$ using the estimate at the previous time step $p(c_{t-1} \,|\, \tau_{1:t}, \theta_{1:t})$ and the new evidence τ_{t+1} and θ_{t+1}. The previous estimate $p(c_{t-1} \,|\, \tau_{1:t}, \theta_{1:t})$ is defined as:

$$p(c_{t-1} \,|\, \tau_{1:t}, \theta_{1:t}) = \mathcal{N}(\mu_{t-1|t}, \Sigma_{t-1|t}) \tag{5.20}$$

Estimates for the next time step $p(c_t \,|\, \tau_{1:t+1}, \theta_{1:t+1})$ are obtained in a recursive way in two steps. The first is the **prediction** step where, $p(c_t \,|\, \tau_{1:t}, \theta_{1:t})$ is computed using the filtered estimate on the previous time step and the transition model $p(c_{t+1} \,|\, c_t)$, without taking into account evidence at time $t+1$:

$$p(c_t \,|\, \tau_{1:t}, \theta_{1:t}) = \mathcal{N}(\mu_{t|t}, \Sigma_{t|t}) \tag{5.21}$$

where $\mu_{t|t} = \mu_{t-1|t}$ and $\Sigma t \,|\, t = \Sigma_{t|t} + \Sigma_{tr}$. Then, the **filtered estimate** modifies the predicted estimates using the observation at the time $t+1$ as (dependency of A and B on the state transition is omitted for compactness):

$$p(c_t \,|\, \tau_{1:t+1}, \theta_{1:t+1}) = \mathcal{N}(\mu_{t|t+1}, \Sigma_{t|t+1}) \tag{5.22}$$

where,

$$\mu_{t|t+1} = \mu_{t|t} + \Sigma_{t|t} B^T (B \Sigma_{t|t} B^T + \Sigma_{obs})^{-1} (\tau_{t+1} - A - B\mu_{t|t}) \tag{5.23}$$

$$\Sigma_{t|t+1} = \Sigma_{t|t} - \Sigma_{t|t} B^T (B \Sigma_{t|t} B^T + \Sigma_{obs})^{-1} B \Sigma_{t|t} \tag{5.24}$$

5.3.3 Relationship to load identification

The approach to contextual parameter estimation described is closely related to classical load estimation work in robotics, e.g. (Swevers et al., 2002, 2000; Dutkiewicz

et al., 1993a). There are two main classes of load estimation methods. One requires torque sensing at all joints of the robot (Olsen and Bekey, 1986; Swevers et al., 1997), while the other requires force and torque sensing only at the wrist of the manipulator (Atkeson et al., 1986; Dutkiewicz et al., 1993b). Both classes of methods, are based on a linear relationship of the inertial parameters of the load and links to the dynamics of the robot, however existing methods rely on analytically derived dynamics of the arm in a form similar to (5.3). The method that requires torque sensing at all joints, is essentially the approach presented previously. The other approach to load estimation uses a similar relationship which relates linearly the torque and force applied by the load to the last link of the robot to the inertial properties of the load π_{n+1} only (not the union of the last link and load). A force and torque sensor at the wrist of the robot (between the last link and load) provides a set of measurements and a set of new "regressor matrices" is computed and a least squares problem is solved to estimate π_{n+1}. It is also worth noting that, estimation methods other than Kalman filtering can be used for context estimation, e.g. recursive least squares.

There is an important issue in the described procedure that was not discussed. In general not all inertial parameters can be identified. There are two reasons for that. The first is that some parameters do not contribute at all in the dynamics. For example, consider a single link robot that revolves only around the y axis, then the moments of inertia around the x and z axes do not contribute at all in the dynamics and the corresponding columns of $\tilde{Y}$ are zero. The second reason is that due to the structure of the manipulator, some columns of $\tilde{Y}$ are linearly dependent. In general, the inertial parameters can be grouped in three categories:

- Identifiable parameters, that correspond to linearly independent columns of $\tilde{Y}$.
- Partially identifiable parameters, that correspond to linearly dependent columns of $\tilde{Y}$.
- Unidentifiable parameters, that do not contribute at all in the dynamics and the corresponding columns of $\tilde{Y}$ are zero.

Existence of partially identifiable and unidentifiable parameters means that $\tilde{Y}$ is not full rank and thus the least squares problem cannot be solved without numerical problems. A solution is to do ridge regression instead of least squares estimation. The accuracy of the estimates of identifiable parameters will depend on the selection of the penalty term λ, whereas non-identifiable parameters' estimates will in general be

significantly erroneous. A better solution is to remove the columns of $\tilde{Y}$ corresponding to unidentifiable parameters and replace the columns corresponding to partially identifiable parameters by a proper linear combination. There has been some work on either symbolically (Khalil and Bennis, 1995; Huo, 1995) or numerically (Antonelli et al., 1999; Gautier, 1990) characterizing the identifiability of inertial parameters. The most popular numerical approach involves doing Singular Value Decomposition on the $\tilde{Y}$ matrix, for more details see (Gautier, 1990).

Classification of the inertial parameters of each link depends on the structure of the manipulator. However, some parameters may appear to be unidentifiable for specific movements of the manipulator although they are not for others. For this purpose, a sufficiently rich movement has to be generated and there has been some work on finding rich movements (Swevers et al., 1997; Presse and Gautier, 1993). In general, polynomial or sinusoidal trajectories are sufficiently rich. In the context of this study, the richness of the movement should not be an issue, as a sinusoidal trajectory is used. In addition, since the focus is on using local methods and particular trajectories, if identification of only the relevant variables for these trajectories is possible, it is sufficient for our purposes. Thus, we will not focus on identifiability issues and generation of rich movements. We will assume that we know to which class each of the inertial parameters of the compound body that comprises of the last link and load belongs.

It should be possible though to do automatic classification of the identifiability of the inertial parameters. Since a learned set of models with their respective inertial parameters can give an estimate of the actual regressor matrix, a sequence of estimates can be used with a numerical algorithm like SVD to achieve this.

5.3.4 Unknown inertial parameters of the reference loads

Previous analysis relied on the fact that adequate number of reference learned dynamic models with *known* inertial parameters exists. In many cases, an accurate estimate of the parameters, even for reference loads may not be forthcoming. Can we avoid using the real inertial parameters altogether? Consider introducing a linear $s \times s$ (where $s-1$ is the number of inertial parameters to be identified) transformation A and it's inverse between the regressor matrix and the set of inertial parameters in (5.13) like:

$$\tau = \tilde{Y}(q, \dot{q}, \ddot{q}) A^{-1} A \tilde{\pi}_n \tag{5.25}$$

Then (5.15) becomes:

$$R = \tilde{Y} A^{-1} A \tilde{\Pi} \tag{5.26}$$

and we can group the terms as:

$$R = (\tilde{Y}A^{-1})(A\tilde{\Pi}) \tag{5.27}$$

Hence, instead of estimating $\tilde{Y}$, one can estimate the transformed regressor matrix $\tilde{Y}A^{-1}$ as:

$$\tilde{Y}A^{-1} = R\tilde{\Pi}^{+}A^{-1} = RB \tag{5.28}$$

Where, $B = \tilde{\Pi}^{+}A^{-1} \in \mathbb{R}^{M\times s}$. If we assume that we have $M = s$ learned dynamic models and the vectors of the inertial parameters for the different loads of the learned models are linearly independent, then one can set B arbitrarily to any square full rank matrix. In the case that there are more than M models, we cannot initialize the B matrix arbitrarily since there may be no exact mapping A^{-1} from the pseudoinverse of the actual parameters matrix $\tilde{\Pi}$ to the arbitrarily set B matrix. The obvious choice for B is the identity matrix. Then, (5.28) simply becomes:

$$\tilde{Y}A^{-1} = R \tag{5.29}$$

In other words, the predictions of the reference models can be used for estimating the transformed regressor matrix directly and no further computation is required. In what follows, it will be assumed that B has been set to the identity matrix. The estimate of the transformed regressor matrix can be used for estimating the transformed context which can then be used for control.

In this setup, the estimated load is not an estimate of the actual load but an estimate of a linear transformation of the load. However, when multiplied by the (automatically) appropriately transformed regressor matrix, it gives the correct model of dynamics for the current load.

If we have less learned models than are needed, i.e. if $M < s$, it is not possible to obtain an augmented model with $s-1$ hidden variables. If more learned models are available, i.e. if $M > s$ using only s out of the M reference learned models is very limiting. As suggested in (Chai, 2008), a solution could be to perform Principal Component Analysis (PCA) so that $\tilde{Y}A^{-1}$ is an approximation to the principal modes of variation in R.

5.3.5 Experiments

The augmented model proposed for extracting the continuous context (latent variable) was empirically evaluated.

The problem was constrained in a way such that, for the three DOF robot arm used in our simulations, only three out of the ten inertial parameters of the last link could be estimated and thus, three contextual variables were needed to describe the augmented model. These inertial parameters are the mass, the mass $\times$ the $y-position$ of the center of mass and the moment of inertia around the z axis . This was achieved by constraining the center of mass both of the link and the object to lie on the y axis (see Fig. 5.2). Thus, mass $\times$ the $x-position$ and mass $\times$ the $z-position$ are zero. Furthermore, the off-diagonal elements of the inertia tensor are zero and only the moment of inertia around the z axis has significant contribution to the dynamics.

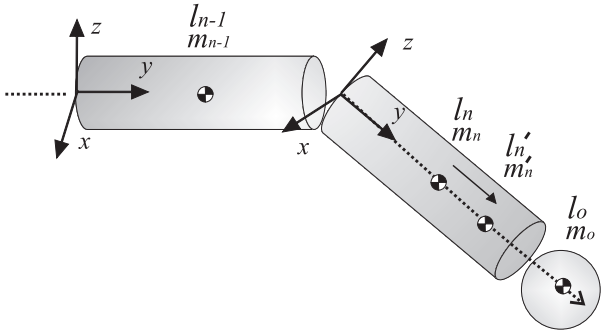


Figure 5.2: In our experiments, both the center of mass of the last link l_n and the load l_o are constrained to lie on the y axis of the last link's reference frame, so that the center of mass of their union $\acute{l}_n$ also lies on the y axis.

In contrast to previous experiments, now both the mass and shape of the manipulated object change randomly and can take any value in a specific range. We start by not using the context estimates for control, i.e. we apply PD control and we observe the accuracy of the context estimates. We then repeated the same experiments but using the context estimates for control to see if the resultant accuracy is sufficient for motor control. Experiments were executed for both cases five times, using different reference models for each of the runs. Fig. 5.3 (a) shows the estimation accuracy of the three context variables for the no control and control cases. The error measure used is the nMSE of the target variable. The mass and the product of y-position of the center mass with the mass were more accurately estimated than the moment of inertia around the y axis. Furthermore, we can see that the performance is better when estimates are actually used for control, showing that the augmented model actually captures the modulation of the dynamics due to the nonstationary context. Figures 5.3 (b-d) show a

snapshot of the actual and estimated contextual variables. For the case that the context estimates were used for control, the average ratio of feedback to composite command was 0.1101 with standard deviation between the runs of 0.0176.

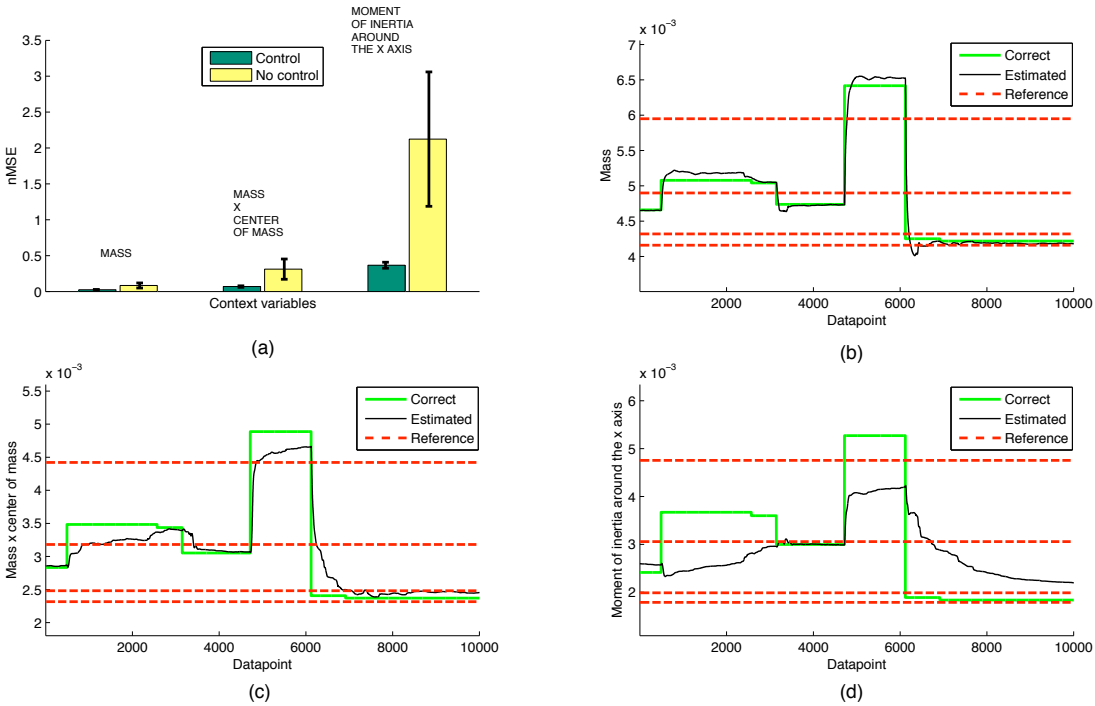


Figure 5.3: Context estimation using the augmented inverse dynamics model. (a) nMSE of the three contextual variables while using and not using the context estimates for control (b-d) Actual and estimated context variables, along with the values of the context variables of the reference models that were used for deriving the augmented model.

Finally, the same experiments were repeated without using the inertial parameters of the reference models. However, since the estimated context is now an estimate of an unknown linear transformation of the actual context, we cannot evaluate the performance by comparing the estimated with the actual context. Instead we judge the performance from the ratio of feedback to composite command. Fig. 5.4 (a) shows the ratio of feedback to composite command for the three joints. The average ratio of the three joints is just 0.0893 with standard deviation 0.0239 between runs. Fig. 5.4 (b) shows the actual and reference masses (left axis) and the estimated mass (right axis). There cannot be a direct comparison, however we observe that the evolution of the estimate is strongly correlated to the evolution of the actual mass.

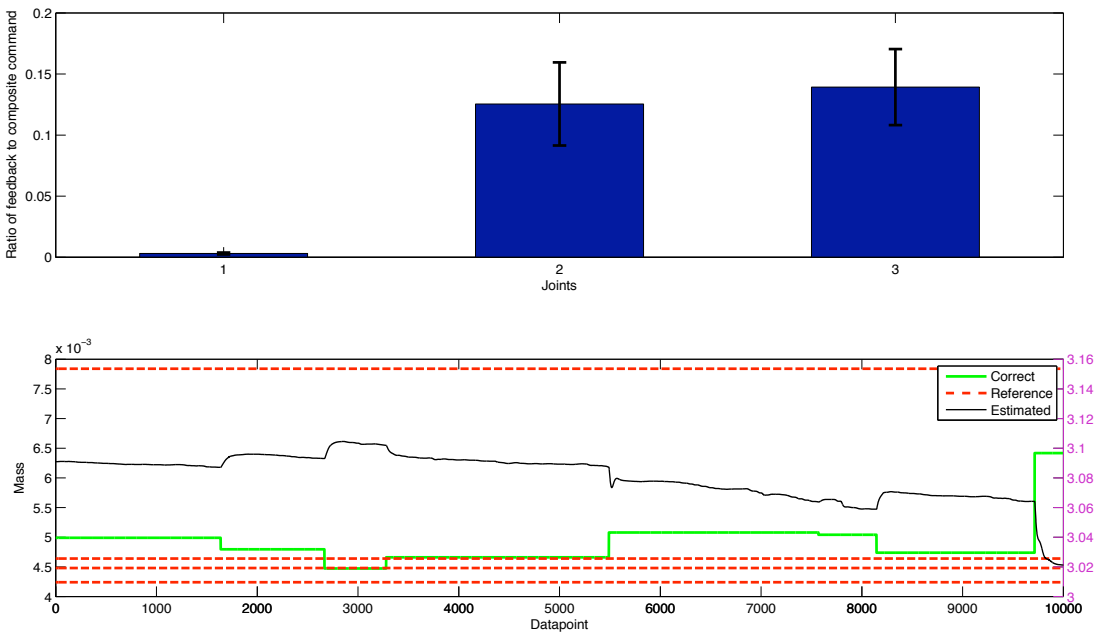


Figure 5.4: Context estimation using the augmented inverse dynamics model (derived without the reference models' inertial parameters). (a) Ratio of feedback to composite command for the three joints (b) Estimated (right axis), actual and reference models' mass (left axis).

5.4 Inferring context from tactile sensors [1]

In this section, another example of using prior knowledge on the relationship of some hidden context to the modulation of dynamics is presented. It is demonstrated that tactile sensing can be used for inferring the unknown mass of a manipulated object and that this estimate can be used for control purposes in a setting similar to the one in the previous section.

5.4.1 A simplistic model of tactile sensors

Tactile sensing is modeled as force sensing at the interface between hand and object assuming that the hand grasps tightly the manipulated object. We first show that the sensory values s_i, the 'tactile' forces applied by the load to the hand are linear in the mass m held in the robot's hand. In the reference frame of the hand, the acceleration a of an object leads to a small displacement dx (Fig. 5.5).

[1] A major part of this Section, in particular, the formulation of the tactile sensors as force sensors and the ODE implementation is work done by Heiko Hoffmann. In addition, the experiments were done in collaboration with Heiko Hoffmann and Sebastian Bitzer. The author's contribution was the formulation

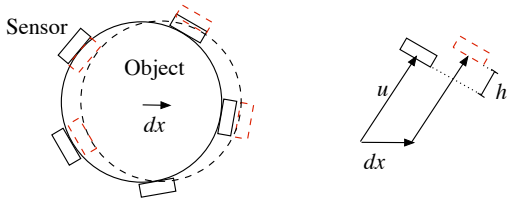


Figure 5.5: A force on the object held in the robot's hand leads to a displacement dx. This displacement shifts each sensor at position u (relative to the object's center) by h.

This displacement pushes each sensor by the amount h_i depending on the sensor's position u_i. Let e_i be a vector of unit length pointing in the direction of u_i, then $h_i = e_i^T dx$. Our sensors act like Hookean springs; thus, the resulting force equals $f_i = \kappa h_i e_i$, with κ being the spring constant. Since the object is held such that it cannot escape the grip, the sum of sensory forces f_i must equal the inertial force ma,

$$ma = \sum_{i=1}^{n} f_i = \kappa \sum_i (e_i^T dx)\, e_i. \tag{5.30}$$

This linear equation allows the computation of dx,

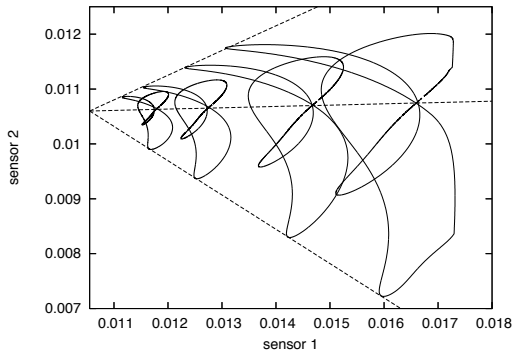


Figure 5.6: Two-dimensional projection of sensor values during figure of 8 movements with four different masses, indicating that the sensor values have a linear relationship to the mass of the manipulated object. From left to right, the mass increases as 0.005, 0.01, 0.02, and 0.03.

$$dx = \frac{m}{\kappa}(E^T E)^{-1} a, \tag{5.31}$$

where E is a matrix with row vectors e_i. Thus, each f_i is proportional to m. The total force measured at a sensor equals f_i plus a constant grip force (whose sum over all sensors equals zero). Therefore, the sensory values s can be written as

$$s = s_0 + m\varphi(\theta, \dot{\theta}, \ddot{\theta}), \tag{5.32}$$

of the augmented tactile model (learning and use for context inference).

where φ is a function depending on the state and acceleration of the robot arm. This linearity is illustrated in Fig. 5.6 using data from our simulated sensors. Based on (5.32) and using a set of learned sensory models, mapping state transitions to sensory values, an augmented sensory model that depends on the mass can be created. The same inference as in Section 5.3 is possible using the augmented sensory model and the estimated m can subsequently be used for control with the augmented inverse dynamics model to provide control commands under varying contexts.

For accurate inference of mass of an object, tactile sensors have two main advantages over the method using the augmented inverse dynamics model. First, tactile forces relate more directly to the mass m of an object: $s - s0$ is directly proportional to m (s_0 is constant and can be therefore accurately determined); the control torques are proportional to m plus the mass of the end-effector link. For example, estimating the mass of an egg using the augmented inverse dynamics model of a heavy robot arm may be difficult. However, tactile forces may reveal this mass.

The second advantage is that the error of the mass estimate can be reduced by increasing the number of sensors. The expected squared error of m is inversely proportional to the number n of sensors. The mass m can be inferred based on the probability density of $p(s|m)$,

$$p(s|m) = \eta_s \exp\left(-\frac{1}{2}(s - s_0 - m\varphi)^T R^{-1}(s - s_0 - m\varphi)\right), \tag{5.33}$$

where η_s is a normalization constant and R is the covariance matrix of the sensor noise. Using Bayes' rule and without prior knowledge of m, $p(m|s)$ is proportional to $p(s|m)$. Thus, the variance σ^2 of m is

$$\sigma^2 = \left(\varphi^T R^{-1}\varphi\right)^{-1} = \left(\sum_{i=1}^{n} \varphi_i^2 / R_{ii}\right)^{-1}, \tag{5.34}$$

where the last equality assumes that the noise from different sensors is uncorrelated (R is diagonal). Since the variance of φ_i is proportional to R_{ii} (because $\mathrm{var}(s) = m^2\mathrm{var}(\varphi)$ if we ignore the variance of s_0, which can be accurately estimated), the average of φ_i^2/R_{ii} is independent of i and n. Thus, the expectation value of σ^2 is inversely proportional to n. Basically, with more sensors, the noise cancels out. Therefore, in the inference from sensors case, we can use arbitrarily many sensors to reduce the noise of m. Increasing the number of sensors seems more feasible for most cases than increasing the given number of joint torques. Given these advantages and assuming that $\tau(\theta, \dot{\theta}, \ddot{\theta})$ and $s(\theta, \dot{\theta}, \ddot{\theta})$ can be learned with similar confidence boundaries, we expect a lower variance on the m estimate from tactile sensors.

5.4.2 Experiments

The DLR arm is again simulated using ODE. As end-effector, we attached a simple hand with four stiff fingers; its purpose was to hold a spherical object tightly with the help of five simulated force sensors. Furthermore, three of the joints were controlled while the remaining joints were kept fixed. The simulated DLR arm, with the active joints marked and the hand is displayed in Fig. 5.7

Our force sensors are small boxes attached to damped springs. In the simulation, damped springs were realized using slider joints, whose positions were adjusted by a PD controller. The resting position of each spring was set such that it was always under pressure. As sensor reading s, we used the current position of a box (relative to the resting position).

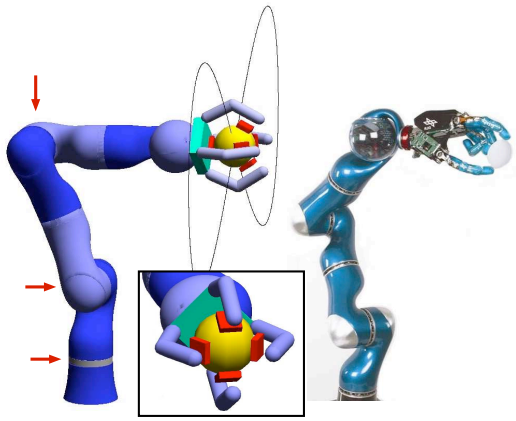


Figure 5.7: Simulated robot arm with gripper and force sensors and its real counterpart, the DLR light-weight arm III. Arrows indicate the three active joints used for the experiments. The curve illustrates the desired trajectory of the ball.

The task was to move a ball in a figure of eight trajectory. Three trajectories of different size were used: Fig. 5.7 shows the big eight; small and medium eight are 0.9 and 0.95 of the big eight's size respectively (see Fig. 5.12). The inverse kinematics were computed to obtain the trajectory in terms of joint angles. The length of the trajectory was 5000 time steps (i.e. 50 seconds since simulation and control are performed at 100 Hz). For training, data points were used from the two extremal trajectories, excluding the middle trajectory. For testing, all three trajectories were used. The maximum mass ($m = 0.03$) of the ball was about one seventh of the total mass of the robot arm.

For each mass context, 10000 data points were collected. Half of these points (every second) were used for training and the other half for testing the regression performance. Two types of mappings were learned. The first maps the state and acceleration

values $(\theta, \dot{\theta}, \ddot{\theta})$ onto joint torques (to obtain the augmented inverse model). The second maps the same input onto the five sensory values (to obtain the augmented sensory model). These mappings were trained on two different labeled masses ($m_1 = 0.005$ and $m_2 = 0.03$) and were subsequently used to obtain the augmented sensory and inverse dynamics models. Again LWPR was used to learn the individual models.

To generate training data, a PID controller was used. For testing the control performance, a composite controller provided the joint torques. We have two classes of experiments. In the first, the object's mass was estimated using the augmented inverse dynamics model and in the second using the augmented sensory model (similarly to the inference in Section 5.3.2). This estimate was then used in both cases with the augmented inverse dynamics model to provide the feedforward command.

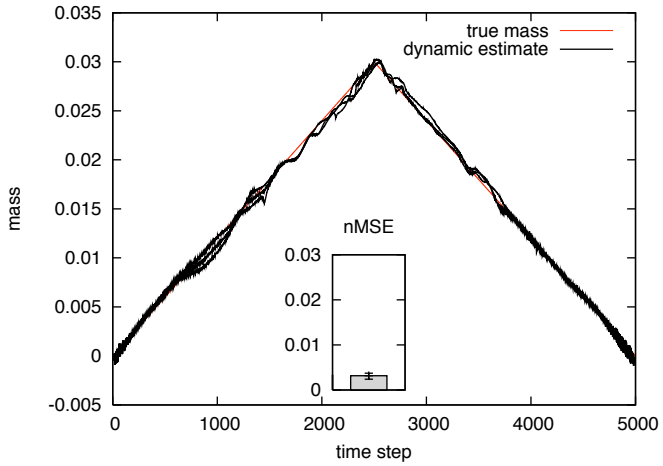


Figure 5.8: Inferring mass purely from dynamics. The inference results are shown for all three trajectories. The inset shows the normalized mean square error (nMSE) of the mass estimate. The error bars on the nMSE are min and max values averaged over an entire trajectory.

Both inference strategies (torque based and tactile based) allowed to infer the unknown mass accurately (Figs. 5.8 to 5.11).

Both types of mappings from state and acceleration either onto torques or onto sensors could be learned with low regression errors, which were of the same order (torques: for m = 0.005 the nMSE was $2.9 * 10^{-4}$ and for m = 0.03 the nMSE was $2.7 * 10^{-4}$; sensors: for m = 0.005 the nMSE was $1.3 * 10^{-4}$ and for m = 0.03 the nMSE was $2.2 * 10^{-4}$). The error of the inferred mass was about the same for dynamics and sensor pathway. However, the variation between trials was lower in the sensor case.

Despite the similar error, however, the sensor based mass estimate had a lower

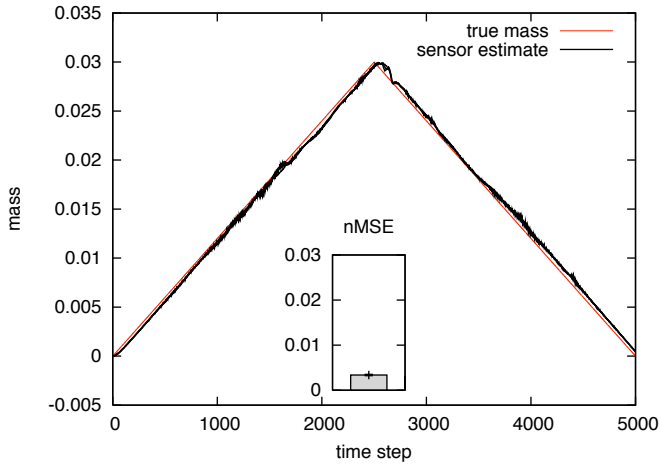


Figure 5.9: Inferring mass using tactile sensors. For details see Fig. 5.8.

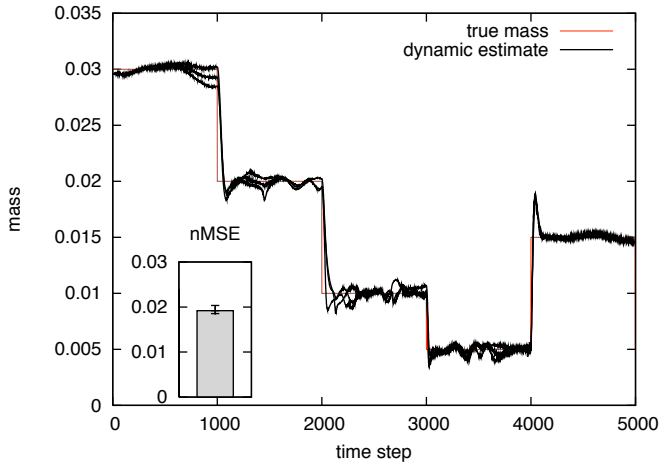


Figure 5.10: Inferring mass purely from dynamics. For details see Fig. 5.8.

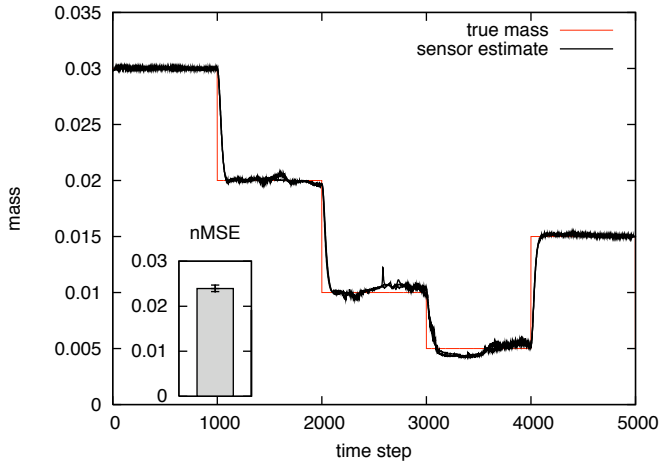


Figure 5.11: Inferring mass using tactile sensors. For details see Fig. 5.8.

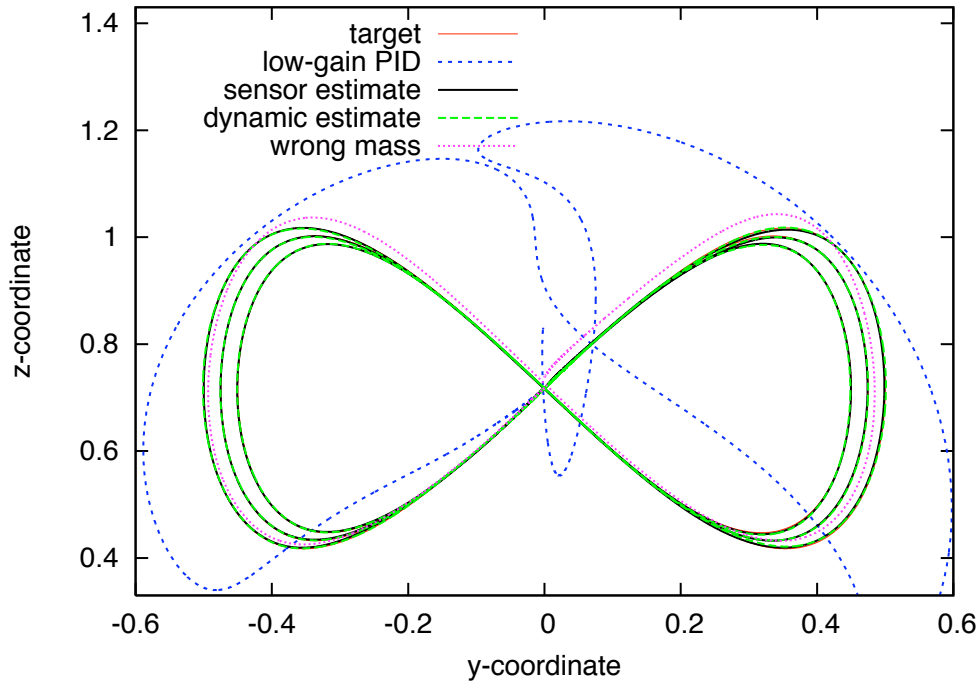


Figure 5.12: Tracking of three figure of eight trajectories. The true mass decreased continuously from 0.03 to 0. The three solid red lines show the target trajectories. The three solid black lines show tracking of the three trajectories when the tactile based estimate is used. The three dashed green lines show tracking of the three trajectories when the dynamics based estimate is used. The dashed purple line, shows tracking of the large eight using low-gain PID control. Contrary to composite control with an inverse dynamics model using the tactile or dynamics based context estimates (solid black and dashed green lines respectively), tracking is very poor. Tracking is also not perfect when a wrong mass estimate is used ($m = 0.03$, on large eight only), shown with the dotted purple line.

variance between trials than the dynamics based estimate (Figs. 5.8 to 5.11). This is in agreement to the previous discussion, which argues that the prediction error can be reduced with tactile sensors.

Nevertheless, here, in the experiments, the nMSE of the mass estimate from sensors was about the same as the estimate from dynamics. The estimate from the sensors lagged behind (see Figs. 5.9 and 5.11) resulting in a systematic error. This error probably results from a characteristic of our simulated sensors, which are modeled as damped springs and thus require a finite time to relax to their equilibrium.

Furthermore, as shown in Fig. 5.12 tracking was accurate using the context estimates. The three solid red lines are the three target figure of eight trajectories. The true mass decreased continuously from 0.03 to 0 for each trajectory. The realized trajectories using both estimation methods, tactile and dynamics based, shown with the three solid black and the three dashed green lines, overlap with the desired trajectory. This indicates that the context estimates were accurate and useful for control. The dotted purple line shows tracking using a wrong context estimate ($m = 0.03$, on large eight only) and illustrates that without a correct mass estimate, tracking is impaired.

5.5 Continuous hidden context: nature of context not known

Previously, prior knowledge on the nature of the hidden contextual variables and their relationship to the modulation of the dynamics was used to formulate an augmented inverse dynamics model. However, such knowledge may not always be available. A different methodology needs to be formulated to treat the problem in such cases. If we follow the same probabilistic approach and considering that the relationship of the contextual variables to the dynamics may be arbitrary and that the way it affects the dynamics may be nonlinear, the linear state-space model that was formulated in the previous sections becomes a nonlinear state-space model. As it will be discussed, inference and learning are considerably more difficult in nonlinear state-space models than in linear state-space models.

The problem of learning and using an augmented model without knowledge of the nature of the context differs considerably from the same problem when there is such knowledge. First, the augmented model does not factorize as the one presented in the previous chapter and thus we will not be able to derive it from a set of other models: it

will be necessary to learn it directly as a single model. Secondly, we will not be able to use the straightforward and principled linear estimation techniques that were used before.

We first discuss some of the existing approaches for *inference* and *learning* in nonlinear state-space models. We will then attempt to apply some techniques for nonlinear state-space models to the problem of learning the dynamics of a robot manipulator and controlling it under nonstationary loads.

5.5.1 Inference in nonlinear state-space models

From a graphical model point of view, a state-space model has the same structure as an HMM. The difference is that the hidden variable is continuous rather than discrete. Since the graphical model structure of a state-space model is the same as the HMM's, Bayesian inference has the same structure, but with integration, instead of summation, over the hidden variable. Given a sequence of T observations $z_{1:T}$ and denoting the continuous hidden variable at time t as x_t, the main inference problem is computing the smoothed estimate $p(x_t|z_{1:t})$. The smoothed posterior can be computed as:

$$p(x_t|z_{1:T}) = \frac{p(z_1,...,z_t,x_t)p(z_{t+1},...,z_T|x_t)}{\int p(z_1,...,z_t,x_t)p(z_{t+1},...,z_T|x_t)dx_t} \tag{5.35}$$

The term $p(z_1,...,z_t,x_t)$ is analogous to α and the term $p(z_{t+1},...,z_T|x_t)$ is analogous to β in HMM inference. Again, $p(z_1,...,z_t,x_t)$ can be computed using a recursive formulation from $p(z_1,...,z_{t-1},x_{t-1})$ as follows:

$$p(z_1,...,z_t,x_t) = p(z_t|x_t)\int p(z_1,...,z_{t-1},x_{t-1})p(x_t|x_{t-1})dx_{t-1} \tag{5.36}$$

The quantity $p(z_{t+1},...,z_T|x_t)$ can also be computed from $p(z_{t+2},...,z_T|x_{t+1})$ using a similar backward pass as:

$$p(z_{t+1},...,z_T|x_t) = \int p(z_{t+2},...,z_T|x_{t+1})p(x_{t+1}|z_{t+1})p(z_{t+1}|z_t) \tag{5.37}$$

When there is a linear relationship of the hidden variables to the observed variables and a Gaussian noise model is assumed – just like in the previous formulation of the augmented model – computing the relevant quantities exactly is possible and is performed using the Kalman filter (or smoother) equations. We have already seen an instance of Kalman filtering in Section 5.3.2. Nevertheless, in the case of nonlinear observation models or non-Gaussian noise, the relevant quantities cannot be computed

analytically. For the forward recursion for example, it is usually not possible to compute the integral exactly. Thus, various approximation techniques are commonly used. Please refer to (Arulampalam et al., 2002) for a review.

An option for approximate inference in nonlinear state-space models is extended Kalman filtering and smoothing. This uses a linearization of the system around the current state estimates and the resulting filtering and smoothing equations are almost the same as the Kalman filtering and smoothing equations. Extended Kalman filtering and smoothing can only be used for mildly nonlinear systems and will usually fail when the linearization is not a good model of the system in the current operating state.

Another approach for inference in nonlinear state-space models is particle filtering (and smoothing) . In particle filtering (and smoothing) the posterior $p(x_t|z_{1:t})$ (or $p(x_t|z_{1:T})$ respectively) is approximated using a set of samples x_t^i with corresponding weights w_t^i, i.e. they have the form:

$$\sum_{i=1}^{P} w_t^i \delta(x_t - x_t^i) \tag{5.38}$$

Particle filtering (Doucet et al., 2001) will be presented below. For more details on particle smoothing methods please see (Kitagawa, 1996), (Isard and Blake, 1998).

There are different variants of the particle filter. A common variant that will be used later is presented below.

1. Sample P particles x_0^i from the prior distribution $p(x_0)$. Set $t = 0$.

2. Evaluate the weights of the particles x_t^i as: $w_t^i = \frac{p(x_t^i|z_t)}{\sum_{j=1}^{N} p(x_t^j|z_t)}$

3. Set $t = t + 1$ and sample P particles from $p(x_t^i|z_{1:t})$

4. Go to step 2

Another possibility for the E-step is to use variational methods (Jordan et al., 1999). In variational methods, a simpler than the actual parametric form of the posterior is assumed, e.g. a factorized distribution and the parameters of the simpler distribution are adjusted so that they match as well as possible the actual distribution. The form of the simpler distribution is chosen so that this match can be performed efficiently and the relevant characteristics of the distribution are maintained.

5.5.2 Learning nonlinear state-space models

Learning of a nonlinear state-space model clearly depends on the form of the observation and transition model. It also depends on what method is used for the inference step (E-step).

One approach was presented in (Ghahramani and Roweis, 1999; Roweis and Ghahramani, 2001). There, the observation model is a radial basis function network and extended Kalman smoothing is used for inference. It turns out that the EM update equations for this system are very similar to the EM equations for a linear state-space model (Ghahramani and Hinton, 1996). The use of Gaussian radial basis functions makes learning of the model computationally efficient. The centers and widths of the kernels of the radial basis function network are fixed, however it it possible to extend the algorithm in order to adapt these parameters as well. This approach has the disadvantage that a very large number of basis functions may be required for a latent space with even a moderate number of dimensions.

Another approach is presented in (Briegel and Tresp, 1999). In the E-step, an approximation to the posterior distribution of the latent variables is computed using either extended Kalman smoothing or the Fisher scoring algorithm (can be used to approximate either a single Gaussian or a mixture of Gaussians as the posterior). In the M-step, a set of samples are generated from the approximated posterior distributions and these are used as training data for neural networks that represent the transition and observation models.

Another approach comes from (Valpola and Karhunen, 2002). In this work, a fully Bayesian nonlinear state-space model is formulated and ensemble learning is used. The model is computationally very expensive like most Bayesian techniques.

Something that needs to be noted is that in none of the mentioned approaches is there any discussion about finding appropriate representations of the hidden variable. In practice, one may be interested in finding representations that have some particular characteristic, e.g. being smooth or have some particular range. The approach of (Valpola and Karhunen, 2002) seems more appropriate for achieving this as it is straightforward to define appropriate priors on the hidden variables. The other way to achieve the same effect is to regularize the hidden variables by setting its dynamics appropriately, i.e. define the transition model in such a way that specific representations are encouraged.

5.5.3 Learning a nonlinear state-space model for control using particle filtering

An EM algorithm for learning a nonlinear state-space model has been formulated. Pseudocode for the algorithm is given in Table 5.1. For details and the derivation of the algorithm please see Appendix D.

Table 5.1: Pseudocode for learning a nonlinear state-space model for control using particle filtering

```
set up simulation environment;
for t=1:T
        control arm using feedback control;
        collect t^th movement datum;
        switch context with some low probability;
end

for t=1:T
        fill in the missing value c_t for the t^th datapoint
        with a random value and train the model;
end
set the maximum likelihood noise estimate
of the transition and observation models to some values;

//EM
for iteration=1:em_iterations
        estimate the filtered posterior p(c_t|θ_{1:t+1},τ_{1:t})
        of each datapoint t using particle filtering;
        train the model with each datapoint t
        filling in the hidden variable with the particle with the maximum weight;
        compute the maximum likelihood estimate of the
        transition and observation variances using (D.19) and (D.14);
end


//Testing with composite control
for t=1:T
        perform particle filtering to estimate the hidden context;
        control arm using composite control, filling in the hidden variable in
        the augmented inverse model with the particle with the maximum weight;
        switch context with some low probability;
end
```

In the E-step, particle filtering is used to compute the posteriors of the hidden contextual variables. In principle, particle smoothing would be more appropriate, in practice though it turned out to be computationally prohibitively expensive. In the

M-step, the particles are used in combination with their weights to learn the model. Using all the samples for training again proved computationally too expensive and in practice, only the sample with the highest weight will be used for training. An EM algorithm like this where samples of the posterior of the hidden variables are used for training in the M-step is called Monte Carlo EM.

Once the nonlinear state-space model has been learned, particle filtering can be performed for estimating the context in a nonstationary context control scenario. The most likely sample can then be used with the augmented model to provide the feedforward command.

5.5.4 Experiments

The Monte Carlo EM algorithm in Table 5.1 has been tested on a simulated one degree of freedom arm. The single joint of the arm allows up and down movements of the single link. The task of the arm was to execute a sinusoidal trajectory. 400000 datapoints were first collected with the arm being controlled by PD control under randomly changing unknown loads. In the E-step 100 particles were used (as detailed in only the particle with the maximum weight was used in the M-step though) and 50 iterations of the EM were executed.

As mentioned earlier, the representation of the hidden context may be arbitrary and we may want to encourage particular solutions. In our case, after some trials it was found that regularization could be achieved by keeping the the transition model fixed. We used a mixture of a uniform distribution and a Gaussian with low noise (so that the estimates do not have very large variation) as the transition model. The uniform component of the Gaussian will be useful for handling abrupt context transitions whereas the Gaussian will be useful for slow transitions and refining estimates when the context remains stationary. The prior of the uniform component was set to 0.2, reflecting the belief that most of the time the context remains the same. The variance of the Gaussian was set to 0.001.

The context estimates along with the actual mass of the manipulated load are shown in Fig. 5.13. The plot has two vertical axes, in the left (blue line) the actual mass of the load is plotted, whereas in the right (red line) the estimated context is plotted. As it can be seen, the representation of the estimated context is negatively correlated to the actual mass of the load: when the actual mass increases, the estimated context decreases and then the actual mass decreases, the estimated context increases. There

is also a difference in the scale of the actual mass and the context variable. In other experiments, the acquired representation could be positively correlated and only the scale between the internal representation and the mass differ. The feedforward and feedback commands are displayed in Fig. 5.14. The arm is controlled for the 20000 first datapoints by a PD controller and then a composite controller that utilizes the estimates is used. It is clear that the context estimates, although not directly representing the mass (which is the main context variable in this case) are clearly useful for control: the error-correcting feedback command is zero most of the time. Also there is almost no tracking error (mean squared error was around 0.00001).

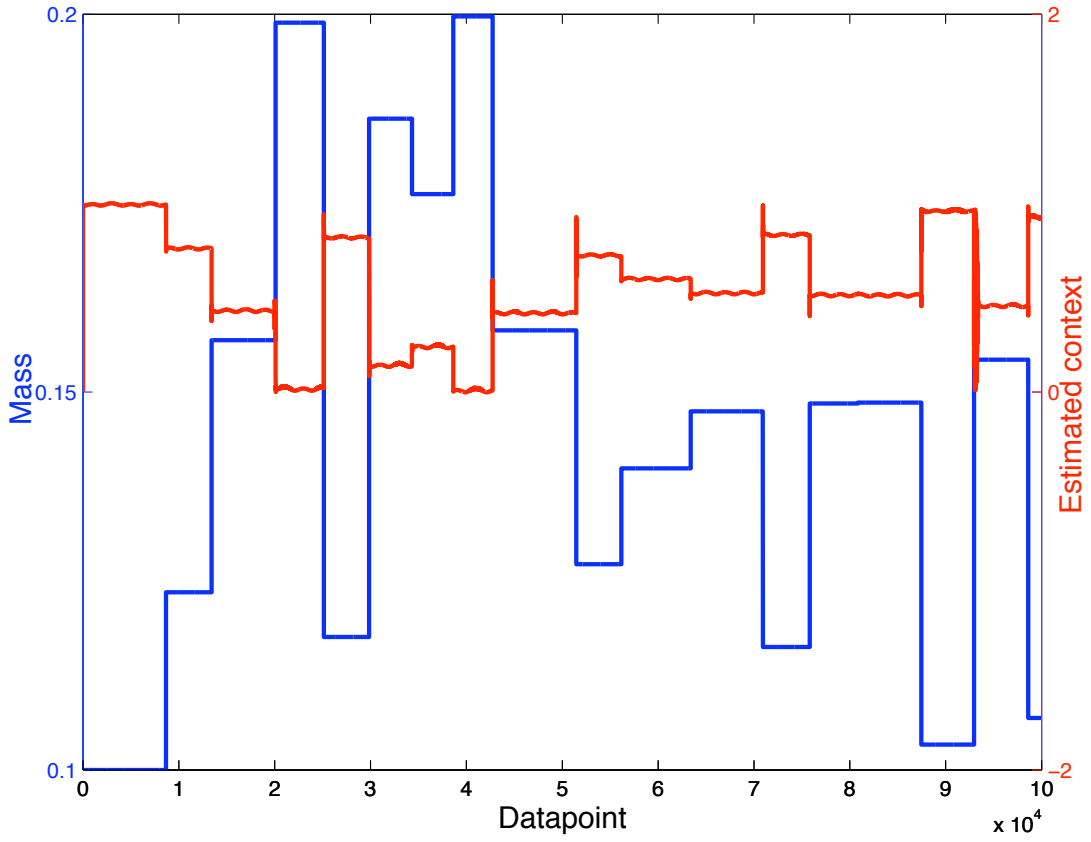


Figure 5.13: Using a nonlinear state-space model for control of a one degree of freedom arm with a nonstationary load. The blue line shows the actual mass of the load (plotted against the left axis) and the red line shows the estimated context (plotted against the right axis). Note that the two vertical axes have different scales. The representation of the context matches the variation of the actual mass (it is negatively correlated to the actual mass of the load).

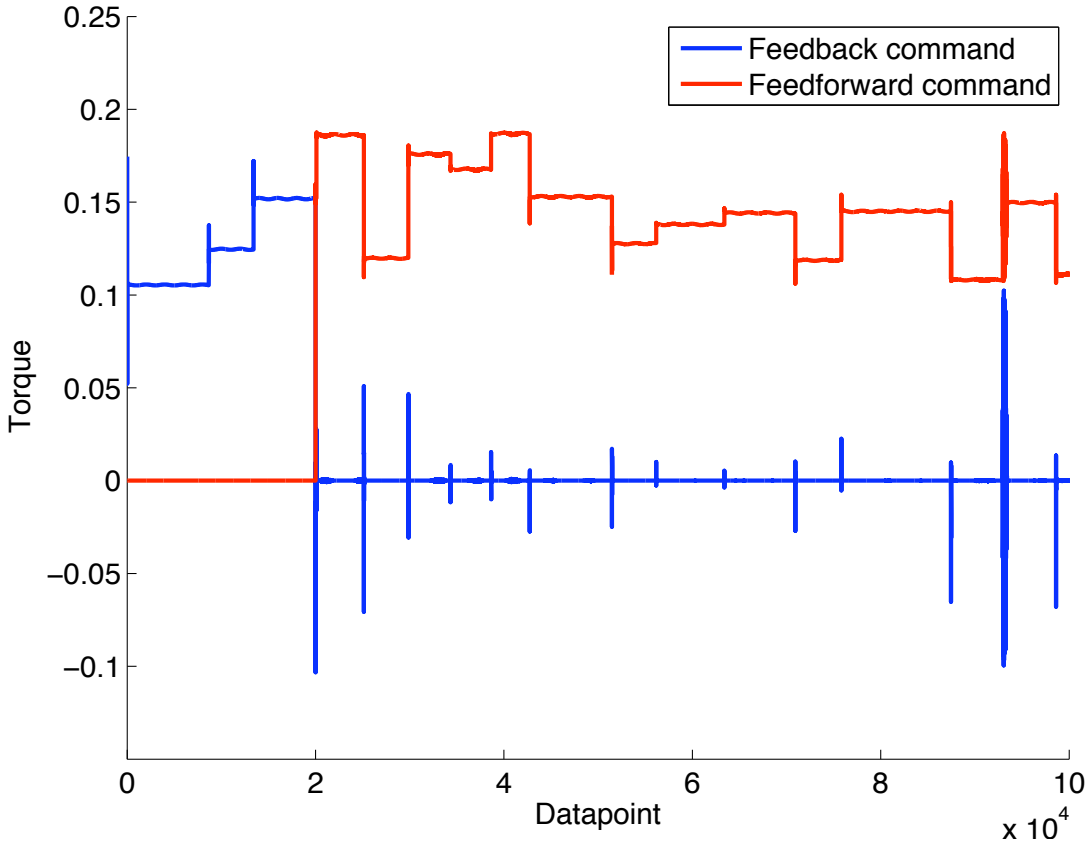


Figure 5.14: Feedback and feedforward commands using a nonlinear state-space model for control of a one degree of freedom arm with a nonstationary load. We switch from feedback to composite control at datapoint 20000. The fact that the feedback command is very small after we switch to composite control implies that the feedforward command is accurate, which in turn implies that the estimated context (Fig. 5.13 is useful for control and that the augmented inverse model represents the dynamics of the arm under different contexts successfully.

5.6 Discussion

In this Chapter, methodologies for generalizing the experience of a limited number of contexts to new contexts have been proposed. Two different scenarios have been examined. In the first, prior knowledge on the relationship of the hidden context to the modulation of the dynamics is used. This was demonstrated on two different setups (learning an augmented inverse dynamics model and learning an augmented tactile model). In the second scenario, no prior knowledge on the relationship of the context to the dynamics is used. As demonstrated, learning in such a model is quite data intensive but feasible.

Stability analysis is very difficult to perform for any of the methods developed in this Chapter. Nevertheless, a change of context did not trigger a persistent increase of error in any of the experiments performed.

Comparing this approach to the multiple model scenario, it has the advantage that generalization is *not only limited to the convex space between the learned set of models* (since positive context responsibilities are used in MPFIM and MOSAIC to combine the predictions of individual models). In the case that prior knowledge is used, generalization is achieved in the whole *space spanned by the reference models*, whereas in the case that prior knowledge is not used, generalization is also achieved at least around the space of experienced dynamics (since local models are used for learning the augmented model). In comparison to MPFIM and MOSAIC in particular, the augmented model presented here also has the advantage that it can deal with nonlinear dynamics. To the best of our knowledge this is the first piece of work for learning dynamics that exhibit nonstationarity under a continuous range of contexts.

Chapter 6

Resolving nonstationarity using observed contextual information

In previous chapters, it was attempted to resolve the problem of nonstationarity by estimating a latent variable that describes the modulation of the dynamics. In the case that the latent variable is discrete, it is used to select one of a set of models to use for control. In the case that it is continuous, it is used as an additional input to the inverse model.

These approaches were useful for scenarios where contextual information was not contained in any observed input variables of the dynamics model. Contextual information was reflected through the modulation of the input-output relationship of the dynamics model. Nevertheless, in some cases, there may be some directly observed sensory input that is closely associated with the context. For example, tactile sensing (Section 5.4) conveys contextual information about the manipulated load, albeit in an indirect way. When such variables that convey contextual information are directly observed, a different and more effective strategy can be used: a single augmented model of the dynamics is learned, similar to the augmented model that took as additional input continuous latent variables, however it takes as additional input these observed variables that carry contextual information. With reference to the schematic of Fig. 5.1, the observed contextual variables take the place of the hidden contextual variables and is used to disambiguate the context. There is no need for context estimation and it is only required to directly learn an augmented model of the form

$$\tau = G(q, \dot{q}, \ddot{q}, S),$$

where S denotes the additional sensory input. The idea is illustrated in Fig. 6.1.

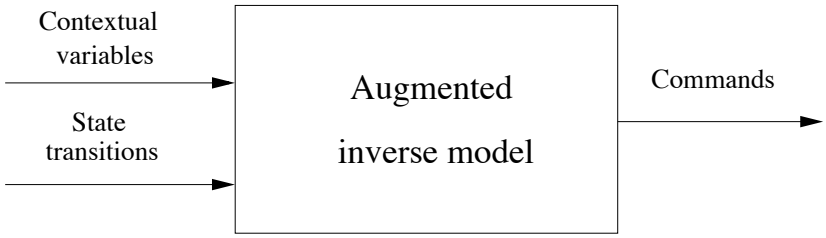


Figure 6.1: An augmented inverse model that takes as additional input sensed variables that convey contextual information.

As already hinted, for the scenario of manipulation of varying loads, this additional sensory input S could be for example tactile sensing. Also, as we will see, this approach is applicable even when such specific sensory input is not available.

6.1 Augmented model using tactile sensing

We have already seen that tactile sensing can be useful for control under nonstationary conditions and extracting hidden continuous context. Tactile sensing can also be used as the additional input of the augmented model. That is, if the tactile input is denoted as T, we can learn the model

$$\tau = G(q, \dot{q}, \ddot{q}, T)$$

and use it for control.

As mentioned, this approach has the advantage that there is no need to do context estimation. The disadvantage is that it requires to learn a model with a higher dimensional input. If the number of additional variables is high, the model may be difficult to learn. Pseudocode for implementing this approach is given in Table 6.1.

6.1.1 Experiments

The use for motor learning and control of an inverse model augmented with tactile sensing has been tested experimentally on the DLR arm. The same setup and target trajectory as in the experiments of Section 5.4 were used. 100 iterations of the trajectory were repeated. with the mass of the load changing randomly. Learning of the augmented model was executed only during the first 80 iterations, while the learned augmented model was used as part of a composite controller as described in Table 6.1 during the whole simulation. Learning is switched off at the 80^{th} iteration to show that

the model is not just adapting very fast to recently seen data but is actually learning an augmented model that is able to disambiguate between varying contexts and is also able to generalize to novel contexts as well. Five runs were executed, with different random changes of the load on each run. Results are presented averaged over the five runs, with bars displaying the standard deviation between runs.

Table 6.1: Pseudocode for learning an inverse model augmented with tactile input and using it for control

```
set up simulation environment;
initialize lwpr model;
//model has (n_joints x 3 + n_tactile_sensors) inputs and n_joints outputs;
for i=1:T
  compute feedforward command;
  //input to lwpr model are the desired current joint angles,
  //velocities, accelerations and last tactile sensing
  compute feedback command;
  apply sum of feedforward and feedback command;
  simulate and observe transitions and tactile sensing;
  train augmented inverse model;
  //use as input the last state transition and tactile sensing
  //and as output the applied composite command
end
```

On Fig. 6.2 (left) the ratio of feedback to composite command is displayed and in Fig. 6.4 (right) the tracking error is displayed. Both are very low and remain low even after learning is switched off showing that indeed the tactile sensing helps to disambiguate the context and also that the learned model is useful for generalizing to novel contexts as well.

6.2 Direct mapping using previous state transition and command

The idea of the augmented model that uses observed contextual sensory input is applicable even when specialized sensory input, like tactile sensing, is not available. The idea is to use directly variables that are anyway available and describe the context, i.e. use the variables that are used to do context estimation as the additional sensory input

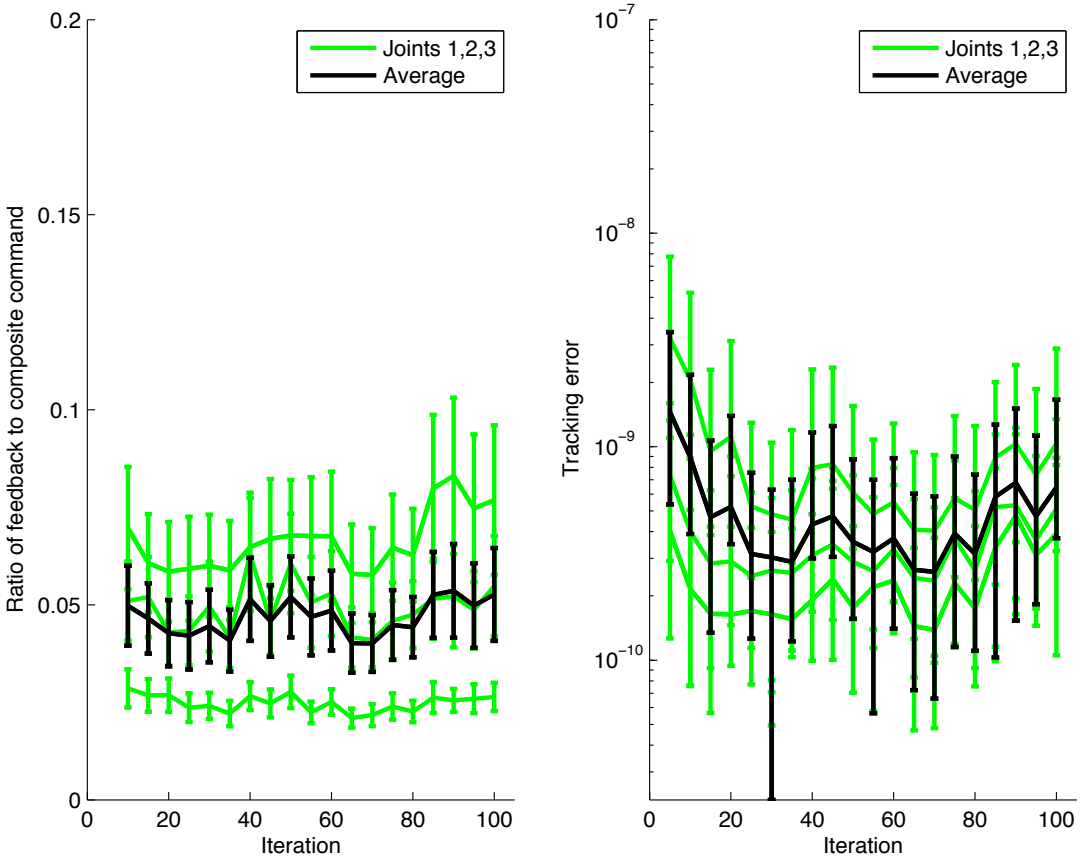


Figure 6.2: Using an inverse model augmented with tactile sensory input for control of the 3 DOF arm. Evolution of the ratio of feedback to composite command (left) and tracking error over 100 iterations of the target trajectory. Results are averaged over five trials and the bars show the standard deviation between trials.

in the augmented model. That is, we learn the model:

$$\tau_t = G(q_t, \dot{q}_t, \ddot{q}_t, q_{t-1}, \dot{q}_{t-1}, \ddot{q}_{t-1}, \tau_{t-1}) \tag{6.1}$$

This depends on the previous time step's state transition and command. This approach is also inspired by (Gomi and Kawato, 1993), where time delayed state transitions and commands were used as inputs to the gating network in their multiple model system. This is an instance of an autoregressive model with exogenous inputs (ARX model) (Ljung, 1998). If we denote the time series variable as y_t and the exogenous input as x_t, an ARX model is:

$$y_t = f(y_{t-1}, \ldots y_{t-m}, x_t, x_{t-1}, \ldots x_{t-n}). \tag{6.2}$$

That is, the time series depends on the last m values of the output and the last $n+1$ values of the exogenous input. When the function f is nonlinear, the model is referred to

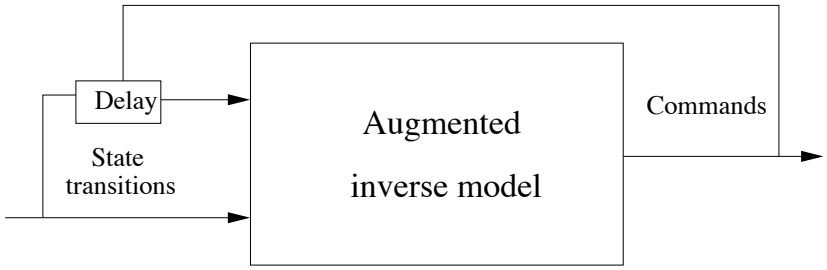


Figure 6.3: An augmented inverse model that takes as additional input time delayed state transitions and commands.

as a nonlinear autoregressive model with exogenous inputs (NARX model) (Leontaritis and Billings, 1985). The augmented model in (6.1) is a NARX model of the form (6.2) with $m = 1$ and $n = 1$. Inclusion of higher order terms would be beneficial for handling sensory noise (similarly to the introduction of the temporal relationship on the hidden contextual variable in the probabilistic formulations of the previous chapters) but would increase the input dimensionality significantly and would therefore require more training data. When there are low levels of noise though, the last transition only is sufficient to disambiguate the context and the model in (6.1) should be used. NARX models have previously been used in adaptive control (Mirizarandi et al., 2005). Pseudocode for implementing this approach is given in Table 6.2.

Table 6.2: Pseudocode for learning an inverse model augmented with time delayed state transition data and using it for control

```
set up simulation environment;
initialize lwpr model;
//model has (n_joints x 7) inputs and n_joints outputs;
for i=1:T
  compute feedforward command;
  //input to lwpr model are the desired current joint angles,
  //velocities, accelerations, previous joint angles, velocities,
  //accelerations and last applied commands
  compute feedback command;
  apply sum of feedforward and feedback command;
  simulate and observe transitions;
  train augmented inverse model;
  //use as input the previous two state transitions and previous
  //composite command and as output the last applied composite command
end
```

6.2.1 Experiments

The idea of augmenting the inverse model with time delayed state transitions and commands has been tested on the 3 DOF simulated robot arm. 125 iterations of the same sinusoidal trajectory were executed, with the mass of the manipulated object changing randomly. Five runs were repeated with different random changes of the mass in order to collect statistics. As in the experiments with the tactile input, learning was switched off to examine how well the learned model could cope with the nonstationary dynamics, without being adapted constantly and using newly acquired experience. Learning stopped at the 100^{th} iteration. The results can be seen in Fig. 6.4. On Fig. 6.4 (left) the ratio of feedback to composite command is displayed and in Fig. 6.4 (right) the tracking error is displayed. Similarly to the case where tactile sensing was used as input to the augmented inverse model, both are very low and remain low even after learning is switched off showing that indeed the time delayed state transition and command help to disambiguate the context and that the learned model is useful for generalizing to novel contexts as well.

6.3 Discussion

The described method has the disadvantage that a higher dimensional regression problem needs to be learned. Depending on the nature of the contextual sensors, this may be an important issue or not. In the experiments, the additional tactile sensing was five-dimensional. Nevertheless, data coming from real tactile sensors will in general have much higher dimensionality. It could be argued that since LWPR uses PLS which effectively does local dimensionality reduction, this may not be a big issue. However, it could be more appropriate in some cases to try some other preprocessing of the additional contextual variables before actually feeding it to the augmented inverse model.

Furthermore, although the issue of higher dimensional input may seem quite daunting, it is the case that actually a not as large part of the input space of the augmented model is visited or explored in practice. In general, the additional variables will be correlated to some of the other input variables. It is mostly so in the case of time delayed state transitions and commands, the previous joint angles and velocities will most of the time be very similar to the current joint angles and velocities and thus a relatively small part of the input space should be relevant. Nevertheless, it should still

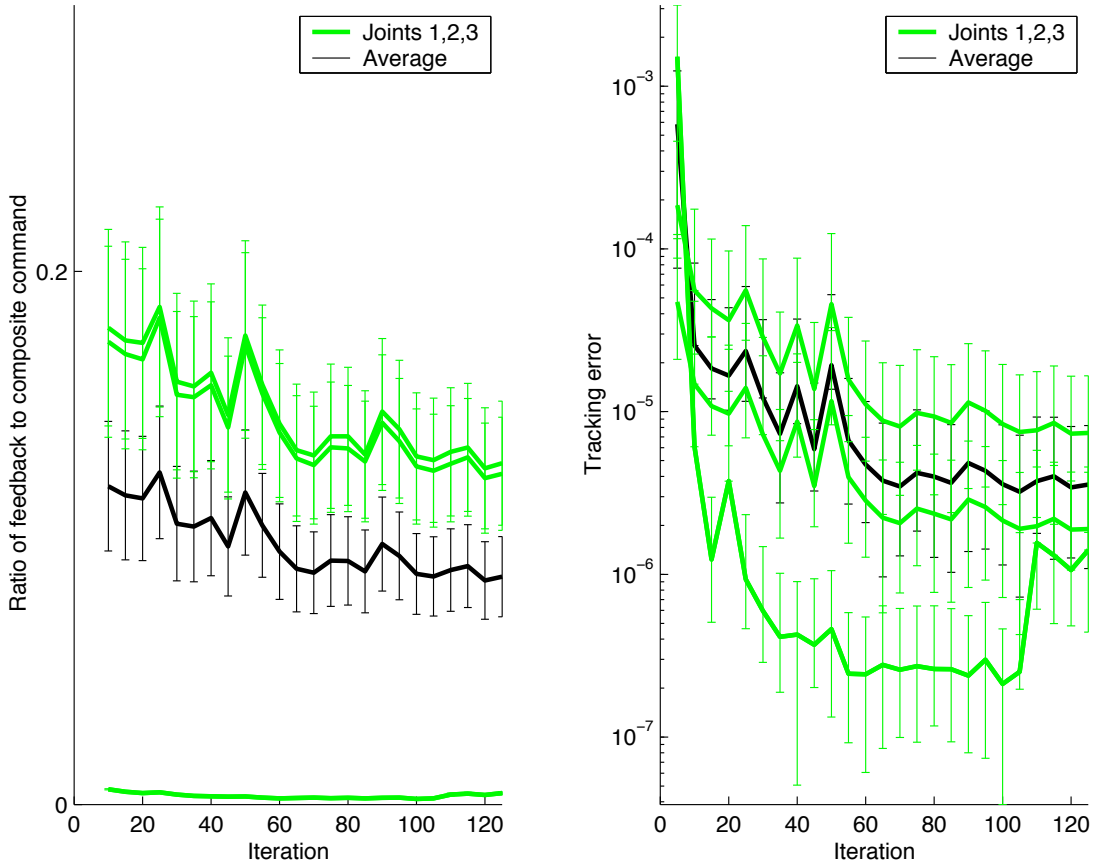


Figure 6.4: Using an inverse model augmented with time delayed state transitions and commands for control of the 3 DOF arm. Evolution of the ratio of feedback to composite command (left) and tracking error over 120 iterations of the target trajectory. Results are averaged over five trials and the bars show the standard deviation between trials.

be expected that – due to the increased dimensionality – more training data would be required for learning the models sufficiently well.

Most importantly, this approach, just like the linear and nonlinear state-space models that were developed in the previous chapter, manage to generalize knowledge from previously experienced contexts to novel contexts as well. In comparison to methods that can also achieve generalization to novel contexts and use prior knowledge on the relationship of the context to the modulation of the dynamics, the advantage of this approach is that no context estimation is required and thus control is simpler. The disadvantage is that it requires more training data, due to the higher dimensional problem. Fig. 6.5 displays this. A NARX model has been trained on the dynamics of the 3 DOF arm executing the same sinusoidal trajectory under varying loads. Five different repetitions were executed, with increasing number of training datapoints (50000,

100000, 200000, 300000, 400000). Each iteration was followed by testing on 100000 datapoints and for each repetition five runs were repeated to accumulate statistics. The ratio of feedback to composite command of these five repetitions is displayed with the red line in Fig. 6.5. In comparison, similar experiments were repeated using the augmented inverse model with the mass as the unobserved contextual variable. Only two reference models were used for obtaining the inverse model and again five different repetitions were executed with an increasing number of training datapoints that the two reference models saw (10000, 20000, 30000, 40000, 50000). Again five different runs were repeated to obtain performance statistics and the ratio of feedback to composite command can be seen with the blue line along with the standard deviation between runs. Please note that the datapoints axis does not follow a linear scale. It can be seen that the NARX model required 400000 datapoints to reach the performance of the latent variable model that has been trained with 50000 datapoints. Noticing also the bars that show the standard deviation between runs, the difference in performance achieved with the hidden variable model using 50000 datapoints seems to be significantly lower than the performance achieved by the NARX model using 50000, 100000, 200000 and 300000 training datapoints.

Regarding stability, there have been no experiments where a persistent increase of error was observed. Nevertheless, stability analysis is again very difficult to perform and therefore no firm statements about stability can be made.

In conclusion, the solution of a model that takes as additional input observed contextual variables may be suitable when there is no prior knowledge on the relationship of the context to the modulation of the dynamics, albeit in a data rich environment. If there is not much training data or if there is prior knowledge on the nature of the context, the latent variable model seems advantageous as it can achieve superior performance with less data.

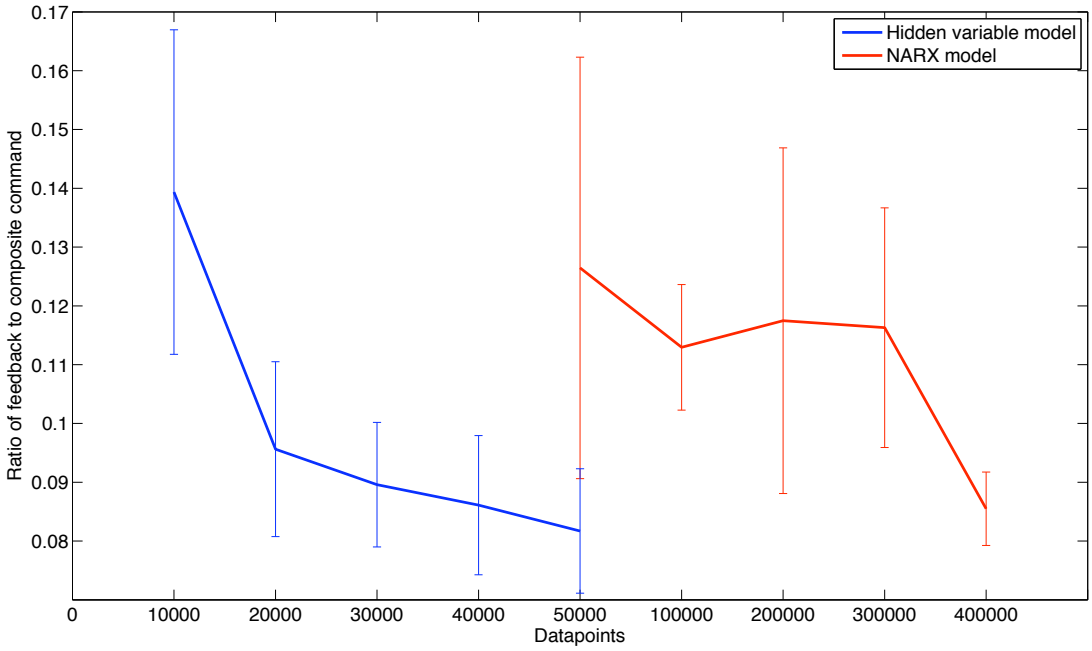


Figure 6.5: Ratio of feedback to composite command as the number of training datapoints increases for a NARX model (blue line) and an augmented model where the mass of the load is the hidden contextual variable (red line). Note that the axis on the number of datapoints is separated in two parts and does not follow a linear scale. The NARX model requires 400000 training datapoints to reach the ratio that the augmented model achieves with 50000 training datapoints.

Chapter 7

Contributions and future work

This thesis examined problems that nonstationarity of dynamics due to varying contexts poses to the task of learning dynamics for robot control. Two goals were set:

- The first goal was to develop methods for reusing experienced dynamics in order to improve performance when previously seen contexts reappear.
- The second goal was to develop methods for using previously experienced dynamics to perform well in novel contexts.

With regard to the first goal, an existing multiple model approach has been extended. With regard to the second goal, a novel approach has been formulated that uses an augmented inverse model with input continuous contextual variables (observed or latent) as additional input.

The contributions of this work are summarized in the next section and suggestions for further work are given in the concluding section of the thesis.

7.1 Contributions

The contributions of this thesis are:

- **Learning of multiple models of nonlinear dynamics for nonstationary contexts**. Learning of multiple models for a set of contexts and selecting the appropriate one based on the currently observed dynamics resolves the problem of having to relearn previously experienced dynamics and improves the transient response of the system. The use of multiple models has been studied before. Nevertheless, either no learning was involved (MMST) or learning was limited

to models with linear dynamics (MPFIM and MOSAIC). In this thesis, multiple nonlinear models of nonstationary nonlinear dynamics are learned. A robust local learning algorithm (LWPR) is used to learn the nonlinear dynamics and a principled probabilistic formulation is developed to treat with model uncertainties (Section 4.2). To overcome the difficulties posed by using a local learner for the individual models, a modified Expectation Maximization algorithm (Table 4.1), has been proposed.

- **Reformulation of the probabilistic multiple model paradigm** in order to deal more effectively with **continuously varying contexts** for the scenario of a robot manipulator carrying different loads (Section 5.3). A special form of a linear state-space model is used. This relates to classic load estimation techniques from robotics and can be referred to as load estimation using learned dynamics models. This model uses prior knowledge on the relationship of the context to the dynamics and is essentially able to generalize the knowledge obtained from learning to manipulate a set of loads to other loads in a principled way .

- **Use of a nonlinear state-space model to learn an augmented model of nonstationary dynamics**. This generalizes the linear state-space model that is formulated in Section 4.2 and is useful when knowledge of the relationship of context to the modulation of the dynamics is not known (Section 5.5).

- **Learning and use for control of dynamics models augmented with observed additional contextual variables**. These additional contextual variables could either come from proper sensory input (e.g. for the scenario of varying loads, tactile sensing) (Section 6.1), or – if special sensory input is not available – from time delayed input and output of the regular dynamics model (essentially by using a first order nonlinear autoregressive model with exogenous input)(Section 6.2). This approach is suitable when there is lack of knowledge on the nature of the context and there is abundance of training data.

7.2 Further work

Some suggestions for further research and improvement of the methods proposed in this thesis are discussed next.

- As discussed in Section 5.5.2, there are other methods for learning nonlinear

state-space models and some of them could be potentially useful. For example, the approach of (Briegel and Tresp, 1999) could provide more accurate estimates in the E-step than particle filtering and could therefore converge faster than the Monte Carlo EM that has been used (Table 5.1). Also, using the Bayesian model of (Valpola and Karhunen, 2002) one could introduce priors on the parameters of the model and could therefore encourage some preferred representation for the hidden contextual variable.

- The data separation procedure for the multiple model paradigm that has been described is essentially an offline batch method. As it was pointed out, since in the E step of the EM algorithm, filtered instead of smoothed estimates are used for learning the models (smoothed estimates are used to estimate the transition probabilities though), it should be possible to use this approach for online data separation. Examining if the data separation method described can be used as the basis for a completely online scenario would be interesting.

- The augmented model with observed contextual variables may present a challenging high dimensional regression problem. This can be quite acute when time delayed state transitions and commands are used and we have a system with many DOFs. LWPR is appropriate for high dimensional problems since it performs local dimensionality reduction. Nevertheless, it would be worth examining if a preprocessing step to decrease the input dimensionality can reduce the complexity of the regression problem and therefore the requirement for large amounts of data (of course without sacrificing the accuracy of the learned model).

- Stability analysis has not been performed for any of the methods that have been developed in this thesis. We only made empirical comments on the stability, based on the experiments. Stability analysis of learning control systems is in general difficult. Analysis in the proposed methods is even more difficult due to the fact that LWPR is a complicated nonlinear regression method that depends critically on a large number of parameters. In addition there is not much previous work on stability of adaptive control with local models (Nakanishi et al., 2005).

- Investigation of the applicability of multi-task regression methods (Bonilla et al., 2008; Evgeniou et al., 2005) would also be interesting as discussed in Section 2.4. A first study can be found in (Chai et al., 2008), where multi-task

Gaussian process regression is used to learn a set of inverse dynamics models of a manipulator with different loads.

- Application of the developed methods to other nonstationary context scenarios. In all experiments, nonstationarity of the dynamics was due to manipulation of different loads. Nevertheless, all methods – with the exception of methods that require prior knowledge on the nature of the context and the way it modulates the dynamics – can be applied to other nonstationary contexts as well. Other nonstationary context scenarios have already been mentioned. E.g. learning dynamics under different force fields or inside liquids of different viscosity. Other interesting applications could be learning of dynamics in different gravity fields, something that could be useful in space robotics.

Appendix A

Sample ODE code for simulating the 3 DOF arm

This Appendix provides sample C++ code for setting up the simulation of the 3 DOF arm (Fig. 3.2) that is used in the experiments.

```
#include <ode/ode.h>

int main(int argc, char **argv){
  //First create a dynamics ODE world
  dWorldID world=dWorldCreate();
  //Then create a collision space
  //(we will not use it in this simulation)
  dSpaceID space=dSimpleSpaceCreate(0);;
  //Set gravity vector
  dWorldSetGravity (world,0,0,-9.81);

  //A set of bodies
  dBody *body;
  //A set of hinge joints
  dHingeJoint *joint;
  //A set of motors (attached to the moving joints)
  dAMotorJoint *motor;
  //A set of fixed joints
  dFixedJoint *fixed;
  //A set of box shapes (for collision)
  dBox *box;
  //Set the number of bodies, boxes,
  //fixed joints, hinge joints and motors
  uint armJoints=3;
  uint Nbody,Njoint,Nmotor,Nbox,Nfixed;
  Nbody=Nbox=armJoints+1;
  Nfixed=1;
  Njoint=Nmotor=armJoints;
  //create the necessary ODE objects
  body=new dBody[Nbody];
  joint=new dHingeJoint[Njoint];
```

```
motor=new dAMotorJoint[Nmotor];
box=new dBox[Nbox];
fixed=new dFixedJoint[Nfixed];

//matrices for keeping actual joint angles and velocities
double joint_angles[Njoint];
double joint_velocities[Njoint];
double torques[Njoint];
//mass-inertia object
dMass m;
//joint angle limits (in degrees)
double angle=70;

//mass of each link
double mass=1.;
//create the body for the pole of the arm
body[0].create(world);
//set its position (in world coordinates always)
body[0].setPosition(0,0,0.5);
//create the shape of the pole
box[0].create(space,0.2,0.2,1.);
//associate the shape with the body
box[0].setBody(body[0]);
//attach rigidly the pole to the
//world with a fixed joint
fixed[0].create(world);
fixed[0].attach(body[0],0);
fixed[0].set();

//then create the three links
//and joints or the robot
uint i,j;
for(i=1;i<Nbody;i++){
  //create the body of the i^th link
  body[i].create(world);
  //set the link's position
  body[i].setPosition(0,-.5+i,1.0);
  //calculate  and set its inertial parameters
  m.setBox(mass,0.2,1.0,0.2);
  body[i].setMass(&m);
  //create a shape of the link
  //(for collision detection)
  box[i].create(space,0.2,1.0,0.2);
  //associate the shape with the body
  box[i].setBody(body[i]);
}
//create the joints between the bodies
for(i=0; i<Njoint; i++){
  //create the i^th joint
  joint[i].create(world);
  //attach the two links of the joint
  joint[i].attach(body[i],body[i+1]);
  //set the position of the joint
  joint[i].setAnchor(0,(float)i,1.0);
  //set the axis of rotation
  if(i==0)
```

```
    //the first joint rotates around the x axis
    joint[i].setAxis(-1.,0.,0.);
  else
    //and the next joints rotate around the z axis
    joint[i].setAxis(0.,0.,-1.);
  //set joint angle limits
  joint[i].setParam(dParamLoStop,-angle);
  joint[i].setParam(dParamHiStop, angle);
}
//In a similar way, we can create another body and
//attach it to the last link with a fixed joint
//to simulate a changing load
```

Appendix B

Numerically stable inference for Hidden Markov Models

The inference equations for the Hidden Markov Model in Section 4.2.2 and Section 4.2.3 are not numerically stable. For example, consider the recursive update for $\alpha(c_t)$ in (4.15). Given that (4.15) involves a product of quantities that are smaller than 0 and usually much smaller than 0, the recursion leads very fast to very small values for $\alpha(c_t)$ that fall below machine precision. This Appendix gives numerically stable inference equations for a Hidden Markov Model. The discussion follows Section 13.2.4 in Bishop (2006).

The original $\alpha(c_t)$ variables were defined as $\alpha(c_t) = p(c_t, \theta_{1:t+1}, \tau_{1:t})$. We define the normalized variables $\hat{\alpha}(c_t)$ as

$$\hat{\alpha}(c_t) = p(c_t|\theta_{1:t+1}, \tau_{1:t}) = \frac{\alpha(c_t)}{p(\theta_{1:t+1}, \tau_{1:t})}. \tag{B.1}$$

The $\hat{\alpha}(c_t)$ for different values of c_t represent a probability distribution and should behave well numerically (since they sum to one). The normalized variables $\hat{\alpha}(c_t)$ are essentially the filtered estimates in (4.14).

In order to relate the original variables $\alpha(c_t)$ with the new variables $\hat{\alpha}(c_t)$ we introduce scaling factors z_t. These are defined as:

$$z_t = p(\theta_{t+1}, \tau_t|\theta_{1:t}, \tau_{1:t-1}) \tag{B.2}$$

Then,

$$p(\theta_{1:t+1}, \tau_{1:t}) = \prod_{i=1}^{t} z_i \tag{B.3}$$

and

$$\alpha(c_t) = (\prod_{i=1}^{t} z_i)\hat{\alpha}(c_t) \tag{B.4}$$

Using (B.4) to substitute $\alpha(c_t)$ and $\alpha(c_{t+1})$ in (4.15), we obtain the recursive formula for $\hat{\alpha}(c_{t+1})$:

$$z_{t+1}\hat{\alpha}(c_{t+1}) = \sum_{c_t} \hat{\alpha}(c_t) p(c_{t+1}|c_t) p(\tau_{t+1}|\theta_{t+1}, \theta_{t+2}, c_{t+1}) \tag{B.5}$$

The scaling factor z_{t+1} can be very easily computed since it is the constant that normalizes the right-hand side of (B.5).

Similarly, we define normalized versions of $\beta(c_t)$ and we obtain the following backward recursion:

$$z_{t+1}\hat{\beta}(c_t) = \sum_{c_{t+1}} \hat{\beta}(c_{t+1}) p(c_{t+1}|c_t) p(\tau_t|\theta_t, \theta_{t+1}, c_t) \tag{B.6}$$

The scaling factors z_t are the same that were computed in the forward phase. We can now use the $\hat{\alpha}(c_t)$ and $\hat{\beta}(c_t)$ to obtain the smoothed estimates as:

$$p(c_t|\theta_{1:T}, \tau_{1:T}) = \hat{\alpha}(c_t)\hat{\beta}(c_t) \tag{B.7}$$

Finally the probabilities $p(c_t, c_{t+1} \,|\, \theta_{1:T}, \tau_{1:T})$ are computed as:

$$p(c_t, c_{t+1} \,|\, \theta_{1:T}, \tau_{1:T}) = z_t \hat{a}(c_t) p(\tau_{t+1}|\theta_{t+1}, \theta_{t+2}, c_{t+1}) \hat{\beta}(c_{t+1}) p(c_{t+1}|c_t) \tag{B.8}$$

Appendix C

Derivation of linear relation of dynamics to the inertial parameters

We show that the dynamics of a manipulator has a linear relationship to a properly defined set of inertial parameters of the links. This fact is used in Section 5.3 in order to obtain the augmented inverse dynamics model from a set of learned dynamics models.

The Lagrange formulation for deriving dynamics will be used (Sciavicco and Siciliano, 2000; Spong et al., 2006). We follow closely the description in (Sciavicco and Siciliano, 2000). We define the Lagrangian:

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \tag{C.1}$$

where $\mathcal{T}$ is the kinetic energy of the system and $\mathcal{U}$ is the potential energy of the system. Then, given that $q_1, q_2 ... q_n$ are the joint angles and $\tau_1, \tau_2 ... \tau_n$ are the generalized forces associated with the corresponding joint angles, the dynamics are given by:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i \tag{C.2}$$

The generalized force τ_i, associated with joint angle q_i, is a sum of joint actuator torques, joint friction torques or other forces acting on the joint (e.g. forces induced by contact with the environment). The total kinetic energy $\mathcal{T}$ is just the sum of the kinetic energy of all the links of the manipulator:

$$\mathcal{T} = \sum_{j=1}^{n} \mathcal{T}_{l_j} \tag{C.3}$$

The kinetic energy for link j is given by:

$$\mathcal{T}_{l_j} = \frac{1}{2} m_{l_j} \dot{p}_{l_j}^T \dot{p}_{l_j} + \frac{1}{2} \omega_j^T I_{l_j} \omega_j \tag{C.4}$$

Here, p_{l_j} is the center of mass vector of link j expressed in the base frame and $\dot{p}_{l_j}$ is the linear velocity of the center of mass. Also, ω_j is the vector of rotational velocity of link j expressed in the base frame and I_{l_j} is the inertia tensor when expressed in the base frame:

$$I_{lj} = \begin{bmatrix} \int (r_{jy}^2 + r_{jz}^2)\rho dV & -\int r_{jx} r_{jy} \rho dV & -\int r_{jx} r_{jz} \rho dV \\ -\int r_{jx} r_{jy} \rho dV & \int (r_{jx}^2 + r_{jz}^2)\rho dV & -\int r_{jy} r_{jz} \rho dV \\ \int r_{jx} r_{jz} \rho dV & -\int r_{jy} r_{jz} \rho dV & \int (r_{jx}^2 + r_{jy}^2)\rho dV \end{bmatrix} \tag{C.5}$$

In this, the integrals are over the volume V of the link, ρ denotes the mass density at the point p in the link and the vector r is the position vector p minus the position vector of the center of mass p_{l_j}. Note that the inertia tensor is symmetric, i.e. it has only six independent elements.

The parallel axis theorem states that if we translate the position around which the inertia tensor is evaluated, the inertia tensor becomes:

$$I^{new} = I^{old} + m_{l_j} S^T(l) S(l) \tag{C.6}$$

where l is the translation vector with elements $[l_x, l_y, l_z]$ and $S(l)$ is the array:

$$S(l) = \begin{bmatrix} 0 & -l_z & l_y \\ l_z & 0 & -l_x \\ -l_y & l_x & 0 \end{bmatrix} \tag{C.7}$$

We will attach a reference frame to each link of the manipulator with origin instead of at the center of mass, exactly at the joint. If the origin of the link's reference frame expressed in the base frame is p_j, then the center of mass of the link can be written as

$$p_{l_j} = p_j + l_j \tag{C.8}$$

And the linear velocity $\dot{p}_{l_j}$ is:

$$\dot{p}_{l_j} = \dot{p}_j + \omega_j \times l_j \tag{C.9}$$

If we substitute this to the kinetic energy equation (C.4) we get:

$$\mathcal{T}_{l_j} = \frac{1}{2} m_{l_j} (\dot{p}_j + \omega_j \times l_j)^T (\dot{p}_j + \omega_j \times l_j) + \frac{1}{2} \omega_j^T I_{l_j} \omega_j \tag{C.10}$$

The vector product can be substituted with the matrix-vector multiplication $S(\omega_j) l_j$. Using that, we get:

$$\mathcal{T}_{l_j} = \frac{1}{2} m_{l_j} \dot{p}_j^T \dot{p}_j + \dot{p}_j^T S(\omega_j) m_{l_j} l_j + \frac{1}{2} m_{l_j} l_j^T S(\omega_j)^T S(\omega_j) l_j + \frac{1}{2} \omega_j^T I_{l_j} \omega_j \tag{C.11}$$

Now, using the fact that $S(a)b = -S(b)a$ we get:

$$\mathcal{T}_{l_j} = \frac{1}{2} m_{l_j} \dot{p}_j^T \dot{p}_j + \dot{p}_j^T S(\omega_j) m_{l_j} l_j + \frac{1}{2} m_{l_j} \omega_j^T S(l_j)^T S(l_j) \omega_j + \frac{1}{2} \omega_j^T I_{l_j} \omega_j \tag{C.12}$$

We now see that we that we can use the parallel axis theorem to express the inertia tensor around the new coordinate frame. We get:

$$\mathcal{T}_{l_j} = \frac{1}{2} m_{l_j} \dot{p}_j^T \dot{p}_j + \dot{p}_j^T S(\omega_j) m_{l_j} l_j + \frac{1}{2} \omega_j^T I_{l_j}^{new} \omega_j \tag{C.13}$$

The total potential energy $\mathcal{U}$ is the sum of the potential energy of all the links of the manipulator:

$$\mathcal{U} = \sum_{j=1}^{n} \mathcal{U}_{l_j} \tag{C.14}$$

The potential energy of link j is:

$$\mathcal{U}_{l_j} = -\int_{V_{l_j}} g_0^T p_j \rho dV = -m_{l_j} g_0^T p_{l_j} \tag{C.15}$$

where g_0 is the gravity acceleration vector. Again using

$$p_{l_j} = p_j + l_j$$

we get:

$$\mathcal{U}_{l_j} = -m_{l_j} g_0^T p_j - m_{l_j} g_0^T l_j \tag{C.16}$$

Substituting (C.13) in (C.3) and (C.16) in (C.14) and then using (C.1), we obtain:

$$\mathcal{L} = \sum_{j=1}^{n} \frac{1}{2} m_{l_j} \dot{p}_j^T \dot{p}_j + \dot{p}_j^T S(\omega_j) m_{l_j} l_j + \frac{1}{2} \omega_j^T I_{l_j}^{new} \omega_j + m_{l_j} g_0^T p_j + m_{l_j} g_0^T l_j \tag{C.17}$$

We can see that the Lagrangian has a linear relationship to the set of inertial parameters:

$$\pi_j = [m_{l_j}, m_{l_j} l_{jx}, m_{l_j} l_{jy}, m_{l_j} l_{jz}, I_{l_j xx}, I_{l_j xy}, I_{l_j xz}, I_{l_j yy}, I_{l_j yz}, I_{l_j zz}] \tag{C.18}$$

(we defined previously I to be a matrix, here we lay down I's six independent elements in a vector) the Lagrangian can be written in the form:

$$\mathcal{L} = \sum_{j=1}^{n} g_j(q, \dot{q})^T \pi_j \tag{C.19}$$

Since the inertial parameters in π do not depend on time or $\dot{q}$ then, using (C.2), the dynamics equation for joint i is:

$$\sum_{j=1}^{n} \pi_j^T \frac{d}{dt} \frac{\partial g_j(q, \dot{q})}{\partial \dot{q}_i} - \pi_j^T \frac{\partial g_j(q, \dot{q})}{\partial q_i} = \tau_i \tag{C.20}$$

Thus, the dynamics can be written in the form

$$\sum_{j=1}^{n} y_{ij}(q,\dot{q},\ddot{q})^T \pi_j = \tau_i \tag{C.21}$$

where $y_{ij}(q,\dot{q},\ddot{q})$ is a function $\mathbb{R}^{3n} \mapsto \mathbb{R}^{10}$ and $\pi_j \in \mathbb{R}^{10}$. This last relationship shows that the dynamics are linear in the inertial parameters vectors π_j. Compiling the dynamics equations of the different models to a single equation we have:

$$Y(q,\dot{q},\ddot{q})\pi = \tau \tag{C.22}$$

where $Y(q,\dot{q},\ddot{q})$ is a function $\mathbb{R}^{3n} \mapsto \mathbb{R}^{n\times 10n}$ and compiles the functions y as

$$\begin{bmatrix} y_{11}^T & y_{12}^T & \cdots & y_{1n}^T \\ y_{21}^T & y_{22}^T & \cdots & y_{2n}^T \\ \vdots & \vdots & \vdots & \vdots \\ y_{n1}^T & y_{n2}^T & \cdots & y_{nn}^T \end{bmatrix} \tag{C.23}$$

and the vector $\pi \in \mathbb{R}^{10n}$ compiles all the links' inertial parameters as:

$$\begin{bmatrix} \pi_1^T \pi_2^T \ldots \pi_n^T \end{bmatrix}^T \tag{C.24}$$

and $\tau \in \mathbb{R}^n$ includes the torque applied at each joint.

Appendix D

Monte Carlo EM for learning a nonlinear state-space model

We are going to develop EM (Dempster et al., 1977) update equations for learning a nonlinear state-space model (Table 5.1). We first formulate the nonlinear state-space model and then proceed with the EM algorithm for learning its parameters.

The state-space model has the structure of Fig. 4.1, with the difference that we will replace states θ_t with joint angles q_t and velocities $\dot{q}_t$. We have to define probability distributions for the nodes in the graph. Let start from the observation (augmented inverse) for τ_t. We will assume that – due to modeling inaccuracies – the augmented (nonlinear) inverse model g is affected by Gaussian noise ε_{obs} with zero mean and covariance U_{obs}, i.e.:

$$\tau_t = g(q_t, \dot{q}_t, \ddot{q}_t, c_t) + \varepsilon_{obs} \tag{D.1}$$

For notational simplicity we will drop the dependency on the observed variables q_t,$\dot{q}_t$ and $\ddot{q}_t$ and we will simply denote $g(q_t, \dot{q}_t, \ddot{q}_t, c_t)$ by $g_t(c_t)$. Thus, the observation model is given by:

$$p(\tau_t | q_t, \dot{q}_t, \ddot{q}_t, c_t) = \mathcal{N}(g_t(c_t), U_{obs}) \tag{D.2}$$

The transition model for c_t also needs to be defined. In general, we cannot expect that the context will vary in a predictable manner. However, it is reasonable to expect that the context c_t will remain constant most of the time and we will add Gaussian noise to deal with the state transitions, i.e.:

$$c_{t+1} = c_t + \varepsilon_{tr} \tag{D.3}$$

Where ε_{tr} is zero mean noise with covariance U_{tr}. Thus, the transition model is:

$$p(c_{t+1} | c_t) = \mathcal{N}(c_t, U_{tr}) \tag{D.4}$$

Normally, we would have to define the distributions of the variables q_t and $\dot{q}_t$, although they are assumed to be always observed. Nevertheless, we will use the graphical model to do context estimation, i.e. compute the posterior $p(c_t|q_{1:t+1}, \dot{q}_{1:t}, \tau_{1:t})$ (filtered estimates, for control) or the posterior $p(c_t|q_{1:T}, \dot{q}_{1:T}, \ddot{q}_{1:T}, \tau_{1:T})$ (smoothed estimates, for learning) and for these knowledge of the distribution of q_t and $\dot{q}_t$ is not required. Thus, we will not estimate the distributions for q_t and $\dot{q}_t$ – $p(q_t)$ and $p(\dot{q}_t)$ respectively – in the EM.

Now, having a sequence of observations $q_{1:T}$, $\dot{q}_{1:T}$, $\ddot{q}_{1:T}$ and $\tau_{1:T}$ the problem is to estimate the model g in Eq. D.1 and the noise variances U_{obs} and U_{tr}. We will formulate an EM algorithm to solve this problem.

In the E-step, we will use particle methods rather than a parametric representation for the posterior $p(c_t|q_{1:T}, \dot{q}_{1:T}, \ddot{q}_{1:T}, \tau_{1:T})$ of the hidden variable given the observed data . That is, we will use a set of samples with corresponding weights as in (5.38). In the experiments, for computational reasons, we approximate the posterior using filtered (Section 5.5.1) instead of smoothed estimates, since particle smoothing is very expensive computationally.

Now, let us compute the update equations for the M-step. The complete data likelihood is:

$$\begin{aligned} P(q_{1:T}, \dot{q}_{1:T}, \ddot{q}_{1:T}, \tau_{1:T}, c_{1:T}) &= p(c_0) \prod_{t=1}^{T-1} p(c_{t+1}|c_t) \prod_{t=1}^{T} p(\tau_t|q_t, \dot{q}_t, \ddot{q}_t, c_t) \\ & \prod_{t=1}^{T-1} p(q_t) \prod_{t=1}^{T-1} p(\dot{q}_t) \end{aligned} \tag{D.5}$$

Taking the logarithm of this quantity and omitting the terms that do not depend on the parameters that we want to estimate, we have the following expression for the complete data log-likelihood:

$$\begin{aligned} \log P(q_{1:T}, \dot{q}_{1:T}, \ddot{q}_{1:T}, \tau_{1:T}, c_{1:T}) &= -\sum_{t=1}^{T} (\frac{1}{2} [\tau_t - g_t(c_t)]^T U_{obs}^{-1} [\tau_t - g_t(c_t)]) \\ & -\frac{T}{2} \log |U_{obs}| \\ & -\sum_{t=1}^{T} (\frac{1}{2} [c_{t+1} - c_t]^T U_{tr}^{-1} [c_{t+1} - c_t]) \\ & -\frac{T-1}{2} \log |U_{tr}| + \log(p(c_0)) \end{aligned} \tag{D.6}$$

The prior $p(c_0)$ will not be modeled neither. If we had many data sequences we could model it but it does not make sense to model it with only a single sequence. Instead,

we will sample the particles at the first time step from a uniform distribution. In the EM algorithm, we want to maximize the expected complete log-likelihood, which can be obtained by taking the expectation of (D.6) with respect to the smoothed posterior $p(c_t|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})$:

$$\begin{aligned}\mathcal{Q} &= -\sum_{t=1}^{T}\int(\frac{1}{2}[\tau_t-g_t(c_t)]^T U_{obs}^{-1}[\tau_t-g_t(c_t)])p(c_t|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})dc_t \\ &\quad -\frac{T}{2}\log|U_{obs}| \\ &\quad -\sum_{t=1}^{T}\int(\frac{1}{2}[c_{t+1}-c_t]^T U_{tr}^{-1}[c_{t+1}-c_t])p(c_t,c_{t+1}|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})dc_t dc_{t+1} \\ &\quad -\frac{T-1}{2}\log|U_{tr}| \end{aligned} \tag{D.7}$$

We first maximize (D.7) with respect to U_{obs}. The terms in (D.7) that contain U_{obs} are (after expanding):

$$\begin{aligned}Q_{V_{obs}} &= -\sum_{t=1}^{T}[\frac{1}{2}\tau_t^T U_{obs}^{-1}\tau_t-\frac{1}{2}\tau_t^T U_{obs}^{-1}\int g_t(c_t)p(c_t|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})dc_t \\ &\quad -\frac{1}{2}\int g_t(c_t)p(c_t|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})dc_t U_{obs}^{-1}\tau_t \\ &\quad +\frac{1}{2}\int g(q_t,\dot{q},\ddot{q}_t,c_t)^T U_{obs}^{-1}g_t(c_t)p(c_t|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})dc_t] \\ &\quad -\frac{T}{2}\log|U_{obs}| \end{aligned} \tag{D.8}$$

Since we approximate the posterior $p(c_t|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})$ with a set of N particles c_t^i with weights w_t^i, we replace the integrals with weighted sums as:

$$\begin{aligned}Q_{V_{obs}} &= -\sum_{t=1}^{T}[\frac{1}{2}\tau_t^T U_{obs}^{-1}\tau_t-\frac{1}{2}\tau_t^T U_{obs}^{-1}\sum_{i=1}^{N}w_t^i g_t(c_t^i) \\ &\quad -\frac{1}{2}\sum_{i=1}^{N}w_t^i g_t(c_t^i)U_{obs}^{-1}\tau_t+\frac{1}{2}\sum_{i=1}^{N}w_t^i g(q_t,\dot{q},\ddot{q}_t,c_t^i)^T U_{obs}^{-1}g_t(c_t^i)] \\ &\quad -\frac{T}{2}\log|U_{obs}| \end{aligned} \tag{D.9}$$

The second and third terms in the brackets are the same, so we can sum them to obtain:

$$\begin{aligned}Q_{V_{obs}} &= -\sum_{t=1}^{T}[\frac{1}{2}\tau_t^T U_{obs}^{-1}\tau_t-\tau_t^T U_{obs}^{-1}\sum_{i=1}^{N}w_t^i g_t(c_t^i) \\ &\quad +\frac{1}{2}\sum_{i=1}^{N}w_t^i g(q_t,\dot{q},\ddot{q}_t,c_t^i)^T U_{obs}^{-1}g_t(c_t^i)] \\ &\quad -\frac{T}{2}\log|U_{obs}| \end{aligned} \tag{D.10}$$

The derivative with respect to U_{obs} is:

$$\begin{aligned}\frac{dQ_{V_{obs}}}{dU_{obs}} &= -\sum_{t=1}^{T}[-\frac{1}{2}U_{obs}^{-1}\tau_t\tau_t^T U_{obs}^{-1}+U_{obs}^{-1}\tau_t\sum_{i=1}^{N}w_t^i g_t(c_t^i)^T U_{obs}^{-1} \\ &\quad -U_{obs}^{-1}\frac{1}{2}\sum_{i=1}^{N}w_t^i g(q_t,\dot{q},\ddot{q}_t,c_t^i)g_t(c_t^i)^T U_{obs}^{-1}] \\ &\quad -\frac{T}{2}U_{obs}^{-1} \end{aligned} \tag{D.11}$$

Given that U_{obs} is a symmetric matrix. Setting to zero and multiplying front and back by U_{obs} we obtain:

$$\begin{aligned}0 &= -\sum_{t=1}^{T}[-\frac{1}{2}\tau_t\tau_t^T+\tau_t\sum_{i=1}^{N}w_t^i g_t(c_t^i)^T \\ &\quad -\frac{1}{2}\sum_{i=1}^{N}w_t^i g(q_t,\dot{q},\ddot{q}_t,c_t^i)g_t(c_t^i)^T] \\ &\quad -\frac{T}{2}U_{obs} \end{aligned} \tag{D.12}$$

Solving for U_{obs} we obtain the maximization step for U_{obs}:

$$U_{obs}=\frac{1}{T}\sum_{t=1}^{T}[\tau_t\tau_t^T-2\tau_t\sum_{i=1}^{N}w_t^i g_t(c_t^i)^T+\sum_{i=1}^{N}w_t^i g(q_t,\dot{q},\ddot{q}_t,c_t^i)g_t(c_t^i)^T] \tag{D.13}$$

In practice, for computational reasons, we will collapse the set of particles to a single particle with unit weight. The maximization step for U_{obs}, then becomes:

$$U_{obs}=\frac{1}{T}\sum_{t=1}^{T}[\tau_t\tau_t^T-2\tau_t g_t(c_t^i)^T+g(q_t,\dot{q},\ddot{q}_t,c_t^i)g_t(c_t^i)^T] \tag{D.14}$$

We follow a similar procedure for maximizing (D.7) with respect to U_{tr}. After expanding, the terms that contain U_{tr} in ((D.7)) are:

$$\begin{aligned}Q_{U_{tr}} &= -\sum_{t=1}^{T}[\frac{1}{2}\int c_{t+1}^T U_{tr}^{-1}c_{t+1}p(c_{t+1}|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})dc_{t+1} \\ &\quad -\int c_{t+1}^T U_{tr}^{-1}c_t p(c_t,c_{t+1}|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_t)dc_t c_{t+1} \\ &\quad +\frac{1}{2}\int c_t^T U_{tr}^{-1}c_t p(c_t|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})dc_t] \\ &\quad -\frac{T-1}{2}\log|U_{tr}| \end{aligned} \tag{D.15}$$

Again, substituting the integrals with weighted sums (since we approximate the posteriors $p(c_t|q_{1:T},\dot{q}_{1:T},\ddot{q}_{1:T},\tau_{1:T})$ with samples), we obtain:

$$
\begin{aligned}
Q_{V_{tr}} &= -\sum_{t=1}^{T}[\frac{1}{2}\sum_{i=1}^{N} w_t^i {c_{t+1}^i}^T U_{tr}^{-1} c_{t+1}^i - \sum_{i=1}^{N}\sum_{j=1}^{N} w_t^i w_{t+1}^j {c_{t+1}^i}^T U_{tr}^{-1} c_t^j \\
&\quad + \frac{1}{2}\sum_{i=1}^{N} w_t^i {c_t^i}^T U_{tr}^{-1} c_t^i] \\
&\quad - \frac{T-1}{2}\log|U_{tr}| \qquad \text{(D.16)}
\end{aligned}
$$

Taking the derivative with respect to U_{tr} we obtain:

$$
\begin{aligned}
\frac{dQ_{V_{tr}}}{dU_{tr}} &= -\sum_{t=1}^{T}[-\frac{1}{2}U_{tr}^{-1}\sum_{i=1}^{N} w_t^i c_{t+1}^i c_{t+1}^i U_{tr}^{-1} + U_{tr}^{-1}\sum_{i=1}^{N}\sum_{j=1}^{N} w_t^i w_{t+1}^j c_{t+1}^i {c_t^i}^T U_{tr}^{-1} \\
&\quad - U_{tr}^{-1}\frac{1}{2}\sum_{i=1}^{N} w_t^i c_t^i c_t^i U_{tr}^{-1}] \\
&\quad + \frac{T-1}{2}U_{tr}^{-1} \qquad \text{(D.17)}
\end{aligned}
$$

Multiplying from the front and back by U_{tr}, setting to zero and solving for U_{tr} we get the update equation for U_{tr}.

$$
U_{tr} = \frac{1}{T-1}\sum_{t-1}^{T}[\sum_{i=1}^{N} w_{t+1}^i c_{t+1}^i c_{t+1}^i - 2\sum_{i=1}^{N}\sum_{j=1}^{N} w_{t+1}^i w_t^j c_{t+1}^i c_t^j + \sum_{i=1}^{N} w_t^i c_t^i {c_t^i}^T] \qquad \text{(D.18)}
$$

Approaching the filtered estimates with a single particle with unit weight we get the update:

$$
U_{tr} = \frac{1}{T-1}\sum_{t-1}^{T}[c_{t+1}^i c_{t+1}^i - 2c_{t+1}^i c_t^j + c_t^i {c_t^i}^T] \qquad \text{(D.19)}
$$

Regarding training of the model g. The respective terms of the expected complete log-likelihood are:

$$
\begin{aligned}
Q_g &= -\sum_{t=1}^{T} - \int \frac{1}{2}\tau_t^T U_{obs}^{-1} g_t(c_t) p(c_t|q_{1:T}, \dot{q}_{1:T}, \ddot{q}_{1:T}, \tau_{1:T}) dc_t \\
&\quad - \int \frac{1}{2} g_t(c_t)^T U_{obs}^{-1} \tau_t p(c_t|q_{1:T}, \dot{q}_{1:T}, \ddot{q}_{1:T}, \tau_{1:T}) dc_t \\
&\quad + \int \frac{1}{2} g_t(c_t)^T U_{obs}^{-1} g_t(c_t) p(c_t|q_{1:T}, \dot{q}_{1:T}, \ddot{q}_{1:T}, \tau_{1:T}) dc_t \qquad \text{(D.20)}
\end{aligned}
$$

Replacing the integrals with weighted sums and substituting for particles we get:

$$
Q_g = \sum_{t=1}^{T}\sum_{i=1}^{N} w_t^i \tau_t^T U_{obs}^{-1} g_t(c_t^i) - \sum_{i=1}^{N}\frac{1}{2} w_t^i g_t(c_t^i)^T U_{obs}^{-1} g_t(c_t^i) \qquad \text{(D.21)}
$$

If g took a parametric form, we could differentiate with respect to its parameters. However, LWPR is a nonparametric algorithm and thus we cannot differentiate with respect

to any parameters. Nevertheless, we will treat g itself as a variable. Differentiating with respect to g we get:

$$\frac{dQ_g}{dg} = \sum_{t=1}^{T}\sum_{i=1}^{N} w_t^i U_{obs}^{-1}\tau_t - \sum_{i=1}^{N} w_t^i U_{obs}^{-1} g_t(c_t^i) \tag{D.22}$$

Setting this to zero and multiplying by U_{obs} we get:

$$0 = \sum_{t=1}^{T}\sum_{i=1}^{N} w_t^i(\tau_t - g_t(c_t^i)) \tag{D.23}$$

Collapsing the filtered distribution to a single particle with unit weight we obtain:

$$0 = \sum_{t=1}^{T} \tau_t - g_t(c_t^i) \tag{D.24}$$

This implies training the LWPR model with the particle with the largest weight.

Bibliography

Antonelli, G., Caccavale, F., and Chiacchio, P. (1999). A systematic procedure for the identification of dynamic parameters of robot manipulators. *Robotica*, 17:427–435.

Arulampalam, M., Maskell, S., Gordon, N., and Clapp, T. (Feb 2002). A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188.

Åström, K. J. and Wittenmark, B. (1994). *Adaptive Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

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):101–119.

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

Baker, W. L. and Farrell, J. A. (1992). An introduction to connectionist learning control systems. In White, D. A. and Solfge, D. A., editors, *Handbook of Intelligent Control*, pages 35–63. Van Nostrand Reinhold, New York.

Bilmes, J. A. (1997). A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical report, University of Berkeley, ICSI.

Bishop, C. M. (1994). Mixture density networks. Technical Report NCRG/94/004, Neural Computing Research Group, Aston University.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Bonilla, E. V., Chai, K. M. A., and Williams, C. K. I. (2008). Multi-task Gaussian process prediction. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*, Cambridge, MA. MIT Press.

Briegel, T. and Tresp, V. (1999). Fisher scoring and a mixture of modes approach for approximate inference and learning in nonlinear state space models. In *Advances in Neural Information Processing Systems 11*, pages 403–409.

Cacciatore, T. W. and Nowlan, S. J. (1994). Mixtures of controllers for jump linear and non-linear plants. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 719–726. Morgan Kaufmann Publishers, Inc.

Chai, K. M. (2008). Personal communication.

Chai, K. M., Klanke, S., Williams, C., and Vijayakumar, S. (2008). Multi-task Gaussian process learning of robot inverse dynamics. In *Advances in Neural Information Processing Systems 21*.

Ciliz, M. K. and Narendra, K. S. (1994). Multiple model based adaptive control of robotic manipulators. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, volume 2, pages 1305–1310.

Ciliz, M. K. and Narendra, K. S. (1996). Adaptive control of robotic manipulators using multiple models and switching. *International Journal of Robotics Research*, 15(6):592–610.

Corke, P. (1996). A robotics toolbox for matlab. *IEEE Robotics and Automation Magazine*, 3(1):24–32.

Craig, J. J. (2005). *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall.

da Silva, B. C., Basso, E. W., Bazzan, A. L. C., and Engel, P. M. (2006a). Dealing with non-stationary environments using context detection. In *ICML '06: Proceedings of the 23rd international conference on machine learning*, pages 217–224, New York, NY, USA. ACM.

da Silva, B. C., Basso, E. W., Perotto, F. S., Bazzan, A. L. C., and Engel, P. M. (2006b). Improving reinforcement learning with context detection. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 810–812, New York, NY, USA. ACM.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

Doucet, A., De Freitas, N., and Gordon, N., editors (2001). *Sequential Monte Carlo methods in practice*. Springer.

Doya, K., Samejima, K., Katagiri, K., and Kawato, M. (2002). Multiple model-based reinforcement learning. *Neural Computation*, 14(6):1347–1369.

Dumont, G. and Huzmezan, M. (2002). Concepts, methods and techniques in adaptive control. *Proceedings of the 2002 American Control Conference.*, 2:1137–1150.

Dutkiewicz, P., Kozlowski, K., and Wroblewski, W. (1993a). Experimental identification of load parameters. In *Proceedings of the 1993 IEEE International Symposium on Industrial Electronics*, pages 361–366.

Dutkiewicz, P., Kozlowski, K., and Wroblewski, W. (1993b). Experimental identification of robot and load dynamic parameters. *Proceedings of the IEEE Conference on Control Applications*, pages 767–776.

Einbeck, J. and Stutz, G. (2006a). The fitting of multifunctions: an approach to nonparametric multimodal regression. In A. Rizzi, M. V., editor, *COMPSTAT 2006, Proceedings in Computational Statistics*, pages 1243–1250.

Einbeck, J. and Stutz, G. (2006b). Modelling beyond regression functions: an application of multimodal regression to speed-flow data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 55(4):461475.

Evgeniou, T., Micchelli, C. A., and Pontil, M. (2005). Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6:615–637.

Flanagan, R., Nakano, E., Imamizu, H., Osu, R., Yoshioka, T., and Kawato, M. (1999). Composition and decomposition of internal models in motor learning under altered kinematic and dynamic environments. *J. Neurosci.*, 19(20):34RC.

Franklin, G. F., Emami-Naeini, A., and Powell, J. D. (1993). *Feedback Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gautier, M. (1990). Numerical calculation of the base inertial parameters of robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1020–1025. IEEE.

Gelman, A., J.B., C., Stern, H., and Rubin, D. (1995). *Bayesian data analysis*. Chapman and Hall, London.

Ghahramani, Z. (1993). Solving inverse problems using an EM approach to density estimation. In Mozer, M. C., Smolensky, P., Touretzky, D. S., Elman, J. L., and Weigend, A. S., editors, *Proc. 1993 Connectionist Models Summer School*, pages 316–323. Erlbaum Associates.

Ghahramani, Z. and Hinton, G. E. (1996). Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, Department of Computer Science, University of Toronto.

Ghahramani, Z. and Jordan, M. I. (1994). Supervised learning from incomplete data via an EM approach. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 120–127. Morgan Kaufmann Publishers, Inc.

Ghahramani, Z. and Roweis, S. (1999). Learning nonlinear dynamical systems using an EM algorithm. In *Advances in Neural Information Processing Systems 11*, pages 431–437.

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

Harries, M. B. and Horn, K. (1996). Learning stable concepts in a changing world. In *PRICAI Workshops*, pages 106–122.

Harries, M. B., Sammut, C., and Horn, K. (1998). Extracting hidden context. *Machine Learning*, 32(2):101–126.

Haruno, M., Wolpert, D. M., and Kawato, M. (1998). Multiple paired forward-inverse models for human motor learning and control. In *Advances in Neural Information Processing Systems 11*, pages 31–37.

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

Hjorth, L.U.; Nabney, I. (1999). Regularisation of mixture density networks. In *Ninth International Conference on Artificial Neural Networks, 1999*, volume 2, pages 521–526.

Hoffmann, H., Petkos, G., Bitzer, S., and Vijayakumar, S. (2007). Sensor-assisted adaptive motor control under continuously varying context. In Zaytoon, J., Ferrier, J.-L., Andrade-Cetto, J., and Filipe, J., editors, *ICINCO-ICSO*, pages 262–269. INSTICC Press.

Hunt, K. J., Sbarbaro, D., Żbikowski, R., and Gawthrop, P. J. (1992). Neural networks for control systems: a survey. *Automatica*, 28(6):1083–1112.

Huo, W. (1995). New formulas for complete determining base parameters of robots. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 3, pages 3021–3026.

Imamizu, H., Kuroda, T., Miyauchi, S., Yoshioka, T., and Kawato, M. (2003). Modular organization of internal models of tools in the human cerebellum. *Proceedings of the National Academy of Sciences of the United States of America*, 100(9):5461–5466.

Imamizu, H., Kuroda, T., Yoshioka, T., and Kawato, M. (2004). Functional magnetic resonance imaging examination of two modular architectures for switching multiple internal models. *Journal of Neuroscience*, 24(5):1173–81.

Isard, M. and Blake, A. (1998). A smoothing filter for CONDENSATION. *Lecture Notes in Computer Science*, 1406:767–781.

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

Jordan, M. and Jacobs, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214.

Jordan, M. I. (1996). Computational aspects of motor control and motor learning. *Handbook of Perception and Action: Motor Skills*.

Jordan, M. I., Ghahramani, Z., Jaakkola, T., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233.

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

Jordan, M. I. and Wolpert, D. M. (1999). Computational motor control. *The Cognitive Neurosciences, 2nd edition*.

Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Karimi, A., Landau, I. D., and Motee, N. (2001). Effects of the design parameters of multimodel adaptive control on the performance of a flexible transmission system. *International Journal of Adaptive Control and Signal Processing*, 15:335–352.

Karniel, A. and Mussa-Ivaldi, F. (2000). Does the motor control system use multiple models and context switching to cope with a variable environment? *IEEE Transactions in automatic control*, 45:1669–1686.

Kawato, M., Furukawa, K., and Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185.

Kelly, R., Santibanez, V., and Loria, A. (2005). *Control of Robot Manipulators in Joint Space*. Springer.

Khalil, W. and Bennis, F. (1995). Symbolic calculation of the base inertial parameters of closed-loop robots. *International Journal of Robotics Research*, 14(2):112–128.

Kitagawa, G. (1996). Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5(1):1–25.

Kohlmorgen, J., Müller, K., and Pawelzik, K. (1994). Competing predictors segment and identify switching dynamics. In *Proceedings of the International Conference on Artificial Neural Networks 1994*, pages 1045–1048.

Kubat, M. (24-27 May 2004). Induction in time-varying domains: motivation, origins, and encouragements. In *Proceedings. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pages 316–321.

Kubat, M. and Widmer, G. (1995). Adapting to drift in continuous domains. In *Proceedings of the 8th European Conference on Machine Learning*, pages 307–322. Springer.

Leontaritis, I. J. and Billings, S. A. (1985). Input-output parametric models for non-linear systems part I: deterministic non-linear systems. *International Journal of Control*, 41(2):303–328.

Liehr, S., Pawelzik, K., Kohlmorgen, J., Lemm, S., and Müller, K. (1999). Hidden markov mixtures of experts for prediction of non-stationary dynamics. *Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 195–204.

Ljung, L. (1998). *System Identification: Theory for the User (2nd Edition)*. Prentice Hall PTR.

Mirizarandi, A.-R., Erfanian, A., and Kobravi, H.-R. (2005). Adaptive inverse control of the knee joint position in paraplegic subject using recurrent neural network. In *Proceedings of the 10th Annual Conference of the International FES Society*.

Müller, K., Kohlmorgen, J., and Pawelzik, K. (1995). Analysis of switching dynamics with competing neural networks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, pages 1306–1315.

Müller, K.-R., Mika, S., Ratsch, G., Tsuda, K., and Schölkopf, B. (Mar 2001). An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201.

Murata, N., Müller, K.-R., Ziehe, A., and Amari, S. (1996). Adaptive on-line learning in changing environments. In Mozer, M., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems*, pages 599–605. MIT Press.

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

Narendra, K. S. and Annaswamy, A. M. (1989). *Stable adaptive systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Narendra, K. S. and Balakrishnan, J. (1997). Adaptive control using multiple models. *IEEE Transactions in automatic control*, 42:171–187.

Narendra, K. S. and Xiang, C. (2000). Adaptive control of discrete-time systems using multiple models. *IEEE Transactions in automatic control*, 45:1669–1686.

Nguyen-Tuong, D., Peters, J., Seeger, M., and Schölkopf, B. (2008). Learning inverse dynamics: A comparison. In Verleysen, M., editor, *16th European Symposium on Artificial Neural Networks*, pages 13–18.

Ogata, K. (2001). *Modern Control Engineering (4th Edition)*. Prentice Hall.

Olsen, H. and Bekey, G. (1986). Identification of robot dynamics. *Proceedings of the IEEE International Conference on Robotics and Automation*, 3:1004–1010.

Petkos, G., Toussaint, M., and Vijayakumar, S. (2006). Learning multiple models of non-linear dynamics for control under varying contexts. In *Proceedings of International Conference on Artificial Neural Networks*, volume 1, pages 898–907.

Petkos, G. and Vijayakumar, S. (2007a). Context estimation and learning control through latent variable extraction: From discrete to continuous contexts. *Robotics and Automation, 2007 IEEE International Conference on*, pages 2117–2123.

Petkos, G. and Vijayakumar, S. (2007b). Load estimation and control using learned dynamics models. *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1527–1532.

Presse, C. and Gautier, M. (1993). New criteria of exciting trajectories for robot identification. In *ICRA (3)*, pages 907–912.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA.

Roweis, S. and Ghahramani, Z. (2001). *Kalman Filtering and Neural Networks*, chapter Learning Nonlinear Dynamical Systems using the EM Algorithm., pages 175–220. Wiley.

Schaal, S. and Schweighofer, N. (2005). Computational motor control in humans and robots. *Current Opinion in Neurobiology*, 15(6):675–82.

Sciavicco, L. and Siciliano, B. (2000). *Modelling and Control of Robot Manipulators*. Springer.

Shadmehr, R. and Wise, S. P. (2005). *The Computational Neurobiology of Reaching and Pointing*. MIT Press.

Slotine, J.-J. and Li, W. (1991). *Applied nonlinear control*. Prentice Hall.

Smith, R. (2006). Open dynamics engine v0.5 user guide.

Spong, M., Hutchinson, S., and Vidyasagar, M. (2006). *Robot modeling and control*. Wiley.

Stengel, R. (1994). *Optimal control and estimation*. Dover.

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

Swevers, J., Ganseman, C., Chenut, X., and Samin, J.-C. (2000). Experimental identification of robot dynamics for control. In *ICRA*, pages 241–246. IEEE.

Swevers, J., Ganseman, C., Tukel, D. B., de Schutter, J., and Van Brussel, H. (1997). Optimal robot excitation and identification. *IEEE Transactions on Robotics and Automation*, 13(5):730–740.

Swevers, J., Verdonck, W., Naumer, B., Pieters, S., and Biber, E. (2002). An experimental robot load identification method for industrial application. *The International Journal of Robotics Research*, 21:701–712.

Thrun, S. and Mitchell, T. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, 15:25–46.

Thrun, S. and Pratt, L., editors (1997). *Learning To Learn*. Kluwer Academic Publishers.

Todorov, E. (2006). *Bayesian brain*, chapter Optimal control theory, pages 269–298. MIT Press.

Valpola, H. and Karhunen, J. (2002). An unsupervised ensemble learning method for nonlinear dynamic state-space models. *Neural Computation*, 14(11):2647–2692.

Vetter, P. and Wolpert, D. M. (2000). Context estimation for sensorimotor control. *Journal of Neurophysiology*, 84:1026–1034.

Vijayakumar, S., D'Souza, A., and Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation*, 17:2602–2634.

Vijayakumar, S., D'Souza, A., Shibata, T., Conradt, J., and Schaal, S. (2002). Statistical learning for humanoid robots. *Autonomous Robots*, 12(1):55–69.

Vlassis, N. and Kröse, B. (1999). Mixture conditional density estimation with the EM algorithm. In *ICANN'99, 9th International Conference on Artificial Neural Networks*, pages 821–825. IEE.

Widmer, G. and Kubat, M. (1993). Effective learning in dynamic environments by explicit context tracking. In *Machine Learning: ECML-93, European Conference on Machine Learning, Proceedings*, pages 227–243. Springer-Verlag.

Wolpert, D. M. and Ghahramani, Z. (2000). Computational principles of movement neuroscience. *Nature Neuroscience*, 3:1212–1217.

Wolpert, D. M., Ghahramani, Z., and Flanagan, R. J. (2001). Perspectives and problems in motor learning. *Trends in Cognitive Sciences*, 5(11):487–494.

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

Xu, L., Jordan, M. I., and Hinton, G. E. (1995). An alternative model for mixtures of experts. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 633–640. The MIT Press.