ONLINE-LEARNING IN HUMANOID ROBOTS

Diploma Thesis

submitted to the Department of

Real Time Systems and Robotics

Institute of Computer Science

Technical University Berlin

by Jörg Conradt

Matrikel Number 180.002

Berlin, Winter Term 2000/01

TABLE OF CONTENTS

Foreword

The basis for this thesis was laid in the academic year 1999/2000, which I spent as an exchange student at the University of Southern California (USC), Los Angeles, and as a guest at the CLMC laboratory. The Computational Learning and Motor Control laboratory (CLMC laboratory), headed by Dr. Stefan Schaal, is pursuing research in the border areas of neuroscience, computer science, statistics, and robotics (control theory). The focus of the laboratory is to study the principles of biological behavior.

During the previous years, research has realized that even simple biological systems achieve by far superior sensory-motor competence compared to any artificial system today. On this basis, trying to understand and imitate these biological principles for artificial systems seems to be a promising approach. In this thesis, the motor control for a human-like robotic arm was learned based on a biologically plausible model.

During my time at the USC, I had the opportunity to participate in the research activities of the CLMC laboratory. I am very thankful for the discussions, for help, and for much interesting information that I experienced there. Especially Dr. Stefan Schaal and Dr. Sethu Vijayakumar have been patient and good-humored, and with their generosity in respect to equipment and other resources they have tremendously eased my start in the new environment. I sincerely hope that the time at the CLMC laboratory has generated a link that will last for much longer. The students belonging to the CLMC lab, Gaurav Tevatia and Biren Metha, also deserve my gratitude for discussing the first results and offering generous help on various topics.

I am also particularly grateful to the Fulbright Commission for providing funds to stay for one year at an American university. I could never have studied in the U.S. without their generous support.

Prof. Dr. Hommel from the TU Berlin was the person who first brought me into contact with the field of robotics. I especially wish to thank him for the encouragement and the careful supervision of this thesis. In addition, he kindly offered advice on all questions of research, by no means limited to the scope of this thesis.

Furthermore, I owe thanks to all the people who have read and commented on parts of earlier drafts and helped to improve the thesis.

1. Introduction

1.1 Introducing the Area of Humanoid Robotics

Humanoid Robotic Systems have gained an increasing significance in the research world within the last few years. Just five years ago, there were hardly any human-like robots in the world, and those available did not represent human properties at all. They neither looked nor behaved like human beings. Today, a variety of research groups around the world is starting to work on topics related to humanoid robots, and it is very likely that these robots will become important within the upcoming decades even beyond the realm of science.

Trying to determine what humanoid robots are, a first draft of a definition might read as follows: such robots are to be called humanoid robots which - to some extent - are able to live and interact with the everyday human world, and represent certain human features, like cognitive or acting abilities. The main strength of such humanoid robots lies in their ability to operate in surroundings that have been designed for humans in the first place. Humanoid robots can be imagined to become useful assistants for every-day life in areas as diverse as

- Rescue and clearing of dangerous situations
- Janitorial services, Housekeeping
- Security services
- Care-taking in hospitals, recreational facilities
- Entertainment

In all these fields, close human interaction is a core issue and can be regarded as the minimum common basis. The interaction happens on many different levels, from physical touch to gesture recognition and the processing of spoken language. On cognitive issues like the two last named, much research has been done in the past few years. One has, however, to keep in mind that also the physical appearance, e.g. smoothness of motions, is an important issue when designing humanoid robots.

1.2 Mechanical Design for Humanoid Robots

Given the close interaction with humans and the potential working spaces listed above, some core requirements for the design of humanoid robots can be set out:

First of all, humanoid robots need the ability to act in environments tailored for human needs and to operate devices originally designed for humans, e.g. when turning knobs. Therefore, they need the basic equipment, e.g. manipulator arms, and legs, which can perform independent tasks. Their movements have to be fast and accurate, and their grip fine and powerful. Their links have to be lightweight to reduce the influence of inertia. It is also desirable to have compliant joints, because only these will allow the robot to react flexibly to external stimuli. This might be the case when the robot is pushed aside during the performance of a task. A natural result of building robots according to these prerequisites is a high degree of redundancy in the whole system - an issue, which has to be handled appropriately.

Moreover, humanoid robots have to be equipped with sensory and processing capabilities to interact with their environment, and they must be mobile within a wide range. In general, their mechanical design has to ensure that they can operate in normal living environments without extensive modifications of these surroundings.

With these requirements, the mechanical design of humanoid robots differs substantially from that of today's robots, e.g. robots used for manufacturing. Such industrial robots are obviously designed according to very different needs: One of their main purposes is to repeat tasks with high accuracy. This requires stiff joints, solid links, and strong actuators. To allow simple control, they are designed to behave as linear as possible.

The amount of compliance in the joints is a particularly important issue for biologically plausible motion and shall now be set out in some more detail. As has been said above, compliance in the joints is not only important for the smoothness of motion. It is above all the central prerequisite for the robot's ability to react appropriately to external stimuli. Such stimuli occur frequently in natural environments and it seems to be one of the most remarkable abilities of living organisms to react to such changes and adapt their own behavior and actions. Such changes might be local impediments while a task is performed, or pushes. Assume shaking hands with the robot: If the robot consists of stiff joints, shaking hands cannot be performed well as one has to follow exactly the robot's desired motion. If the joints are compliant, much more variation in the movement is possible - a human partner can always force the robot to slightly adjust its own position.

On the mechanical level, motion in the joints can be achieved by different means. The two major approaches are electric motors combined with gearboxes, and hydraulic actuation. When using electric motors, gearboxes are a necessary complement, because only they provide sufficient torques for the robot's movements. Gearboxes, however, increase the stiffness in all joints by a considerable amount, since they do not offer back-drive ability. Therefore, motors with gearboxes are unsuitable as actuators for humanoid robots.

Alternatively, robots can be equipped with hydraulics to generate high torques for the joints. When using hydraulics together with load sensors in every joint, the joints' behaviors can be anywhere between very stiff and very compliant, only depending on the controller. This means, the problem of compliance is moved to the level of software.

The only problem that remains is that any increase of compliance goes along with a decrease of accuracy. It is therefore highly important to find a good tradeoff between accuracy of motion and compliance in the joints. The quality of a controlling method can be directly correlated with its ability to find such a balance.

1.3 Controlling Humanoid Robots

Let us assume that all the mechanical desires described in chapter 1.2 can be fulfilled, and that the problems with stiff joints and heavy material used in the links can be solved. There still remains a major problem: By what means could such a robot be controlled? What kind of algorithms allows the robot to use the whole variety of motion that is usually associated with biological motion? For Example, how could it be accomplished that a humanoid robot gives way for external motion, such as pressure enforced by contact with humans? And given the desired compliance is achieved using lightweight material as described in chapter 1.2: How can we cope with the constraints that such materials add on the control algorithms? It is obvious that traditional algorithms, e.g.

those based on rigid body dynamics assumptions, are not well suited to control such mechanics. A fairly novel way to solve the question of controlling the robot is the application of learning approaches. The major advantage of a learned control strategy is that it adapts the control schema based on how the system behaves. This means that a well-suited learning algorithm will always stay accurate.

1.4 Examples of Today's Humanoid Robots

In this chapter, I would like to provide a short overview of today's humanoid robots, and present some examples.

Probably the best-known robot was build by Honda during the last 10 years (e.g. Hirai, 1987, Hirai, Hirose, et al., 1998). Figure 1.1 shows a picture of the robot, which looks very much like a human.

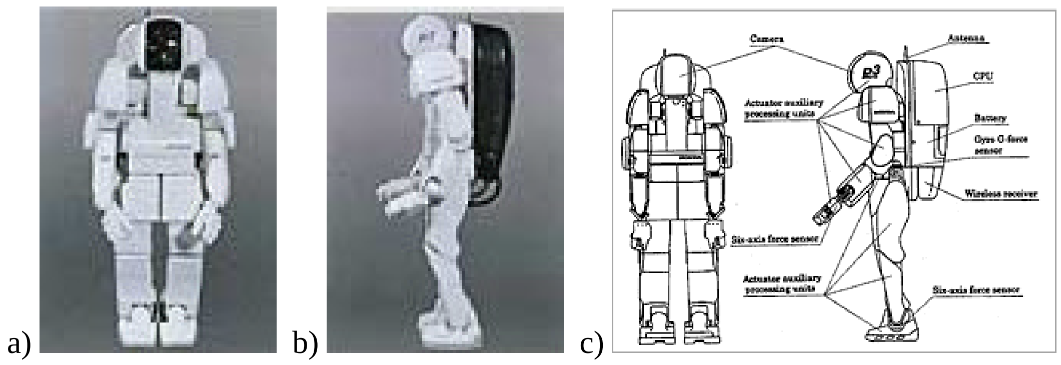

Figure 1.1 The Honda Robot P3: a) Frontal View b) Side View c) Technical Diagram

The main goal of Honda's development was to create an autonomous robot that can move about in a human environment. The robot can walk, climb stairs, and manipulate simple objects. Every singe step, however, needs to be carefully programmed in advance. As soon as more-complex tasks arise, the robot has to be switched to tele-operation mode and becomes controlled by a human supervisor. Corresponding to that, the robot is highly stiff in all joints and the sensing capabilities are very limited. The size of stair-steps it has to climb, for example, has to match exactly the preprogrammed values. Very little visual feedback or other sensor-information is used to correct for unexpected changes in the environment. This robot appears to be a human, but does not at all behave like a humanoid robot.

The robot’s technical parameters are summarized in the following table:

Weight:	130 kg
Height:	160cm
Width:	60cm
Depth:	55,5cm
Walking speed:	2.0 km / hour, ca. 25 minutes autonomous walking
Operated on DC servo motors with gears	
Lift Capacity per Hand: 2kg	
Degrees of Freedom: 28 (Legs: 2x6, Arms: 2x7, Hands: 2x1)	

Table 1.1 Technical Parameters of the Honda Humanoid Robot

Another example of a humanoid robot is COG, a robot torso developed at MIT. A picture of COG is shown in figure 1.2:

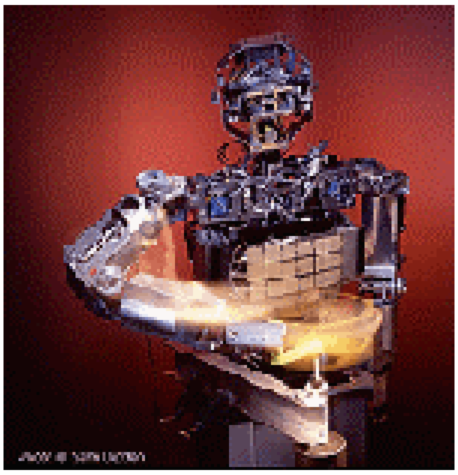
Figure 1.2 COG, the Robot Torso developed at MIT

COG is used to study the question how far a humanoid robot can become 'cognitive'; therefore, the focus of research is on the robot's interaction with people. The robot realizes what people want and acts properly, sometimes also following its own 'desires'. It is not designed for complicated motion. Though the robot's physical behavior is very much different from humans, experiments have shown that it can help to understand the way people interact with each other. In the near future, the research objective of the development team is to achieve a robot's perception that is comparable to that of a six-months-old baby.

The third example for a humanoid robot is DB (Dynamic Brain), a robot at the ATR human resources research laboratories in Kyoto, Japan. DB is a hydraulically actuated anthropomorphic robot with legs, arms (with hand palms but without fingers), a jointed torso, and a head. It was designed and built by Sarcos, a company that usually builds tele-operated robots for entertainment purposes like movies and amusement parks. The robot offers a variety of motions; however, it is mounted on a pelvis, so free standing or walking experiments cannot be performed. The research performed at the ATR laboratories focuses on upper-body movement (e.g. arm motion).

DB’s technical parameters are summarized in the following table:

Weight:	80 kg
Height:	185cm
Width:	60cm
Depth:	35cm
Operated on hydraulic actuators with 650 psi pumps	
25 linear actuators, 5 rotary actuators	
Degrees of Freedom: 30 (Legs: 2x3, Trunk: 3, Arms: 2x7, Neck: 3, Eyes: 2x2)	
Position and load sensors in every Degree of Freedom (except eyes)	
Video cameras providing stereo fovea and panoramic view	

Table 1.2 Technical Parameters of the DB robot

Comparing DB's parameters with those of the Honda robot, one can easily recognize that DB represents human properties much better, especially in terms of the weight and size. Moreover, being actuated by hydraulic pressure, the joints can be controlled in a compliant way, as discussed in chapter 1.2. Figure 1.3 shows the robot.

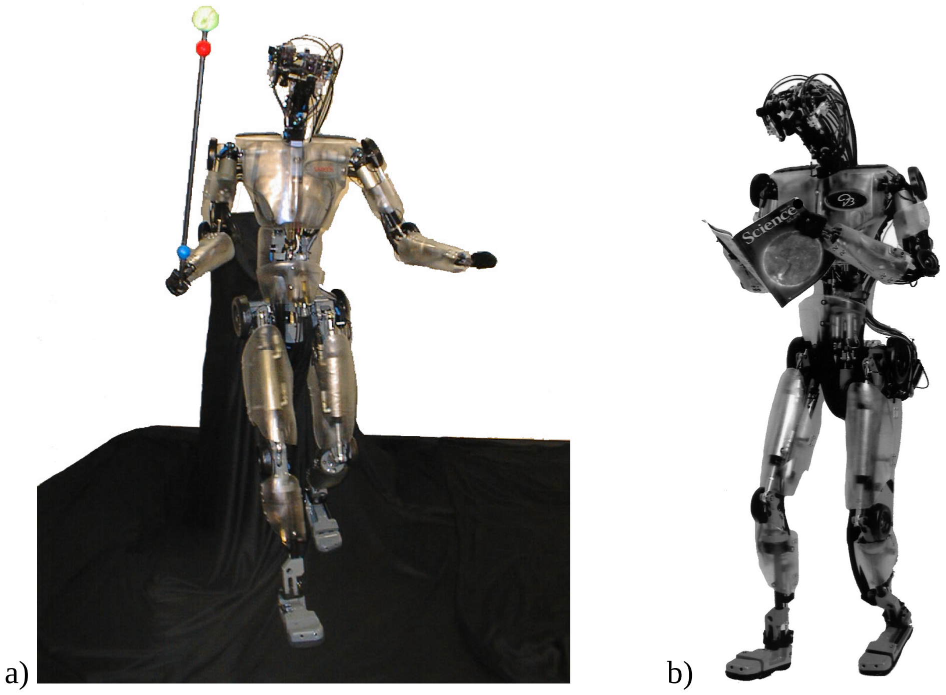

Figure 1.3 a) DB balancing a pole, b) DB reading 'Science'

Other laboratories, where humanoid robots are developed and much research is done are the University of Tokyo, the Waseda University and at the Vanderbilt University.

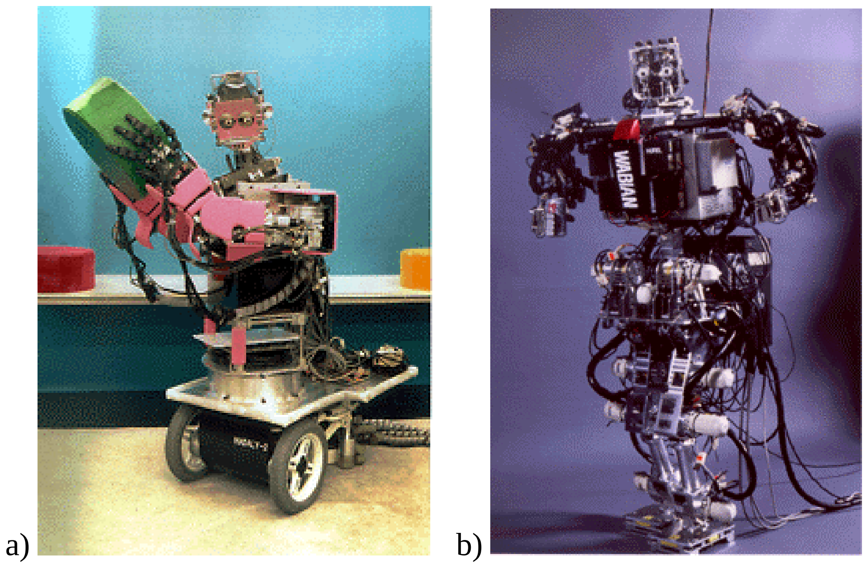

Figure 1.4 a) Hadaly and b) Wabian from Waseda University

1.5 The Purpose of the Thesis

The following study focuses on the control problem of compliant joints used in humanoid robots. As described in chapter 1.2, humanoid robots should be designed and operated with compliant joints. These joints allow biologically plausible motion, i.e. motion that is efficient, differentiated, energetically economic, and offers a wide variety of positions. In addition, it enables the robot to interact in an environment shared with humans. For example, it minimizes the risk of injuring people during interaction. Now the problem is that controlling compliant joints is extremely difficult when at the same time motion shall be accurate. Traditional linear high-gain control methods are not suitable for compliant control. And the model-based nonlinear control with rigid body dynamics models is often too inaccurate (see the further discussion in chapter 2.5).

My thesis offers a new approach to control the motion of humanoid robots. This method is based on neural net learning techniques, and leads to superior motion performance compared to traditional approaches. My test case is learning the inverse dynamics model of a seven degree-of-freedom robot arm. This anthropomorphic robot arm was built from lightweight materials to resemble human arm dynamics. It is hydraulically actuated and has load sensors in each joint to allow compliance.

The learned inverse dynamics model can be used in a computed torque feed-forward controller. In contrast to traditional feed-forward control, the new controller increases the accuracy of the arm motion to a significant degree without reducing its compliance. The results show that with the help of this new approach, natural motion of a human-like robot can be achieved with high accuracy.

My research aims to help closing a gap that opens in the robot control framework. Today, broad research efforts are spend on algorithms to decide what a robot does in a given situation. But if a robot decided what to do, it may not know how to execute the desired motion accurately. Motion execution using compliant joints becomes more relevant as robots start to interact with people.

2. A brief Recapitulation of basic Robot Control

2.1. Introduction to Robot Control

For moving a manipulator (e.g. a robot arm) from a discrete state to another desired discrete state, an appropriate control command (e.g. current for an electric motor) has to be generated all the time during the robot’s motion. This is achieved by updating the previous command at discrete time steps, equivalent to the reciprocal value of the control loop’s frequency: $\Delta \mathrm{T} = {}^{1}\!/_{\mathrm{f}}$. The control loop usually runs at a high frequency to allow fast command update and accurate motion.

For generating commands to achieve desired robot behavior, one has to distinguish three different phases: 1st planning a trajectory in end-effector space, 2nd translating the desired trajectory into joint space,[1] and 3rd executing the planned trajectory. Concerning the first phase, let us assume throughout this thesis that a desired trajectory in end-effector space exists. E.g., a desired behavior can be relatively simple to plan like a point-to-point reaching. This can easily be planned using a direct line as desired line-of-motion and adding constraints, such as bell shaped velocity and acceleration profiles. When these constraints are known, it is possible to estimate all coefficients of an nth order polynomial that describes the desired fingertip position between start and end position. Typically, third or fifth order polynomials are used for planning. They allow 4 or 6 constraints in total. These constraints usually are given by the start- and end-position, and the start and end-velocity. The accelerations can be added if 6 unknowns are used. A path in end-effector space given by a set of via-points can be decomposed into several short paths from point p_n to point p_{n+1}.

The second part of robot control, transferring the desired trajectory from end-effector space into joint space, is called the inverse kinematics problem. To solve this problem, a coordinate transformation from Cartesian space into joint space is needed. If we define the intrinsic coordinates of a manipulator as the n-dimensional vector $\textbf{è} \in \Re^n$, and the position and orientation of the manipulator's end-effector as the m-dimensional

[1] It is also possible to directly plan a trajectory in actuator space, without the need to translate it. However, this is hardly done in practice, as the joint space of a reasonable sized robot is huge and not simple to understand for humans.

vector $\mathbf{x} \in \Re^m$, the kinematics function can be written as $\mathbf{x} = f(\mathbf{è})$. What we need here is the inverse relationship: $\mathbf{è} = f^{-1}(\mathbf{x})$.

There are two general approaches to solving inverse kinematics problems with optimization criteria: 1. One can use global methods to find an optimal path of   with respect to the entire trajectory, or use 2. local methods, which only compute an optimal change in $\mathbf{è}$, $\Delta\mathbf{è}$, for a small change in $\mathbf{x}$, $\Delta\mathbf{x}$. In this case, one would have to integrate $\Delta\mathbf{è}$ to generate the entire joint space trajectory. An example of a local method is Resolved Motion Rate Control (e.g. Whitney, 1969). It uses the Jacobian **J** of the forward kinematics to describe a change of the end-effector's position by $\dot{\mathbf{x}} = \mathbf{J}(\mathbf{è})\,\dot{\mathbf{è}}$. This equation can be solved for $\dot{}$ by taking the inverse of **J**, if it is square, i.e. $\mathrm{n} = \mathrm{m}$, and non-singular. For redundant manipulators (hence for almost all robots), solutions to the inverse equation are usually non-unique (Craig, J. 1986), so that additional optimization criteria have to be introduced. Alternatively, other methods can be used to invert **J**, like the pseudo-inverse method (e.g. Liegeois, 1977), or the Extended Jacobian Method (Baillieul, J., 1985). There exists a variety of literature on inverse kinematics.[2]

The robot used for research at the CLMC-lab consists of seven joints, hence 7 degrees of freedom (DOFs). Our motion algorithms usually calculate a single desired end-effector position in Cartesian space (with no particular constraints on the orientation of the fingertip). This explains our need for an inverse dynamics mapping from $\Re^3 \to \Re^7$ (which must be ill-defined, as it maps from lower into higher space). My thesis, however, does not concentrate on the inverse kinematics problem. It will only give an idea of how the learning algorithm ***LWPR*** can be applied for solving inverse kinematics in the conclusion in chapter 8.2.

The remaining problem, executing the desired trajectory in joint space, will be described in more detail in the following section. One of its subcomponents, the Inverse Dynamics Problem, with a learning approach to solve this problem, will be discussed carefully in the thesis. Therefore, let us assume that a planned desired trajectory in joint space is available at any time.

---

[2] E.g. Schwinn, W., 1992, Rieseler, H., 1992, Kovács, P., 1993

2.2. Controlling the Execution of Desired Trajectories

The following chapter presents a simple controlling method for joint position's during motion. Additional components are added step by step to increase the accuracy:

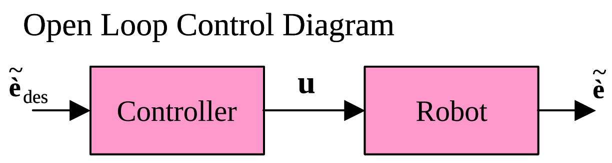

Figure 2.1 Open Loop Control

$$\mathbf{u} = f(\tilde{\mathbf{è}}_{des}, \mathbf{á}, t) \qquad \text{(eq. 2.1)}$$

As can be seen in figure 2.1 and in the according equation, this is a very simple model, which controls the robot ‘in a blind way’. The controller generates commands (**u**) using a function $f(\circ)$ based on a desired state ($\tilde{\mathbf{è}}_{des}$),[3] constant parameters of the system (   ) and the time (t). The robot changes its state to a new state ($\tilde{\mathbf{è}}$) based on the control command it receives. The controller sends a command to the system without monitoring how the system responds. There is no error correction because the controller does not know what the system really does. To solve this problem, we can include a feedback mechanism:

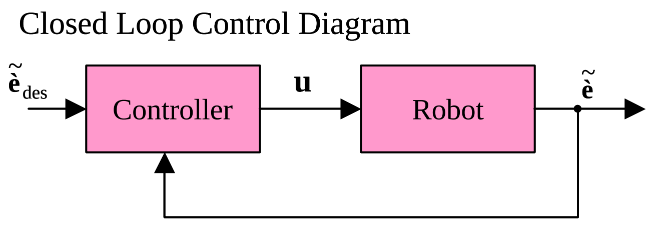

Figure 2.2 Closed Loop Control

$$\mathbf{u} = f(\tilde{\mathbf{è}}, \tilde{\mathbf{è}}_{des}, \mathbf{á}, \mathrm{t}) \qquad \text{(eq. 2.2)}$$

[3] Throughout this chapter, $\tilde{\mathbf{è}}$ denotes the state of a system, given by its position, velocity, and acceleration: $\tilde{\mathbf{è}} = (\mathbf{è}, \dot{\mathbf{è}}, \ddot{\mathbf{è}})$.

Figure 2.2 illustrates that the controller is now enabled to correct errors, because it knows about the state of the system and can verify, whether a previously sent command has moved the robot into the expected state. If the result was not satisfactory, the controller can change the subsequent command $\mathbf{u}^{+1}$ appropriately.

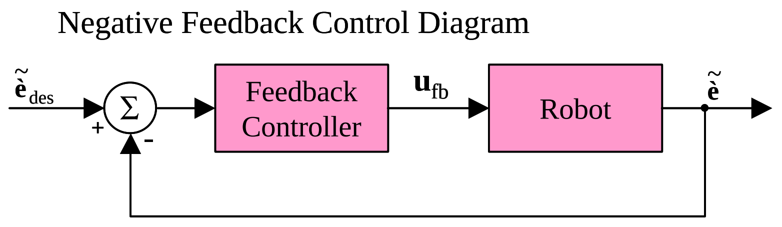

Figure 2.3 Negative Feedback Control

$$\mathbf{u} = f_{fb}(\tilde{\mathbf{è}}_{des} - \tilde{\mathbf{è}}, \mathbf{á}, t) \qquad \text{(eq. 2.3)}$$

A special case of closed loop control is negative feedback control, illustrated in figure 2.3. The controller does not know the desired state of the robot, but only the difference between the desired and the real state ($\tilde{\mathbf{è}}_{des} - \tilde{\mathbf{è}}$). This difference can be interpreted as an error signal of the robot’s state. Generating appropriate commands ($\mathbf{u}_{fb}$) to minimize the input (and thus the error signal) will lead to a control solution. The negative feedback approach will become infinitesimal accurately when using proper command generation functions $f_{fb}(\circ)$. The major disadvantage of the negative feedback control is the time delay, which causes a significant loss of accuracy in the robot's movements. This control strategy is further explained in chapter 2.3. in the context of PID-controllers. To further minimize the error, we can add a feed-forward controller:

Negative Feedback and Feedforward Control Diagram

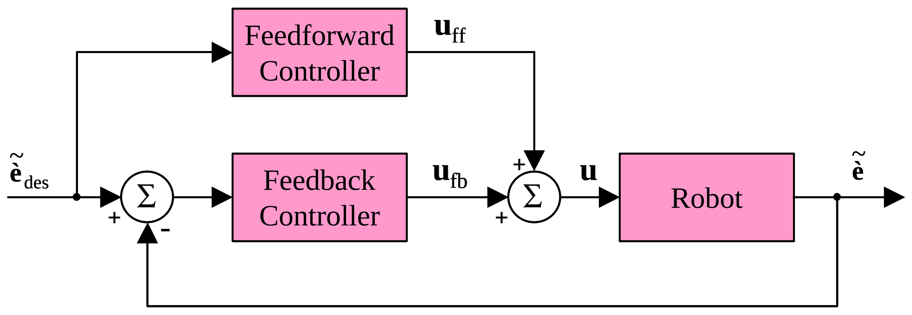

Figure 2.4 Negative Feedback and Feed-forward Control

$$\mathbf{u} = f_{fb}(\tilde{\mathbf{\grave{e}}}_{des} - \tilde{\mathbf{\grave{e}}}, \mathbf{\acute{a}}, t) + f_{ff}(\tilde{\mathbf{\grave{e}}}_{des}, \mathbf{\acute{a}}, t) \qquad \text{(eq. 2.4)}$$

In figure 2.4, a computed-torque feed-forward controller is introduced to reduce the error and thus improve the robot's performance. The feed-forward controller uses a dynamics model of the robot. It is now able to predict the actuator commands, which correspond to a desired motion. Intuitively, the feed-forward controller moves the robot towards the desired state in big steps and as accurately as possible. The dynamics model used in $f_{ff}(\circ)$ to compute the feed-forward commands ($\mathbf{u}_{ff}$) will never be absolutely accurate: It leaves a remaining error to be corrected by the feedback controller. The feed-forward controller operates as a set-top-box, which generates commands ($\mathbf{u}_{ff}$) independently of the negative feedback controller. The final command sent to the robot thus is the addition of both commands, feed-forward and negative feedback: $\mathbf{u} = \mathbf{u}_{fb} + \mathbf{u}_{ff}$.

We will have a closer look at both, the negative feedback and the feed-forward control functions, in the following two chapters.

2.3. The Feedback Control Function

PID-controllers are widely used as control policy in negative feedback controllers. These controllers correct errors in position and velocity. They are usually based on a linear control function. The name PID-controller is derived from:

- **P**roportional Control ('Position Error')
- **I**ntegral Control ('Steady State Error')
- **D**erivative Control ('Damping')

The Control Function f_{fb} of a PID-Controller is given by

$$\mathbf{u}_{\text{fb}} = \mathbf{k}_{\mathbf{P}} \cdot (\grave{\mathbf{e}}_{\text{des}} - \grave{\mathbf{e}}) + \mathbf{k}_{\mathbf{D}} \cdot (\dot{\grave{\mathbf{e}}}_{\text{des}} - \dot{\grave{\mathbf{e}}}) + \mathbf{k}_{\mathbf{I}} \cdot \int (\grave{\mathbf{e}}_{\text{des}} - \grave{\mathbf{e}})\, dt \qquad \text{(eq. 2.5)}$$

with

$\mathbf{u}_{\text{fb}}$:	the (n×1) vector of computed neg.-feedback commands
$\mathbf{k}_{\text{P}}$, $\mathbf{k}_{\text{D}}$, $\mathbf{k}_{\text{I}}$:	the (n×1) gain vectors for P-, D-, and I–control
$\grave{\mathbf{e}}_{\text{des}} - \grave{\mathbf{e}}$:	the (n×1) vector of errors in positions
$\dot{\ }_{\text{des}} - \dot{\ }$:	the (n×1) vector of errors in velocities

The term $(\grave{\mathbf{e}}_{\text{des}} - \grave{\mathbf{e}})$ in equation 2.5 will become non-zero, whenever the robot is in a position $(\grave{\mathbf{e}})$ that differs from the desired position $(\grave{\mathbf{e}}_{\text{des}})$. This position error $(\grave{\mathbf{e}}_{\text{des}} - \grave{\mathbf{e}})$ multiplied by a fixed gain ($\mathbf{k}_{\text{P}}$) yields in an appropriate command to compensate for that error, given that the gain is properly adjusted: Too small gains will cause the system to only slowly correct for errors; too high gains may lead to overcompensation (and ultimately to unstable systems).

The second term, $(\dot{\grave{\mathbf{e}}}_{\text{des}} - \dot{\grave{\mathbf{e}}})$ in equation 2.5, has the same effect on the velocities: whenever a velocity differs from the desired velocity, a correction command will be calculated by the difference multiplied with a gain $\mathbf{k}_{\text{D}}$. This part of the feedback controller introduces a damping term.

The integral part additionally corrects very small errors once a steady state is reached. Those errors might be introduced by external forces, e.g. gravity. To understand the term, assume the robot has reached its target, but gravity makes it drop slightly under the desired position. The first term $(\grave{\mathbf{e}}_{\text{des}} - \grave{\mathbf{e}})$ in equation 2.1 will provide compensation, but only proportional to the distance-error. If this error is small, or the arm is heavy, the generated command may not succeed in moving the arm upwards. Ultimately, there will be an equilibrium state below the desired target, when the correction of the PD-controller compensates for gravity. The arm will not move upwards to the desired target anymore.

The integral controller will trace and accumulate the error. The integral will only stop adding to the accumulation, when $(\mathbf{è}_{des} - \mathbf{è})$ is zero, i.e. when the robot is exactly at the target position. This means, it will continue until the accumulation can achieve a compensation for the position offset (considering the gain $\mathbf{k}_I$). In case the robot overshoots, the accumulation will decrease, as $(\mathbf{è}_{des} - \mathbf{è})$ changes signs. Choosing an appropriate gain $\mathbf{k}_I$, the correction introduced by the integral part of equation 2.5 will keep the robot exactly at the desired position.

Finding a gain $\mathbf{k}_I$ is probably the most difficult task in this process. A poor choice can easily lead to an unstable system with catastrophic results. Integral controllers compensate for steady states only, but this study concentrates on robot arms in motion. Therefore, integral controllers have not been used as part of the negative feedback control. All feedback controllers were modeled by a proportional and a derivative part only.

Looking for appropriate gains $\mathbf{k}_P$ and $\mathbf{k}_D$ to design a PD-controller as shown above, one faces a tradeoff between stiff and accurate systems:

- Assume high gains:[4] Whenever the desired state differs from the robot's state, the command sent for compensation is relatively big. This will ensure fast compensation. However, if you need the robot to give in, e.g. due to a push or when hitting an object, a high gain controller will exert a high force to compensate for this ‘position-error’.
- Assume low gains: The position error is multiplied by a smaller number. Thus, the command send to the robot to correct the error is significantly smaller. Hence, the robot is much more flexible when it hits an object or is being pushed aside. On the other hand, it also needs longer to correct for ‘real’ position errors.

Especially in the context of Humanoid Robotics, this tradeoff is very difficult. Humanoid robots are supposed to work in close human interaction and may not hurt anyone in interacting; but at the same time, they need to be capable of fast and accurate motion. The

[4] High gains can also result in instable systems, as the robot may increase the absolute distance to the desired target with every discrete step. Let us assume that the gains do not exceed the maximal value that guarantees safe operation.

approach usually taken today is to use low gain feedback controllers and add a feed-forward path to enhance performance. I will explain this mechanism in more detail in the following chapters.

2.4. The Feed-forward Control Function

As seen in chapter 2.2, the feed-forward control function uses the dynamics of a system to generate feed-forward commands. Then, these commands are used to move the system quickly along a desired trajectory:

$$\mathbf{u}_{\mathrm{ff}} = f_{\mathrm{ff}}(\tilde{\mathbf{\grave{e}}}_{\mathrm{des}}, \mathbf{\acute{a}}, \mathrm{t})\,. \qquad \text{(eq. 2.7)}$$

The dynamics of the system is hidden in the parameter vector $\mathbf{\acute{a}}$.

In general, the dynamics model provides a description of the relationship between the joint actuator torques and the motion on the structure. It relates the vectors of positions ( ), velocities ($\dot{\ }$), and accelerations ($\ddot{\ }$) to the torque vector ( ). This last vector is necessary to accomplish the acceleration in order to reach the desired state.

The direct dynamics problem assumes a given initial state of the system (position, and velocity) and an incoming stream of joint torques. It will then predict the acceleration in all joints for times $\mathrm{t} > \mathrm{t}_0$. Updates of velocities and positions can be calculated using integration techniques over the accelerations. The direct dynamics is a useful model for simulations, as it allows predictions of the physical system's motion, when the exerted joint torques are known.

In contrast, the inverse dynamics model predicts the joint torques needed to generate a specified motion. Solving the inverse dynamics problem will allow the execution of a desired trajectory, once the trajectory is specified in terms of positions, velocities, and accelerations in joint space. Usually, this is a result of an inverse kinematics process (see chapter 2.1). Simulating the trajectory before executing the motion can be used to verify the trajectory's feasibility: e.g., torques may not exceed a maximal value, nor may they change abruptly.

Using the inverse dynamics model for feed-forward command predictions leads to improved performance of the controller, compared to a ‘simple’ feedback controller. This combined controller will always move the robot close to the correct position using feed-forward commands. Then, the ‘slowly adjusting' feedback part only has to compensate for small position errors. Hence, the overall performance will increase. If we had a perfect robot and knew the dynamics model, no feedback control was necessary at all. But as our robot is not perfect, we do need to find the inverse dynamics model. Thus, the question of how to estimate the dynamics model remains.

2.5. Estimating Dynamics using Rigid Body Assumptions

The Rigid Body Dynamics assumptions[5] are frequently used to estimate the dynamics of a robot system[6]. The assumptions include, that the system to be modeled consists of single stiff bodies. These rigid bodies are assumed to behave like perfect 'single link robots', being connected to form a robot with multiple degrees of freedom. In addition, the link’s motion is not supposed to have influence on any other of the links and joints. Only the link's weight and its position are modeled. The joints connecting two links are not allowed to have friction or position inaccuracies. The dynamics of the system is therefore described by the shape and weight of the links, not by their joint behavior.
The general structure of rigid body dynamics, also called the 'Joint space dynamic model', can be estimated in two different ways:[7]

The Lagrange formulation offers the system's dynamics in a closed form based on the Lagrangian of the systems’ total energy.

The Newton-Euler formulation allows describing the model in a recursive form by forcing a torque balance on every link. This is computationally more efficient. However, parameter estimation for unknown systems is more difficult compared to a closed form model. For this reason, we will have a closer look at the Lagrangian formula of a mechanical system. It is defined as:

[5] E.g. Sciavicco & Siciliano, 2000 or An, Atkeson & Hollerbach, 1988
[6] E.g. Sciavicco & Siciliano, 2000 or Pfeiffer & Reithmeier, 1987

$$\mathcal{L} = \mathcal{T} \quad \mathcal{V}, \qquad \text{(eq. 2.8)}$$

where $\mathcal{T}$ and $\mathcal{V}$ denote the total kinetic energy and, respectively, the total potential energy of the system. The Lagrange's Equation is expressed by

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\text{è}}_{\text{i}}} - \frac{\partial \mathcal{L}}{\partial \text{è}_{\text{i}}} = x_{\text{i}}, \qquad \text{i} = 1,\dots,\text{n} \qquad \text{(eq. 2.9)}$$

where n denotes the number of joints in the robot and x_{i} the generalized force associated with the coordinate è_{i}. The forces include all non-conservative forces, such as joint actuator torques, the joint friction torques, and the joint torques induced by end-effector forces at the contact with the environment. As we have seen earlier, equation 2.5 establishes the relation between generalized forces (i.e. the joint torques) and joint positions, velocities and acceleration. Consequently, it is possible to derive the dynamic model, once the kinetic and potential energy of all links are known. For details of the computation of these energies, please refer to e.g. Sciavicco & Siciliano, 2000. The general formula of the equation of motion derived by the Lagrangian method is

$$\mathbf{B}(\mathbf{è}) \cdot \ddot{\mathbf{è}} + \mathbf{C}(\mathbf{è}, \dot{\mathbf{è}}) \cdot \dot{\mathbf{è}} + \mathbf{G}(\mathbf{è}) = \mathbf{ô} \qquad \text{(eq. 2.10)}$$

with:

$\mathbf{B}_{(\text{n} \times \text{n})}$: a positive definite inertia matrix,
only depending on the robot's current position

$\mathbf{C}_{(\text{n} \times \text{n})}$: a matrix containing the centripetal and Coriolis forces,
depending on the robot's current position and its current velocities

$\mathbf{G}_{(\text{n} \times 1)}$: a vector of gravitational forces,
only depending on the robot's current position

$_{(\text{n} \times 1)}$: a vector of the active torques at the joints 1, …, n

[7] The thesis will only provide a brief overview on the Lagrange method; for further information, please refer e.g. Sciavicco & Siciliano, 2000 or Pfeiffer & Reithmeier, 1987.

A physical interpretation of the parameters in equation 2.10 gives an intuitive understanding of the formula:

- The coefficients b_{jj} (i.e. the elements on the diagonal of the matrix **B**) represent the moment of inertia at joint axis j in the current manipulator configuration, when all other joints are blocked. The coefficients b_{ij} with $i \neq j$ account for the effects of acceleration from joint i on joint j.
- The coefficients in the **C** matrix represent the influence that one joint has on another, as well as the Coriolis effect induced on a joint by the velocities of two other joints.
- The terms g_i (components of the vector **G**) represent the momentum generated at the joint i axis of the manipulator in the current configuration, due to gravity.

Some of these parameters in the matrices can be simplified during the design process of the robot, e.g. by using massive and stiff materials for the links. Another simplification would be the use of gearboxes with high transmission ratios in the joints: in this case, all the coefficients in the **G** and **C** matrices can be neglected, and the **B** matrix becomes almost diagonal. This means, that every joint has influence on itself only. However, the objective of this work is to find a model for a humanoid robot. Recalling the constraints on humanoid robots from chapter 1.2 (e.g. compliant, lightweight), there are not many possibilities for simplification. Gearboxes, e.g., may not be used, as the robot will become too stiff; massive materials will make the robot unbearably heavy.

No matter how simple the robot setup is designed, there still remains the problem of estimating appropriate coefficients in **B**, **G**, and **C**. Modern CAD systems provide parameter estimation based on the design data of real systems. But this estimation is not simple, and the results are poor most of the time. Usually, the process includes idealized assumptions of the system's mechanics and uncertainties in the manufacturing process. In addition, some properties of the links change during the robot's lifetime because of material wear. So this does not seem to be a useful tool for such complex designs as such of humanoid robots.

An alternative approach could be to estimate the parameters by using the running system and collecting data from it. One advantage of this approach is that the matrices **B**, **G**, and **C** offer linearity in the dynamic equations. This can easily be verified during the derivation of the energies when using the Lagrange approach, e.g. Sciavicco & Siciliano, 2000. This is a remarkable property, because it also means that linear methods can be used to estimate the parameters, e.g. least squares error measurement.

Knowing that linear parameter estimation is possible, one can impose a suitable motion on the robot and measure the appropriate quantities for estimation:

- the joint positions
- the joint velocities $\dot{}$
- the joint accelerations $\ddot{\textbf{è}}$
- the joint torques **ô**

Joint positions and velocities can easily be measured with appropriate sensors. For joint accelerations, however, no such sensors exist. Hence, numerical differentiation techniques have to be used to estimate the acceleration, and these cause non-negligible errors. In addition, most robots are equipped only with joint position sensors, which means that already velocities have to be differentiated. To obtain information about the acceleration, the error-containing velocities have to be differentiated again. The result may be that the overall quality of the sampled data is very low. A similar problem arises for the joint torques: hardly any robot is equipped with highly accurate torque sensors. In case of electrical actuation, a current measurement can be used as a substitute. Alternatively, a wrist force measurement also provides torque information.

Sampling data that represents the dynamics of motion in a significant area of the robot's workspace requires a variety of different trajectories to be imposed on the robot. Sampling data from circular motion, for example, can estimate good parameters for further circular motion, but it will not capture the essentials of point-to-point motion. The ideal procedure would be to sample the entire possible robot’s motion and to us it for the parameter calculation. But this does of course exceed a realistic set of data in terms of time and size.

For the design of trajectories, special care has to be taken: If any of the robot’s physical limits are violated (e.g. joint limits, or current limits for actuators), the measurements contain invalid data and thus false parameters are calculated. Usually, an iterative process returns the best results, by increasing the parameters' accuracy in every iteration.

Coming back to the equation 2.10, we can now add some summarizing remarks: it is possible to estimate the unknown parameters hidden in the **B**, **G**, and **C** matrices, by using the rigid body assumptions and running trajectories to collect sensor information. With these parameters, the general form of the equation of motion can be used to calculate the required torques ( ) to accomplish a desired motion specified in positions ( ), velocities (˙), and accelerations (¨):

$$\mathbf{B}(\mathbf{è}) \cdot \ddot{\mathbf{è}} + \mathbf{C}(\mathbf{è}, \dot{\mathbf{è}}) \cdot \dot{\mathbf{è}} + \mathbf{G}(\ ) = \qquad \text{(same as eq. 2.10)}$$

2.6. Recapitulation

In the second chapter, we have introduced the basic problems of robot control: trajectory planning, inverse kinematics. In this thesis, I concentrate on the last issue, trajectory execution. In order to execute a trajectory specified in $\mathbf{è}_{\mathrm{des}}$, $\dot{\mathbf{è}}_{\mathrm{des}}$, and $\ddot{\mathbf{è}}_{\mathrm{des}}$, I have presented an error-correcting PID-controller in chapter 2.3. The concept of a feed-forward controller based on the system's dynamics is introduced in chapter 2.4. to improve the performance. The Lagrange concept for estimating the dynamics model of the system, is briefly described in chapter 2.5, resulting in the general formula given in equation 2.10.

It is now possible to merge the PD- and the feed-forward controller into one single equation, which controls the robot’s behavior (as illustrated in figure 2.4). The proper control function in joint space is to be found:

$$\mathbf{u} = \mathbf{B}(\mathbf{è}_{\mathrm{des}}) \cdot \ddot{\mathbf{è}}_{\mathrm{des}} + \mathbf{C}(\mathbf{è}_{\mathrm{des}}, \dot{\mathbf{è}}_{\mathrm{des}}) \cdot \dot{\mathbf{è}}_{\mathrm{des}} + \mathbf{G}(\mathbf{è}_{\mathrm{des}}) + \mathbf{k_P}(\mathbf{è}_{\mathrm{des}} - \mathbf{è}) + \mathbf{k_D}(\dot{\mathbf{è}}_{\mathrm{des}} - \dot{\mathbf{è}}) \qquad \text{(eq. 2.11)}$$

with

u denoting the generated control command.

As can be seen, the first three terms come from the inverse dynamics, the last two from the PD-controller.

This strategy makes the control of the robot much faster and more accurate compared to only PID control. There is, however, a big problem: the inverse dynamics has to be computed online all the time. This expensive computation might cause problems in high frequent servo loops. Furthermore, higher derivatives of sensor readings are required to calculate velocities and accelerations accurately. To obtain good values for the derivations, the sensor readings have to be filtered. But high frequent servo loops are required for filtering. So on one hand, low frequent servo loops are needed because of the computational complexity, but on the other hand, high frequent servo loops are required by the filter. Given these constraints, the servo loop frequency might become a critical parameter. Thirdly, the formulae are based on the assumption that perfect knowledge of the dynamics model exists. In reality this is, of course, never the case. All these limitations will cause unpredictable errors in real systems' motion. Still, the control function is extensively used it and achieves good results.

A further aspect to point out is the fact that the control function shown in equation 2.11 uses a hybrid centralized-decentralized control strategy. The PID controller is decentralized, i.e. it works independently on all joints: Thus, the error-correction components of the command (**u**) are calculated independently of the other joint-states. In contrast, the feed-forward component of the command considers all other joint-states. The later approach needs to be taken, when coupling terms between joints cannot be neglected. In our case and for our robot system, an appropriate solution is, to have one central feedback controller and *n* independent PID-controller, one for every joint. A decentralized feed forward controller cannot work properly, as the robot runs on low gains and has high coupling influences between the joints.

2.7. Control of Humanoid Robots

In this chapter, the previous discussion about control strategies will be applied to humanoid robots. These robots have to be lightweight, have compliant joints, and use low gain controllers to maintain the joints' compliance.[8] The PID-controller discussion in the previous chapters can directly be transferred to humanoid robots. The Rigid Body Dynamics Framework, however, is not well suited for humanoid robots, as the assumptions are not met. The rigid body model assumes stiff links between the joints no other non-linearities than those modeled by the Newton-Euler equations. Traditional industrial robots are build as much as possible to match the rigid body dynamics framework. Humanoid robots, on the other hand, are built to mimic human dexterity. Light-weight design and compliant control in a human-like arrangement of the degrees-of-freedom introduce non-linearities that are not captured by the rigid body dynamics.

All these reasons reveal, that rigid body dynamics, and hence the above-described model for generating feed-forward commands, is not an appropriate choice for humanoid robots.

The underlying assumption of my approach is that learning the feed-forward control function can be done and will result in superior performance compared to parameter estimation for rigid body dynamics. In this study, I concentrate on finding an alternative way to calculate feed-forward commands for performing a desired motion. This means, mathematically speaking, that the overall goal must be to exchange the first three terms of equation 2.11 by 'something else' that will perform better. Ultimately, the control function

$$\mathbf{u} = \mathbf{B}(\boldsymbol{\theta}_{\mathrm{des}}) \cdot \ddot{\boldsymbol{\theta}}_{\mathrm{des}} + \mathbf{C}(\boldsymbol{\theta}_{\mathrm{des}}, \dot{\boldsymbol{\theta}}_{\mathrm{des}}) \cdot \dot{\boldsymbol{\theta}}_{\mathrm{des}} + \mathbf{G}(\boldsymbol{\theta}_{\mathrm{des}}) + \mathbf{k_P}(\boldsymbol{\theta}_{\mathrm{des}} - \boldsymbol{\theta}) + \mathbf{k_D}(\dot{\boldsymbol{\theta}}_{\mathrm{des}} - \dot{\boldsymbol{\theta}})$$

(same as eq. 2.11)

becomes

$$\begin{aligned}\mathbf{u} = &\ \textbf{Something that replaces the Analytical Inverse Dynamics}\\ &+ \mathbf{k_P}(\boldsymbol{\theta}_{\mathrm{des}} - \boldsymbol{\theta}) + \mathbf{k_D}(\dot{\boldsymbol{\theta}}_{\mathrm{des}} - \dot{\boldsymbol{\theta}})\end{aligned}$$

[8] Please refer to chapter 1

3. Introduction to Robot Learning

3.1 General Remarks on Robot Learning

Robot learning can be seen as a subclass of learning control, which again is a subclass of general function learning. The main idea is to acquire a sensory-motor control strategy for a particular motion task. Usually, this is done by trial-and-error approaches; hence, the robot is allowed to fail during training. During an initialization stage, for example errors in the position of the end-effector may be tolerated if the robot enhances its performance while learning.

Referring to the discussion in chapter 2.1, there are three separate control problems that can be learned in terms of motor control: trajectory planning, trajectory transformation (e.g. from Cartesian to joint space, the inverse kinematics problem) and motion execution. It is advantageous to learn these problems independently, opposed to learning one single relation between desired behavior and motor commands. If only one mapping was learned, every small change in the prerequisites of the desired task requires a new model and complete relearning. As an example, a robot carrying an object will also have to relearn trajectory planning (instead of only trajectory execution), if the whole mapping was melted in one big function. In the case of separately learned modules, an unknown desired trajectory might be composed of pieces that the execution module has learned before during other tasks. Thus, only the planning module has to re-learn. These simple examples highlight the advantages of learning different stages of control with different modules.

In this thesis, I will concentrate on the execution of previously planned motion. Hence, I will focus on topics in learning that are relevant for motor control. A brief introduction on learning inverse kinematics is provided in the conclusion, chapter 8.2.

A closer look at the concept of the PID-controller (equation 2.5) reveals, that there are no open parameters or uncertainties to be learned: The only unknown parameters are the gains $\mathbf{k}_P$, $\mathbf{k}_D$, and $\mathbf{k}_I$, and these gains can be used as a tradeoff parameter between the system’s stiffness and its accuracy in keeping a position. Usually, the range of the gain is determined by the desired task, e.g. humanoid robots will operate on very low gains.

In contrast to the PID-controller, the feed-forward controller (equation 2.7), that has so far been approximated using rigid body dynamics (equation 2.10), needs to be improved. The following diagram (figure 3.1) provides an intuitive view of the way a learning system will interact with the control system presented so far:

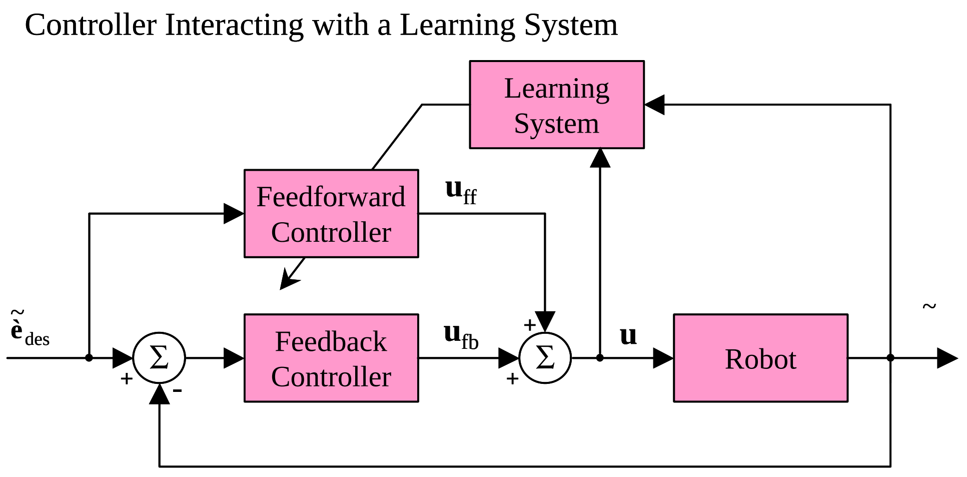
Controller Interacting with a Learning System
Learning System
Feedforward Controller
$\mathbf{u}_{\mathrm{ff}}$
Feedback Controller
$\mathbf{u}_{\mathrm{fb}}$
$\mathbf{u}$
Robot
$\tilde{\mathbf{è}}_{\mathrm{des}}$
Σ
Σ
+
-
+
+
~

Figure 3.1 Learning the Feed-forward Control Policy

The diagram shows a learning system that adjusts the feed-forward controller based on what it observes the system doing. The learning system will retrieve data samples in pairs, consisting of a command and the robot's state after executing this command. These pairs are given by data vectors $(\mathbf{u}, \tilde{\mathbf{è}})$. The learning system will try to draw general conclusions from the data it has experienced.

Unfortunately, a core question remains: what is it that should be adjusted by the learning system? As described in 2.7, the rigid body dynamics assumptions will not hold for humanoid robots. Therefore, estimating the parameters hidden in the **B**, **G**, and **C** matrices (equation 2.10) will not provide an adequate solution. Very generally speaking, the learning system is searching a function that will map the state vector of a system (~) to a control vector (**u**) - and this is exactly what is done by the feed-forward controller:

$$\mathbf{u}_{\mathrm{ff}} = f_{\mathrm{ff}}(\tilde{\mathbf{è}}_{\mathrm{des}}, \mathbf{á}, \mathrm{t}), \qquad \text{(eq. 3.1, same as eq. 2.7)}$$

Using a general learning approach (e.g. on the basis of neural net techniques), not only the parameters ($\mathbf{á}$) of given function will be learned, but also the function $f_{\mathrm{ff}}(\circ)$ itself. In the beginning of the procedure, the knowledge about the function to be approximated is very limited. In the case of humanoid robots, hardly anything is known about the inverse dynamics model. So, learning can be seen as approximating unknown functions by averaging over given samples of input/output relations. The only assumption I use about the function to be learned is that it holds the standard form:

$$y = f(\mathbf{x}) + \quad , \qquad \text{(eq. 3.2)}$$

where $\mathbf{x} \in \Re^{n}$ is an n-dimensional input vector, the noise term[9]   has mean zero, $E\{å\} = 0$, and the output is one-dimensional.[10] This assumption implies that the relation between input and output can be expressed by a function at all. The local telephone book, for example, contains a huge amount of data; nevertheless, any learning approach will fail, as there is no way to generalize from names to numbers. The only chance to learn it is to store every piece of information and retrieve it when needed.

3.2 The Bias / Variance Tradeoff

Data samples that are used to learn are most likely contaminated with noise, so no single data measured from the system can be assumed to be absolutely correct. Still, the function has to be estimated only based on the data presented. Further knowledge about the function is not available, e.g. the order of a polynomial or the fact that the function is a polynomial at all. This introduces a well-known problem in learning, the 'bias-variance tradeoff':[11] How much shall an algorithm generalize, i.e. smooth between data samples, or assume the data seen is exact. A visualization of this problem is given in figure 3.2:

[9] Modeling e.g. sensor noise
[10] Multi-dimensional output can be learned by learning multiple one-dimensional outputs independently of each other.
[11] Geman, Bienenstock, & Doursat, 1992

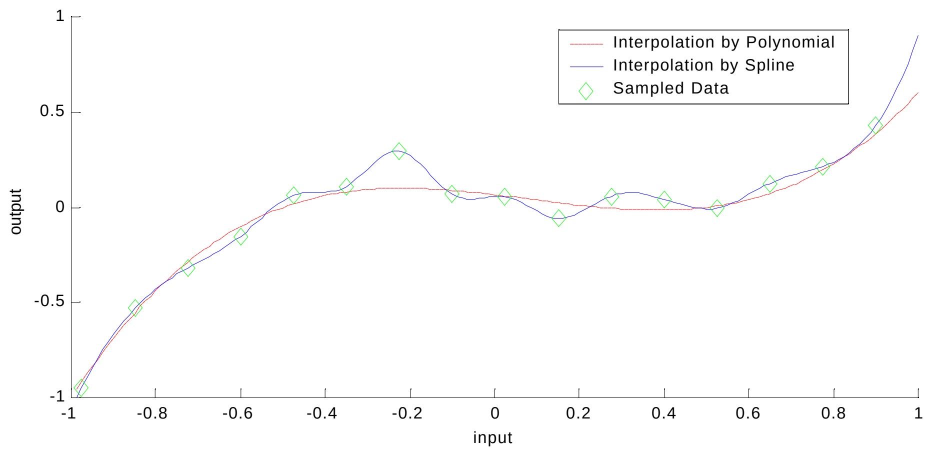

Figure 3.2 Bias / Variance Tradeoff for Sampled Data

The diamonds represent noisy data samples from sensor readings. The solid line shows a spline interpolation, and it is easy to see that it definitely over-fits the data. It is not difficult to imagine that new sensor readings will only accidentally fall on the line. And it becomes clear that a spline interpolation will not result in a perfect prediction of unknown data. The dotted line, however, representing a third order polynomial may over-smooth the data.

The problem may be solved by using distinct sets of data for training and validation. If the performance of the regression is evaluated from previously unknown data, i.e. the test-set, then the regression has to generalize in order to match the 'new' data properly. Otherwise, if the algorithm learned every single training information only, it will fail to predict the test set, as it has never seen the test data before. Thus, the algorithm has to capture the essentials of the function to achieve good predictions for the test set. Evaluating the approximation's performance on unseen data is a widely used method in unsupervised learning.[12]

In order to evaluate the regression’s performance, a cost function must be introduced. This function, usually a squared error criterion, has to be minimized to achieve best performance:

[12] The term 'unsupervised learning' refers to a learning system without a teacher providing feedback on how to increase performance.

$$J = \sum_{i=1}^{N} \left(y_{\text{des,i}} - y_{\text{i}} \right)^2 \qquad \text{(eq. 3.3)}$$

where ($y_{\text{des,i}} - y_{\text{i}}$) is a measure of the displacement between the prediction at a given query point i (y_{i}), compared to the real function value of point i ($y_{\text{des,i}}$). N denotes the total number of test data samples available. Squaring the error will additionally penalize big errors. The underlying idea is that it is worse to have one big error compared to having few smaller errors. The cost function represents a measure of the learning system’s overall approximation quality. Minimizing the cost function implies that there are parameters in the learning model that can be adjusted during learning. This adjustment can happen in a variety of ways, with many possible functions in the learning system. Only two examples are given by polynomials with adjustable coefficients, or by neural nets with adjustable connection strength between single neurons.

3.3 Global versus Local Learning Strategies

A major differentiation in learning models is the distinction between local and global approaches. Global approaches use activation functions that are not limited to a finite domain in the input space x. Such can be polynomials, or, in the case of neural nets, a sigmoidal activation function. Let us take a more detailed look at the second, which is given by

$$y = \frac{1}{1 + e^{-a\,x}} \qquad \text{(eq. 3.4)}$$

with a being a parameter to adjust the slope of the activation function. Once a threshold is passed ($x_{\text{thres}} \approx 0$ in equation 3.4), the activation function will return values significantly different from zero. Also, polynomials are valid for all the input range of *x*. In contrast, local learning uses activation functions that are non-zero in a restricted domain only, like Gaussian activations given by

$$y = e^{-s\,x^2} \qquad \text{(eq. 3.5)}$$

where s denotes the variance of the Gaussian. Figure 3.3 shows the graphs of these two functions that are mainly used as activation functions in neural nets.

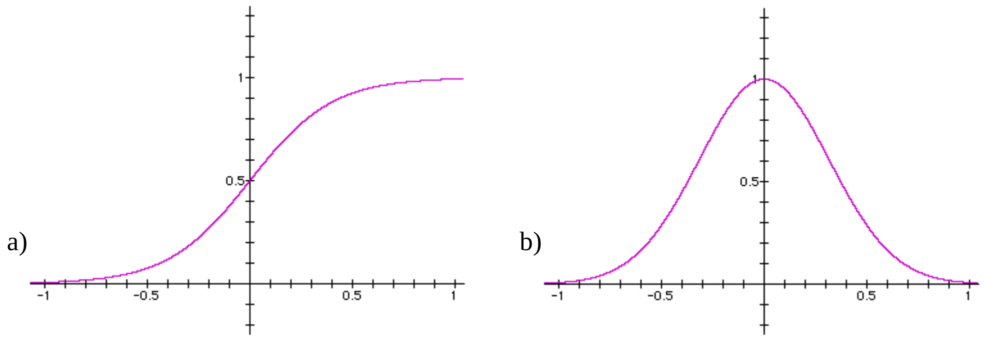

Figure 3.3 Sigmoidal (a) and Gaussian (b) Activation Function

The parameter estimation for rigid body dynamics from chapter 2.5 can be seen as a global function approximation. The equation of motion (equation 2.10) is valid for the whole input range and thus, all parameters in **B**, **C**, and **G** have to be valid for the whole range.

From a theoretical point of view, both approaches (local and global learning) are capable of approximating arbitrarily complex functions. However, the learning speed, the time for convergence and their applicability towards high dimensional input data differs a lot. This is especially important for the type of input data we expect from robotics.

A closer look at global function approximation will help to understand the difference: Imagine that data from a limited range is used to train the learning algorithm, like a robot performing a certain task. The approximation will become highly accurate in this area. If however, during further training only data from another input range is learned, the approximation will focus on the new range only and re-adjust the parameters appropriately for this range. This means it will 'forget' what it has been trained before: The approximation will become significantly worse in the old training range compared to how it has performed before. This effect of forgetting what has been learned before is termed 'Negative Interference' and will be discussed in chapter 4.2.2. The only solution to this problem is to store all data and re-learn everything once new input ranges come into play. But storing all data from real robots’ motion is unfeasible due to the huge amount of information.

Another important aspect of global learning is the adjustment of meta-parameters. These parameters have to be set before learning begins and they remain unchanged while learning. Examples for these meta-parameters are the order of a polynomial or the number of neurons in a neural net. If these meta-parameters turn out to be inappropriately adjusted, they can only be changed when re-starting the whole learning process. A meta-parameter modification has significant influence on the overall performance and will make it necessary to re-learn all parameters.

In contrast to global learning, local approaches use activation functions that are non-zero in a limited range of the input dimension only, as shown in figure 3.3b. An intuitive approach to understand local learning starts with the assumption that a complex function is represented by small, simple local patches, e.g. constants or low order polynomials. The size of a local patch is determined by its activation function. Not only the activation function, but also parameters like the position and the approximation function of a patch have to be adjusted. Additionally, the overall number of patches used to achieve accurate function representation throughout the input domain has to be estimated. A significant difference of local learning compared to global learning techniques lies in their ability to learn data from different input domains sequentially, and without suffering from negative interference. Intuitively, data from new domains will activate only those local patches that 'live' close to the new domain, and, thus, they will update only these patches. The patches from previously learned domains remain unchanged.

3.4 The Curse of Dimensionality

Another problem which local learning approaches face copes with the fact that the number of patches needed for good approximation explodes in high dimensional spaces. This problem is called the 'Curse of Dimensionality' and can be illustrated by a simple example:

Assume that every input dimension shall be divided in only 10 local regions. This means there are 10 total patches for one input dimension, $10^2 = 100$ patches for two dimensions, $10^3 = 1000$ patches for three dimensions; up to 10^n patches for n input dimensions. For only 12 input dimensions, we get 10^{12} which roughly compares to the number of neurons in the human brain - and, as all local patches have to be checked for activity, an output prediction can definitely not be computed in reasonable time. As a solution, one might increase the size of the patches, i.e. use fewer patches along every input dimension. But increasing the size does not seem to be a very satisfying solution, as this will also decrease the accuracy of the prediction.

Facing the problem from the other side will lead to better results: Collecting data samples at 100Hz will roughly need 300 years of continuous, non-repetitive data to obtain 10^{12} data points. No robot will collect motion trajectories for 300 years to explore its workspace! This observation leads to the assumption that real motion data is locally distributed in very few dimensions, even if it is globally high dimensional.[13] Thus, the implementation of local dimensionality reduction techniques can help to learn efficiently in high dimensional spaces.

3.5 Online-Learning

Another problem to keep in mind is how the stream of incoming data is processed. While the robot is in motion, a huge stream of data arrives, summing up to several KB every second. This data has to be used for learning; either it is stored and processed later in total (called 'batch learning') or it is processed online and discarded afterwards (i.e. it is learned immediately when new data arrives). The second possibility of data processing is by far superior. In the first, data will sum up to huge sets. They will have to be stored, and learning on such huge sets of data needs much time. Online learning in the contrary makes it possible to continue learning once the robot is performing its desired task, and to continuously improve the learned function as the robot moves. This, of course, implies

[13] Explaining the learning algorithm in chapter 4 will raise this question again and provides further examples.

real-rime learning, meaning that the data has to be used for learning as soon as new data arrives. If the data is not processed within a fixed time, a continuously increasing pile of incoming data rises. This pile cannot be processed in total, until the robot finishes its motion. To process data in real time, one has to force constraints on the computational complexity of the learning approach. In our case, local linear models (i.e. with only few adjustable parameters) will lead to a computational complexity that is online solvable for all seven joints with sensor readings at a frequency of 50Hz.

An additional advantage of using online learning is that it offers the possibility to increasing the quality of the learned function continually. This is especially valuable once the robot is performing its desired task and therefore moving close to the desired trajectory. Repetitive motion close to the desired trajectory will generate data in the neighborhood of the desired motion, and the learning approach can focus on improvements in this region.

Continuous learning will cure another problem that arises due to material wear and change of dynamics properties. As the robot ages, its mechanics changes slightly, and thus the previously learned function (or the estimated parameters in **B**, **C**, and **G** in the case of rigid body dynamics) are no longer accurate. With continuous learning, the new sensor readings will reflect these changes and the learning algorithm can incorporate the changes appropriately. The learned regression will always stay accurately no matter how much change in dynamics properties is accumulated over time.[14]

Using the robot's motion to learn its inverse dynamics also bears risks. E.g., one has to be careful in designing trajectories to explore the workspace: A good starting exploration will lead to fast learning and good performance in further explorations. In contrast, a bad exploration decreases the ability to learn and leads to poor further exploration. It has therefore to be ensured that the robot covers a representative range of distinct motions from the very beginning.

[14] Assuming that the change happens continuously.

3.6 Learning Inverse Dynamics

After the above discussion on robot learning and the discussion on the inverse dynamics model of humanoid robots in the previous chapter, one can easily realize that learning is a suitable approach for the approximating of the feed-forward function.

Sensory-motor transformation in high dimensional spaces for the inverse dynamics problem is one of the basic prerequisites to ensure the success of autonomous robots. Engineering models, on the other hand, are often not capable of modeling the mechanisms used to build humanoid robots accurately, and it turns out that using rigid body dynamics is a poor approach when working with humanoid robots. An alternative way can be seen in function learning without any prior assumptions on the function to be learned. A learning algorithm that reads sensor information and automatically acquires useful local models to predict feed-forward commands can easily replace the rigid-body-dynamics equation of the feed-forward model. This learned approach leads to superior performance and further improvement while the robot is operating. The ultimate goal must be to compare the robots' performance with human motion, as well as to compare the algorithm's learning behavior in terms of speed and complexity to those of humans.

3.7 Results

In this chapter, I have introduced the basic concepts and problems of robot learning. The first issue was the bias/variance tradeoff. It deals with the question how much the interpolation between the noise-contaminated data shall be smoothened to represent the real function accurately. A solution was found in using separate data sets for training and testing. We saw that consequently the parameters of the learning algorithm have to be adjusted in such a way that a cost function on the test data is minimized.

In the second section, I have introduced global learning strategies. Local strategies seem to be more successful for learning a robot control problem, as they can deal with shifting input distributions and can change meta-parameters more easily than global strategies. The third section discussed the 'Curse of Dimensionality' - the problem of needing a huge number of patches for local function representation with high dimensional input data. When working on motion data from real robots, this problem can be solved by projecting the input data on a few local projection dimensions. Finally, the topic of online learning was discussed. Online learning addresses the computational speed needed to incorporate incoming data and offers further increasing accuracy while keeping the robot in motion.

Local learning techniques can deal with all these issues, using local dimensionality reduction techniques and local simple models for function fitting. Additional questions address the number and position of local models, which are needed to approximate the function accurately. The parameter adjustment of the local models also has to be thought of.

In the following chapter, the learning algorithm *Locally Weighted Projection Regression* (***LWPR***) will be introduced. This algorithm is designed to deal with the above-mentioned problems - name the problems arise with online-learning of motion data for the inverse dynamics problem. The algorithm will demonstrate that learning the inverse dynamics model can lead to far superior performance compared to the analytical approach using rigid body dynamics assumptions.

4. The Learning Algorithm *LWPR*

This chapter explains the algorithm *Locally Weighted Projection Regression* (***LWPR*)** that will be used for learning the inverse dynamics controller. It follows the outline and recapitulates the basic ideas of the publications Schaal & Atkeson, 1998, Vijayakumar & Schaal, 2000a, Conradt, Tevatia, et al., 2000, Vijayakumar & Schaal, 2000b. Please refer to these publications for references and topics that could not be covered in more detail in this thesis.

4.1 Advantages of Learning Approaches

One of the most significant properties of biological systems is their capability to adapt to constraints induced by their environment. They can change their behavior into a somehow advantageous way and perform better and better in their natural surroundings. This 'learning' from the interaction with the environment happens in real time and based on incrementally incoming data from a variety of sources. Of course, biological sensors like tactile-, odor-, and acoustic-sensors are not free of errors, so the biological 'input' for learning is contaminated by noise.

It seems, thus, that biological learning has to face the same problems as the artificial learning approach introduced in chapter 3. In the case of motor control, animals learn how to generate motion from noisy data distributions in unknown high dimensional spaces. Biology has solved the problem very well, whereas engineering solutions still perform comparatively poor unless the respective system maintains very special properties, e.g. the rigid body dynamics assumptions as shown in chapter 2.5. Unfortunately, there is nothing like a black box labeled 'learning' that can be connected between input and output and that will do the learning job for every possible function to be learned. However, such a black box is a desirable goal, so that one can use it for regressing the dynamics of the robot without any prior knowledge. Up to today, artificial learning approaches have to incorporate prior knowledge of the data distributions and the amount of data that has to be learned.

Spatially localized learning is a possible way to deal with these shortcomings. Local information processing has been known for a long time from neuro-physiological experiments (e.g. Mountcastle, 1957, Hubel & Wiesel, 1959), and since then, a huge amount of research has been done to show advantages of local information processing in neurobiology (e.g. Lee, Rohrer, & Sparks, 1988, Georgopoulos, 1991, Field, 1994, Olshausen & Field, 1996, Daughman & Downing, 1995). Roughly a decade ago learning with spatially localized functions has started to become a popular paradigm in machine learning and neurobiological modeling, e.g. with radial basis networks (e.g. Moody & Darken, 1988, Poggio & Girosi, 1990). The theoretical issues in local learning can be solidly grounded in approximation theory (Powell, 1987). A learning system suitable for inverse dynamics control that is based on local models should fulfill at least the following properties:

- it should be able to learn from noisy continually arriving data without the need of storing and without the danger of forgetting previously seen data
- it should have the ability to estimate the appropriate number of resources, i.e. local models, and change these models depending on the input data
- it must be capable of dealing with a large number of possibly redundant and / or irrelevant inputs

This chapter will present a new algorithm called *Locally Weighted Projection Regression* (***LWPR***) developed by Stefan Schaal, Chris Atkeson and Sethu Vijayakumar that is well suited to fulfill the above requirements. ***LWPR*** is the latest child in a series of algorithms that incrementally improved the function approximation quality in terms of accuracy and computational performance by modifying the methods used for predictions. The statistical assumption underlying the approach is that the unknown function to be modeled holds the form $\mathbf{y} = f(\mathbf{x}) + \mathbf{å}$, with **x** being an n-dimensional input vector, **y** an m-dimensional output vector. The term   denotes an additive random noise with the assumption that   is independently distributed, $E(e_i, e_j) = 0$ for i ≠ j and mean zero.

The basic idea of the ***LWPR***-algorithm is to locally approximate the unknown function by linear models. Predictions for a given query point will then be made from local neighborhoods around the query point only. The area of validity of each local approximation is called a 'Receptive Field'. Each receptive field is trained independently of all other receptive fields, i.e. it adjusts its parameters (the size and shape of the region of validity and the coefficients of the linear function) without knowing the other fields' parameters. New receptive fields are allocated as needed, whereas those fields that are not needed anymore become pruned.

4.2 Desired Algorithmic Properties

4.2.1 Infinite Incremental Update

The basic concept of incremental update is that data produced by the robot's motion will not be stored and ultimately processed - it will rather be processed immediately once it is generated. This is useful, as the amount of data will grow all the time during the robot's motion. Not only storing but also processing the huge amount of data leads to a computational complexity that is not feasible. Additionally, incremental learning will enable the system to continue learning once the first learning stage has past. There is no time when training stops and the robot continues with only using the learned function to generate motion. Instead, motion will be predicted by the learned function and while executing the motion, the learning system can continue to improve its performance. The system will learn throughout its life.

The continuous learning approach does also address another problem in motion generation: It enables the robot to cope with input domains that have not previously been learned. For example, when performing two very different tasks like driving a car and riding a bike, very different strategies for motion control are required. If learning has stopped before the new task has started, the robot will perform poorly and will not have a chance to improve. A continuous learning system in contrast will allow the robot to increase the performance everywhere in his workspace. This scenario resembles the learning of sensory motor transformation in biology and, again, biological systems are performing very well at it.

Two major problems arise, when using a continually learning system: Firstly, the robot might forget what has been learned before when it learns data in new input domains. Secondly, the problem of allocating the appropriate number of local receptive fields in order to represent the data accurately has to be addressed. Both issues, which all incremental learning systems have to face, will be discussed further in the following two chapters.

4.2.2 Limited Negative Interference

Interference is a natural side effect of the ability to generalize. Every learning approach needs to generalize from the data it has seen; otherwise, it would simply behave like a huge look-up table and not 'learn' the underlying model that generated the data. So, predictions for unseen data will always be made by generalizing from similar previously encountered data. Generalization is achieved by adjusting those parameters of a learning system that have non-local effects. If these effects reduce the overall correctness rather than improve it, the interference is called 'negative' or even 'catastrophic interference'. The question of interference is especially important for incremental learning, as incremental learning systems cannot balance positive interference (i.e. generalization of the presented data) with negative interference. Because old data is not present anymore, every parameter update happens with respect to the current data only. For learning inverse dynamics model, this might lead to the problem that a robot learning to play tennis (with fast swinging arm motion) forgets that it previously learned how to chop vegetables in a kitchen.

However, localized receptive fields have a potential robustness towards interference: if learning is spatially localized, the interference will be spatially localized as well. This means, that during learning only such local fields will be active, that contributed to the prediction at the given input. Thus, only these local fields will be updated, and only their parameters will be changed. It is easy to imagine, that for the two tasks described above (playing tennis and cooking), only distinct receptive fields will be active due to the very different input data. In this case, training data has negligible effect on the parameters of distant receptive fields, and cannot change learned predictions from distant local models.

An artificial example to illustrate the interference problem in a one dimensional input space is shown in figure 4.1:

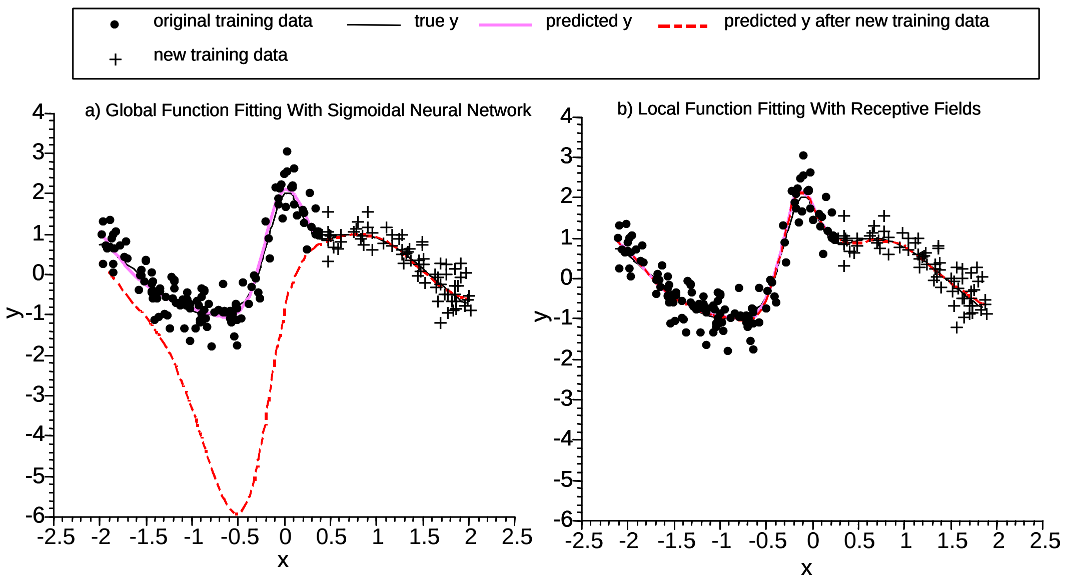

Figure 4.1. An example for interference using neural nets and local receptive fields (taken from Schaal & Atkeson, 1998)

The data generating function used for learning is given by

$$y = \sin(2x) + 2 \cdot \exp(-16x^2) + N(0,0.16) \qquad \text{(eq. 4.1)}$$

in the range of $x \in [-2.0, 2.0]$. This synthetic data set suggested by Fan and Gijbels (1995) was used to train a three-layer sigmoidal feed-forward neural net with six hidden units. The training took place in two regions: First 130 noisy points of data were drawn from $x \in [-2.0, 0.5]$, shown as ● in figure 4.1a. The excellent function fit is shown by the 'predicted y' trace. Then, training the network was continued using 70 noisy inputs from the range $x \in [0.5, 2.0]$, illustrated by + in figure 4.1a. The network learned to predict the new data accurately, but in doing so, it significantly changed the prediction in the previously learned range $x \in [-2.0, 0.5]$. The resulting prediction for the whole input range is illustrated by the dotted line in figure 4.1. This change of the prediction happens due to the global activation of the hidden neurons that change their properties to adapt for the new data.

In strong contrast to the neural network, the local learning approach with receptive fields used for consecutively learning the same function hardly shows any change after training on the new data (figure 4.1b). Interference cannot be seen in predicting the function. This robustness towards negative interference is accomplished by the local nature of the learning approach: Only those receptive fields are updated that accomplish function approximation in the region of the input data. Therefore, input data from a different region cannot modify previously learned data.

4.2.3 Automatic Resource Allocation

Traditional learning approaches have to estimate meta-parameters before the learning process starts to achieve a good compromise between over-smoothing and over-fitting the data (please refer to chapter 3.2). Usually, this is done on the first representative set of data during a model selection phase. Alternatively, a human supervisor can estimate the meta-parameters while designing the learning system. These parameters must remain unchanged once they are selected; otherwise learning has to start all over again. The question is: How many free parameters should be adjusted, and can appropriate values for these be found to achieve good learning results?

Unfortunately, this question cannot be answered in general, so it remains unclear how to find good meta-parameters for general function approximation. For spatially localized function approximation, however, the question arises in a slightly different way: How wide is the region of validity for a certain local model? The total number of local models is directly correlated to the width of all single models, as the whole input range has to be covered by one model at least. This also implies, that the number of local models can change (only increase), when new regions of input data are explored. This means that the global problem has been transformed to a local one: each receptive field has to find good parameters to regress the function.

But how wide can the region of validity be expanded so that the assigned linear model still fits the data? Figure 4.2 illustrates a local model with a maximal tolerated error bound by m. The region of validity (shown as a Gaussian function) will be adjusted such that the bound m will hold within the valid region. If the model learns new data that does not fit in the linear model with respect to m anymore, it will shrink the region of validity of this receptive field. The original function is more like a linear function in a smaller region, which will allow to hold the bound m. Eventually, a new receptive field will be introduced to model the data between two receptive fields that have shrunken.

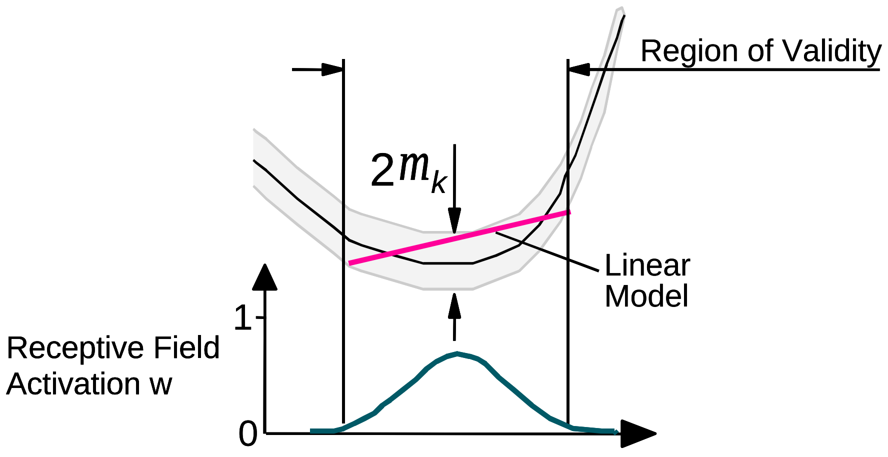

Figure 4.2 Region of Validity of a Local Linear Model
(adapted from Schaal & Atkeson, 1998)

Having a set of error-bound locally valid receptive fields will allow the system to model complex functions within a wide input range. All the input range has to be covered by at least one local active model to get a prediction for this input. If more local models are active, the overall prediction will be the weighted average over all active models' predictions (as will be explained in chapter 4.4.). Calculating a weighted average of the active fields will ensure that the overall prediction is also bound by m. Indeed, having more active models predicting for a certain input will tend to improve the overall accuracy (Perrone & Cooper, 1993).

Despite of this simplification, it is hard to find the optimal local bias-variance tradeoff (Friedmann, 1984, Fan & Gijbels, 1996). Local models allow new ways to find solutions that are satisfying and computationally affordable. The leave-one-out cross-validation approach (see chapter 4.5.3) in conjunction with regularization techniques can be used to realize local bias-variance tradeoffs and to control the expected bias m in each linear model.

4.2.4 Dimensionality Reduction of the Input Data

Another problem is the sparsity of data in high dimensional spaces. Due to this sparsity, learning requires a huge amount of prior knowledge about the learning task, usually provided by a human supervisor. Unfortunately, this knowledge does not exist for the inverse dynamics model of a humanoid robot, because humanoid robots suffer from high non-linearities induced by their mechanical limitations. Non-parametric learning algorithms have advantages in learning such tasks (e.g. Geman, Bienenstock & Dursat, 1992), but they fail in high dimensional spaces. The concept of neighborhood is without use in these high dimensional spaces, as all data points have roughly the same distance to each other (Scott, 1992). So the power of describing similar properties of neighboring areas used by local learning approaches is gone. Given this problem, it seems that non-parametric learning algorithms have little merit for sensory-motor control.

However, when examining real motion data of human and robot studies, Vijayakumar and Schaal (1997) found that this data is locally very low dimensional. Original data that was sparse in a 21 dimensional space reduces to locally dense data in 3-6 dimensions only. This observation triggered the idea of using a preprocessing step to project the input data into a low dimensional space before using a linear regression to model the function. The dimensionality of the local low dimensional space for every receptive field must have the ability to grow dynamically as new data samples are used for training. If the dimensionality was fixed, new significant properties might not be represented in the local model.

There exists a variety of algorithms that perform a dimensionality reduction, among them principle component analysis, independent component analysis, factor analysis and partial least squares (PLS). A specially adapted variant of the last named PLS is used by the ***LWPR*** algorithm. Further details will be provided in chapter 4.3, which deals with the preprocessing of input data.

4.2.5 Summary of the Desirable Properties

Spatially localized receptive fields seem to be a promising route towards robust incremental learning. However, implementing local learning techniques for function approximation also requires some care for additional desired properties. In the case of inverse dynamics for humanoid robots the main properties are:

- Capability of Incremental Learning
  As data will be generated by the robot's motion all the time, it would be a waste of information to stop learning at all. But as no system can store and process an infinite amount of data, learning algorithm has to learn from incoming data and discard the data immediately after learning.
- Avoiding Negative Interference
  The robot's motion might explore new areas within its range of motion. While learning in the new range, previous knowledge must be kept unaltered such that the robot can return to its old operating area and continue moving accurately.
- Automatic Resource Allocation
  Meta-parameters like the size and number of receptive fields have to be adjusted automatically without a human supervisor. Only this property ensures superior performance for previously unknown function estimation.
- Dimensionality Reduction of the Input Data
  The important concept of neighborhood is lost in high dimensional input spaces; thus, the main strength of local learning approaches is gone. As high dimensional motion data is locally very low dimensional, a dimensionality reduction has to be performed as a preprocessing step on the input data.

The following chapters will demonstrate how all these properties are integrated in the learning algorithm ***LWPR*** that is especially suited for learning motion data.

4.3 Input Data Preprocessing

The only input-data-preprocessing that is necessary is the dimensionality reduction discussed above. In the previous years, research found good global projections for fitting non-linear one-dimensional functions, e.g. projection pursuit (Friedmann & Stutzle, 1981). ***LWPR*** however, is needs good local projections, that can be used to accomplish local function approximation in the neighborhood of a given query point. This will allow fitting simple functions with high accuracy on the given data in a local region. The regression problem is much simpler, when using linear instead of complex one-dimensional functions.

However, principle component regression does not provide help in finding efficient local projections, nor does it detect irrelevant input dimensions that might be falsely interpret as important directions. ***LWPR*** needs an algorithm that discovers efficient projection directions for locally weighted linear regressions. The basic idea is to project the input data $\mathbf{X}$ onto r orthogonal directions $\mathbf{u}_1$, $\mathbf{u}_2$, …, $\mathbf{u}_r$ along which they carry out univariate linear regressions, hence the name projection regression. If the linear model of the data were known, it would be easy to determine the optimal projection direction. It would then be given by the vector of regression coefficients beta, which is the gradient of the data. Only a single projection along this direction would result in the optimal regression. Unfortunately, the projection direction $\mathbf{u}$ has to be estimated from the data and usually remains sub-optimal.

The projection-technique used in ***LWPR***, Partial Least Squares, is extensively used in chemometrics (Wold, 1975, Frank & Friedman, 1993). This projection technique recursively computes orthogonal projections of the data and performs single variable regressions along these projections on the residuals of the previous iteration step. The important difference between PLS and most other projection algorithms is that PLS chooses projection directions according to the correlation of the input data with the output data, not on the basis of the input alone. Table 4.1 illustrates PLS in a pseudo-code implementation. For simplicity, a batch process is described instead of an incremental version. It is assumed that all the input data exists in the columns of the matrix $\mathbf{X}$ and the output data in vector $\mathbf{y}$. The derived incremental version will be shown in chapter 4.6, embedded in the pseudo-code implementation of the complete ***LWPR*** algorithm.

```
Partial Least Squares (PLS) Pseudo-Code
1. INITIALIZE: $\mathbf{X}_{res} = \mathbf{X}$, $\mathbf{y}_{res} = \mathbf{y}$
2. FOR i = 1 TO r DO        (r denotes the number of projections)
  (a) $\mathbf{u}_i = \mathbf{X}_{res}^T \mathbf{y}_{res}$
  (b) $\hat{\mathbf{a}}_i = \frac{\mathbf{s}_i^T \mathbf{y}_{res}}{\mathbf{s}_i^T \mathbf{s}_i}$        where $\mathbf{s}_i = \mathbf{X}_{res}\mathbf{u}_i$
  (c) $\mathbf{y}_{res} = \mathbf{y}_{res} - \mathbf{s}_i \hat{\mathbf{a}}_i$
  (d) $\mathbf{X}_{res} = \mathbf{X}_{res} - \mathbf{s}_i \mathbf{p}_i^T$        where $\mathbf{p}_i = \frac{\mathbf{X}_{res}^T \mathbf{s}_i}{\mathbf{s}_i^T \mathbf{s}_i}$
```

Table 4.1. Pseudo-Code of PLS projection

The first line in table 4.1 initializes the Matrix $\mathbf{X}_{res}$ (input) and the vector $\mathbf{y}_{res}$ (output) with the data given for approximation. The iteration itself starts with step 2a, which calculates the direction of maximal correlation **u** between the residual input and output data. Step (b) performs the univariate regression along the direction **u**, and returns the regression coefficients . The regression itself will be explained in chapter 4.5.2. Additionally, PLS regresses the data of the previous step against the projected data of this step (**s**). PLS therefore subtracts the used information from the input and output data to ensure the orthogonality of consecutive projections **u** (steps (c) and (d)). The regression in (d) modifies the input data $\mathbf{X}_{res}$ such that each resulting data vector has coefficients of minimal magnitude and, hence, pushes the distribution of $\mathbf{X}_{res}$ to become more spherical. This simple, yet efficient algorithm will return the desired number of projection directions (r) in $\mathbf{u}_i$.

As an open question remains, how the number of local projections r can be estimated. ***LWPR*** uses a simple trial-and-error approach: The initial number is set to r=2. Starting from this, ***LWPR*** recursively keeps track of the mean-squared-error (MSE) as a function of the number of projections included in a local model (i.e. step j in the ***LWPR*** pseudo-code in table 4.2). The algorithm will stop adding new projections locally, if the MSE at the next projection does not decrease by more than a certain percentage, i.e.

$$\frac{\mathrm{MSE}_{i+1}}{\mathrm{MSE}_{i}} > f \text{, where } f \in [0,1] .$$

This criterion leads to a good upper bound for the number of needed projections. Adjusting the parameter f has no major influence on the algorithm's performance, as additional dimensions will not decrease the regression results.

Using these simple mechanisms, the number and directions of the projections will be learned by the algorithm without human supervision and with only f as a rather insensitive meta-parameter that needs to be adjusted. From now on, the input vector **x** for the linear regression will be assumed to be the projection of the original input **x**. The position of the local model in input space however will be with respect to the original input **x**. This distinction is necessary to locate appropriate receptive fields and activate only these.

4.4 Predicting Output Data using *LWPR*

The ***LWPR*** algorithm constructs a system of K local linear models that will give individual predictions $\hat{\mathbf{y}}_k$ for a query point **x**. The weighted sum of all these single predictions is the algorithm's overall prediction:

$$\hat{\mathbf{y}} = \frac{\sum_{k=1}^{K} w_k \cdot \hat{\mathbf{y}}_k}{\sum_{k=1}^{K} w_k} \qquad \text{(eq. 4.2)}$$

In equation 4.2, $\hat{y}_k$ denotes the individual prediction of the receptive field k, whereas w_k is the weight (the importance) of this prediction. The weights w_k correspond to the activation strength of the receptive field, characterized by the shape of its kernel function. The kernel function is a measure for the distance from a query point **x** to the center of the receptive field. This follows the assumption that the distance is reciprocally correlated to the quality of the approximation. A variety of kernel functions have been suggested

(Atkeson, More, et al., 1997), ***LWPR*** uses a Gaussian kernel given by equation 4.3 for analytical simplicity.

$$w_k = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{c}_k)^T \mathbf{D}_k (\mathbf{x}-\mathbf{c}_k) \right) \qquad \text{(eq. 4.3)}$$

$$\text{with} \quad \mathbf{D}_k = \mathbf{M}_k^T \mathbf{M}_k \qquad \text{(eq. 4.4)}$$

Equation 4.3 characterizes the receptive field by its location in space ($\mathbf{c}_k$) and a positive definite distance metric $\mathbf{D}_k$. For convenience reasons, the distance metric $\mathbf{D}_k$ shall be constructed from an upper triangle matrix $\mathbf{M}_k$, so that $\mathbf{D}_k$ will stay positive definite.

Each receptive field models the relationship between input and output data by a simple parametric function. For these functions, locally low order polynomials have found widespread use in non-parametric statistics (e.g. Nadarayan, 1964, Watson 1964, Wahba & Wold, 1975, Cleveland 1979, Cleveland & Loader, 1995). ***LWPR*** uses linear functions according to equation 4.5, which seem to offer a good tradeoff between computational complexity and accuracy (e.g. Hastie & Loader, 1993):

$$\hat{\mathbf{y}}_k = (\mathbf{x}-\mathbf{c}_k)^T \mathbf{b}_k + b_{0,k} = \tilde{\mathbf{x}}^T \hat{\mathbf{a}}_k \qquad \text{(eq. 4.5)}$$

$$\text{with} \quad \tilde{\mathbf{x}} = \left((\mathbf{x}-\mathbf{c}_k)^T, 1\right)^T \qquad \text{(eq. 4.6)}$$

$$\hat{\mathbf{a}}_k = (\hat{\mathbf{a}}_k^T, \hat{a}_{0,k})^T \qquad \text{(eq. 4.7)}$$

Here, $\hat{\mathbf{a}}_k$ denotes the regression parameters of the local linear model k and $\tilde{\mathbf{x}}$ is a compact representation of the center-subtracted, augmented input vector to simplify the notation. The additional '1' is used to represent a bias term $\hat{a}_0$. Figure 4.3 gives a nice illustration of the functional blocks that the ***LWPR*** algorithm is composed of.

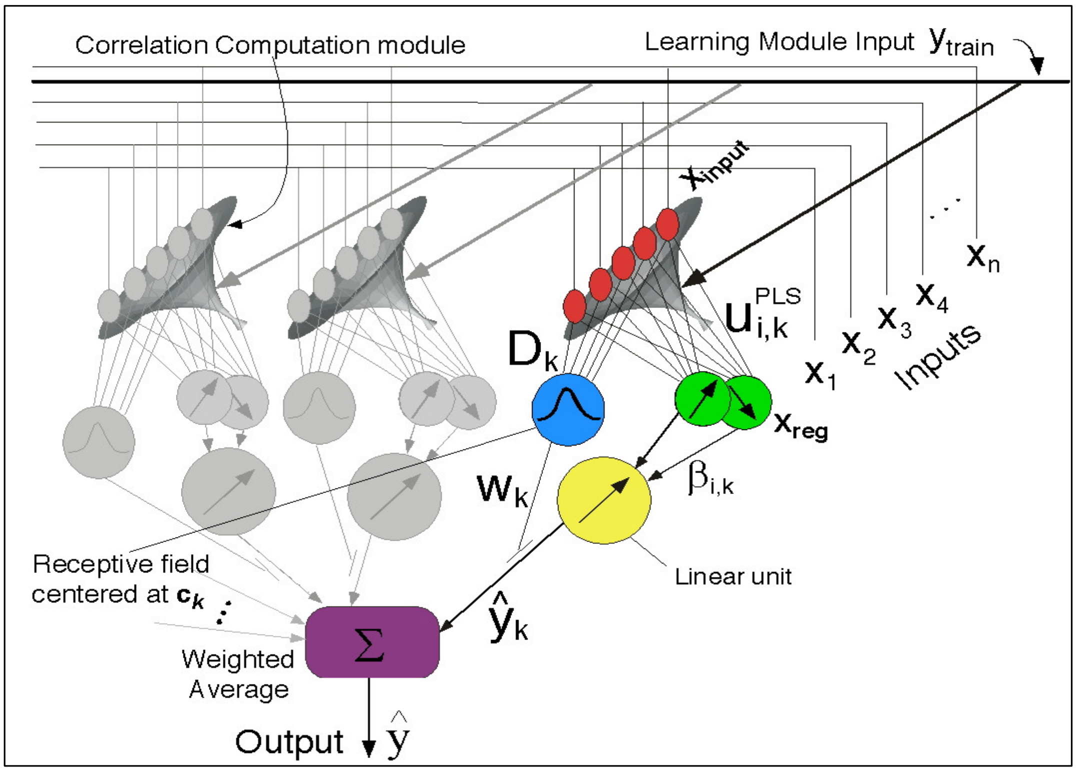

Figure 4.3 Architectural Overview of the ***LWPR*** Algorithm
(taken from Vijayakumar & Schaal, 2000b)

Three receptive fields shown in figure 4.3 each consist of a projection unit (**u**), a linear regression unit ( ), and a Gaussian weighting unit (**D**). The input **x** is routed independently to all receptive fields. The projections of the input **x** (achieved by multiplication with $\mathbf{u}_{i,k}^{PLS}$) are routed to the linear regression units. In this figure, only two projection directions are shown for each receptive field. The linear units generate a prediction that is weighted by the weight w_k from the Gaussian weighting unit. The sum of all weighted predictions ($\hat{y}_k$) is the total system's output ($\hat{y}$).

Assuming, that the parameters $\mathbf{c}_k$ (center of receptive field k), $\mathbf{M}_k$ (shape of the receptive field k), and $\hat{\mathbf{a}}_k$ (regression parameter of the linear model) are known, the model predicts an output y for every given input **x**. These parameters are independent for every model, so every model predicts independently of each other. The important question how these parameters can be calculated will be discussed in the following sections.

4.5 Learning Parameters

4.5.1 General Remarks

This chapter concentrates on learning the parameters needed for predicting an output at a given query point. As shown in the previous chapter, these parameters are: the center of a receptive field ($\mathbf{c}_k$), the shape of the distance metric ($\mathbf{M}_k$), and the regression coefficients ($\hat{\mathbf{a}}_k$). For clarity, the subscript k will be dropped in the following description, as all receptive fields are updated in the same way.

I my explanation, I will first concentrate on updating the linear model in each receptive field (chapter 4.5.2), and then discuss updating the distance metric of a receptive field (chapter 4.5.3). The focus of the remaining two chapters, 4.5.4 and 4.5.5, will be the question of updating the number of receptive fields, i.e. adding and pruning these fields. Adding receptive fields also discusses finding a proper center in input space for each field, because once a receptive field is generated, its center is not moved anymore.

4.5.2 Learning the Local Linear Model

Learning the linear regression coefficients is by definition a linear problem and thus straightforward. The process is easier to understand if we first leave out the incremental learning and try to estimate in a batch update. In a second step, the algorithm will then be modified to suit an incremental version. To start the batch process, we have to summarize all input vectors in a matrix **X** and all outputs in a vector **y**.

$$\mathbf{X} = \left(\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_3, \ldots, \tilde{\mathbf{x}}_M\right)^T \qquad \text{(eq. 4.8)}$$

$$\mathbf{y} = (y_1, y_2, y_3, \ldots, y_M)^T \qquad \text{(eq. 4.9)}$$

Remember that $\tilde{\mathbf{x}}$ denotes the center subtracted projection of the input data **x** (equation 4.6). The weights are arranged in a diagonal weight Matrix **W**:

$$\mathbf{W} = \begin{pmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & w_M \end{pmatrix} \qquad \text{(eq. 4.10)}$$

Next, the parameter can be estimated with a standard weighted regression technique that is widely used in non-parametric statistics (e.g. Cleveland, 1979, Cleveland & Loader, 1995), in time series prediction (e.g. Farmer & Sidorowich, 1987, 1988), and in regression learning problems (Atkeson, 1989, Moore, 1991, Schaal & Atkeson, 1994, Atkeson, Schaal et al. 1995).

$$\hat{\mathbf{a}} = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{Y} = \mathbf{P} \mathbf{X}^T \mathbf{W} \mathbf{Y} \qquad \text{(eq. 4.11)}$$

$$\text{with } \mathbf{P} = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \qquad \text{(eq. 4.12)}$$

These simple formulae will allow a calculation of the linear regression coefficients under the assumption that all data is available in **X** and **y**, and that the weights **W** are known. The weights w_k are given in equation 4.3, but the representation of the data in **X** and **y** has to be changed for an incremental learning approach, before using ***LWPR*** to predict new outputs.

To modify the regression in equation 4.11 towards in incremental update, one can make an astonishing observation: the result for equation 4.11. is exactly the same as if is calculated from one swoop through the whole training set by only using one data point at a time and without looking back at the other training data. Using recursive least squares as update rules (Ljung & Söderström, 1986), one obtains the following formulae for the incremental version:

$$\hat{\mathbf{a}}^{n+1} = \hat{\mathbf{a}}^{n} + \mathrm{w}\mathbf{P}^{n+1}\tilde{\mathbf{x}}\mathbf{e}_{\mathrm{cv}}^{\mathrm{T}} \qquad \text{(eq. 4.13)}$$

$$\mathbf{P}^{n+1} = \frac{1}{\ddot{e}}\left(\mathbf{P}^{n} - \frac{\mathbf{P}^{n}\tilde{\mathbf{x}}\tilde{\mathbf{x}}^{T}\mathbf{P}^{n}}{\frac{\ddot{e}}{\mathrm{w}} + \tilde{\mathbf{x}}^{T}\mathbf{P}^{n}\tilde{\mathbf{x}}}\right) \qquad \text{(eq. 4.14)}$$

$$\mathbf{e}_{\mathrm{cv}} = (\mathbf{y} - \hat{\mathbf{a}}^{n\,\mathrm{T}}\tilde{\mathbf{x}}) \qquad \text{(eq. 4.15)}$$

The initialization of **P**, which corresponds to the inverted covariance matrix, will be discussed in chapter 4.5.4. Equation 4.14 contains an additional parameter: the forgetting factor $\ddot{e}$, $\ddot{e} \in [0,1]$. This factor describes the reliability of old data, and is one of the most significant differences between batch and incremental updates. Imagine, that a robot's mechanical properties change during its life, such that old data does not represent the new mechanical properties anymore. Then, new data should be trusted more than old data. In the case of ***LWPR***, this forgetting factor is especially important, since the distance metric **M** changes during the learning process. The update of **M** will be discussed in the following chapter. The result is that old data was learned with inappropriate weights and that this data will be misinterpreted for later queries. So helps to gradually cancel the contribution from early data points when **M** was not yet learned properly (e.g. Ljung & Söderström, 1986).

Equations 4.13 - 4.15 offer an incremental update rule for the local linear models, which satisfies the introduced constraint that the update should happen incrementally without the need to store old data. Additional properties of these formulae are that they

- are guaranteed to converge to the global minimum of a weighted squared error criterion (Atkeson, Moore, et al., 1997)
- avoid the matrix inversion (equation 4.12).

This update rule is implemented in ***LWPR***, as will be seen in chapter 4.6, where the final algorithm is described.

4.5.3 Learning the Size and Shape of Receptive Fields

Updating the distance metric **D** needs to be done by updating the decomposed upper triangular distance metric **M**, where $\mathbf{D} = \mathbf{M}^{\mathrm{T}}\mathbf{M}$ (equation 4.4) in order to ensure the positive definiteness of **D**. The first idea might be that this can be done by using a gradient descend technique with weighted mean-squared-error criterion, which is also the basis of the locally weighted regression in equation 4.11:

$$J = \frac{1}{W}\sum_{i=1}^{p} w_i \left\|\mathbf{y}_i - \hat{\mathbf{y}}_i\right\|^2 \qquad \text{(eq. 4.16)}$$

$$\text{with } W = \sum_{i=1}^{p} w_i \qquad \text{(eq. 4.17)}$$

This is, however, a very poor approach. It can easily be imagined that every input point is modeled by exactly one receptive field, which has its center exactly at this point. Then the distance metric can be very small and the model will predict this point with no error. But it will only predict this single point, and there will be no generalization to capture the essentials of the data generating function. This version has no error for known points, but it will encounter strong over-fitting and, thus, not be able to predict unseen data accurately. For this reason, a global competition among receptive fields has been introduced (e.g. Moody & Darken, 1988, Jordan & Jacobs, 1994). The global competitive process will not allow narrow distance matrices, since all receptive fields start competing against each other for data points, at least if there are more points than receptive fields. But the introduction of global competition among the receptive fields removes the ‘local learning’ character from the system.

An alternative way to address this issue is to use leave-one-out cross validation: The testing of data i is performed in a system that has been trained on all the data without the single data i. This will lead to a new cost function:

$$J = \frac{1}{W}\sum_{i=1}^{p} w_i \left\| \mathbf{y}_i - \hat{\mathbf{y}}_{i,-i} \right\|^2 \qquad \text{(eq. 4.18)}$$

with $\hat{\mathbf{y}}_{i,-i}$ meaning that the prediction for point i is calculated from a net trained without point i as input data. The major disadvantage of this approach is its computational expense, as a new system has to be trained for every point of data. This will result in a p-fold that has to be run through for p points in the training set. Another partially unresolved problem is the question of how the p different systems should be merged into one after learning.

For linear regression problems, however, the Sherman-Morrison-Woodbury theorem, also known as the 'matrix inversion theorem', (e.g. Belsey, Kuh, et al., 1980) allows a simplification of the expression in terms of computational complexity:

$$J = \frac{1}{W}\sum_{i=1}^{p} w_i \left\| \mathbf{y}_i - \hat{\mathbf{y}}_{i,-i} \right\|^2 = \frac{1}{W}\sum_{i=1}^{p} \frac{w_i \left\| \mathbf{y}_i - \hat{\mathbf{y}}_i \right\|^2}{(1 - w_i \tilde{\mathbf{x}}_i \mathbf{P} \tilde{\mathbf{x}}_i)^2} \qquad \text{(eq. 4.19)}$$

Instead of using the computationally expensive leave-one-out cross validation, the cost criterion can be computed weighting the sum by the mean squared error calculated using the inverted covariance matrix $\mathbf{P} = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1}$ (equation 4.12.). This means that a criterion has been found, which can be used to adjust the distance metric M in a computationally affordable way.

But there is still one more problem: With an incrementally increasing number of training data samples, more and more receptive fields will be generated. So the size of each receptive field will shrink to small values, which again reduces the ability to generalize. To avoid this behavior, an additional penalty term is introduced in equation 4.19, yielding in equation 4.20:

$$J = \frac{1}{W}\sum_{i=1}^{p} \frac{w_i \left\| \mathbf{y}_i - \hat{\mathbf{y}}_i \right\|^2}{(1 - w_i \tilde{\mathbf{x}}_i \mathbf{P} \tilde{\mathbf{x}}_i)^2} + g \sum_{i,j=1}^{n} \mathbf{D}_{ij}^2 \qquad \text{(eq. 4.20)}$$

where the scalar g determines the strength of the penalty. The additional term penalizes the second derivative of the distance metric, and therefore an abrupt decay of the receptive field's region of validity.

Equation 4.20. is an appropriate function to use as cost criterion for changing the distance metric with a gradient descend method in a batch update. To modify the update towards an incremental version, a learning rate is introduced:

$$\mathbf{M}^{n+1} = \mathbf{M}^{n} - a \cdot \frac{\partial J}{\partial \mathbf{M}} \qquad \text{(eq. 4.21)}$$

The derivative of the cost function J with respect to the distance metric **M** can be found in Schaal & Atkeson, 1998. The algorithm ***LWPR*** uses an approximation of this result, which can be found in chapter 4.6 included in the finial algorithm.

4.5.4 Adding Receptive Fields

As ***LWPR*** is designed to perform incremental online learning, it will be necessary to add receptive fields for such new data, which is living in previously unknown input domains. The way ***LWPR*** decides whether a new receptive field has to be created is to monitor the activation of all existing receptive fields for new data. If the new data point **x** does not activate any receptive field by more than a threshold ($w_i < w_{gen}$ for all i), then a new receptive field with the following properties is created:

- The center of the new receptive field is at the position of the data sample in input space ($\mathbf{c} = \mathbf{x}$).
- The distance metric **M** is set to a manual default value $\mathbf{M}_{def}$. This default value has to be chosen with attention: It should rather span the field too wide (resulting in small numbers for M) than to narrow. Because a receptive field that spans a wide domain in input space will be activated by many subsequent points of new input data, it will quickly shrink to a roughly appropriate size. In the other case, a

receptive field having a narrow valid region will need more time to find data that allows it to increase its region of validity.

- All remaining parameters are initialized to zero.

Only the matrix **P** needs special treatment, since P is used as a 'memory' in estimating the linear model. However, a new point does not have any memory. A reasonable initialization is to create a diagonal matrix where the elements on the diagonal are set to $p_{ii} = {}^{1}\!/_{r_i^2}$, with r having small quantities ($r_{\mathrm{i}} \approx 0.001$ for all i) (Ljung & Söderström, 1986). This initialization can be interpreted in two different ways. Firstly from a probabilistic point of view: then, these **r** are priors that the coefficients of $\hat{\mathbf{a}}$ are zero. Secondly, from an algorithmic perspective: then they are fake data points of the form $[\mathbf{x}_r = (0,0,...,0,r_i^2,0,...,0), \mathbf{y}_r = 0]$ (Atkeson, Moore, et al., 1997). Under normal circumstances, the size of the coefficients of r is too small to introduce noticeable bias.

4.5.5 Pruning Receptive Fields

Two different criteria cause pruning of receptive fields: Overlap with other fields and an excessively large mean-squared-error compared to other fields.

In the first case, the overlap of two receptive fields can be detected when new training samples are added. If a data point activates two fields simultaneously with more than a constant w_{prune}, the field with the larger determinant of the distance metric **D** (i.e. the one with the smaller receptive field) is pruned. For computational convenience, det(**D**) can be approximated by $\sum_{\mathrm{j}} \mathrm{D}_{\mathrm{jj}}^2$ (Deco & Obradovic, 1996). However, pruning due to overlap is only done to gain better performance in terms of computational complexity. For the accuracy of the prediction, it does not make a difference, if two fields are overlapping (see chapter 4.2.3).

The other cause for pruning is given if the bias-adjusted weighted mean-squared-error

$$wMSE = \frac{E^n}{W^n} - g \sum_{i,j=1}^{n} D_{ij}^2 \qquad \text{(eq. 4.22)}$$

of a linear model happens to become excessively large in comparison to other units. Testing the size is necessary, because also $\mathbf{M} = \mathbf{0}$ will result in a minimum for the optimization criterion, given in equation 4.20. Setting M to the zero matrix is equivalent to performing global instead of local regression, using an infinite receptive field. This global regression has a large wMSE and, thus, the receptive field will be pruned and a local approach will be pursued.

In general, pruning rarely happens. If it happens, it is usually due to an inappropriate initialization, so it only occurs at the beginning of learning when pruning does not cause big harm.

4.6 The Final *LWPR* Algorithm

We will now move on to provide a more complete description of the algorithm ***LWPR***, and will merge all the different functional blocks into a single algorithm in pseudo-code.

Table 4.2 shows an incremental update of the linear regression parameters (from chapter 4.5.2) and the shape of the distance metric (from chapter 4.5.3) using projection techniques on the input data. The variables SS, SR, and SZ are memory terms that allow the univariate regression in step f) to happen in a recursive least squares fashion. All recursive variables in table 4.2, i.e. those with the superscript n, will be initialized to zero before the algorithm starts. Step g) regresses the projection $\mathbf{p}_i$ from the current projected data s and the current input data **z**. This step guarantees, that the next projection of the input data for the consecutive regression will result in a $\mathbf{u}_{i+1}$ that is orthogonal to $\mathbf{u}_i$. Step j) is recursively keeping track of the mean-squared-error in consecutive projections. It is used to decide when to add a new projection direction, as described in chapter 4.5.4.

```
***Given :*** A training point ( $\mathbf{x}, y$)
***Update the means of inputs and output:***
$\mathbf{x}_0^{n+1} = \frac{lW^n \mathbf{x}_0^n + w\,\mathbf{x}}{W^{n+1}}$
$b_0^{n+1} = \frac{lW^n b_0^{n+1} + w\,y}{W^{n+1}}$
where $W^{n+1} = lW^n + w$

    ***Update the local model:***
    Initialize: $\mathbf{z} = \mathbf{x}, res = y - b_0^{n+1}$
    For i = 1: r,
    a)  $\mathbf{u}_i^{n+1} = l\,\mathbf{u}_i^n + w\,\mathbf{z}\,res$
    b)  $s = \mathbf{z}^T \mathbf{u}_i^{n+1}$
    c)  $SS_i^{n+1} = l\,SS_i^n + w\,s^2$
    d)  $SR_i^{n+1} = l\,SR_i^n + w\,s\,res$
    e)  $SZ_i^{n+1} = l\,SZ_i^n + w\,\mathbf{z}\,s$
    f)  $b_i^{n+1} = SR_i^{n+1} / SS_i^{n+1}$
    g)  $\mathbf{p}_i^{n+1} = SZ_i^{n+1} / SS_i^{n+1}$
    h)  $\mathbf{z} \leftarrow \mathbf{z} - s\mathbf{p}_i^{n+1}$
    i)  $res \leftarrow res - s b_i^{n+1}$
    j)  $MSE_i^{n+1} = l\,MSE_i^n + w\,res^2$
```

Table 4.2. Update Rule for every Receptive Field

Finally, the following table shows a pseudo-code implementation of ***LWPR***'s main loop that has to be processed for new data. All the elements used are introduced and discussed in the previous chapters.

```
Locally Weighted Projection Regression (LWPR) Pseudo-Code
INITIALIZE THE SYSTEM WITH NO RECEPTIVE FIELD (RF)
FOR EVERY NEW TRAINING SAMPLE ($\mathbf{x}$, Y):
    FOR I=1 TO #RF
        CALCULATE THE ACTIVATION FROM EQUATION 4.3
        UPDATE THE RF ACCORDING TO TABLE 4.2
    IF NO LINEAR MODEL WAS ACTIVATED BY MORE THAN $w_{\text{gen}}$
        CREATE A NEW RF ACCORDING TO CHAPTER 4.5.4
```

Table 4.3. Pseudo-Code of the ***LWPR*** algorithm

4.7 Properties of *LWPR*

The learning algorithm ***LWPR*** is capable of approximating unknown functions between incoming streams of input and output data in high dimensional spaces.

Its core is formed by local linear models, which span a small number of univariate regressions in selected directions in input space. The spatial localization of the linear models offers robustness against negative interference. The parameters of these models, i.e. the regression coefficients $\hat{\mathbf{a}}$ and the region of validity given by the distance metric **D**, are learned automatically. The learning algorithm only uses data from the local neighborhood of the model, not interacting with other models. Competition between local models is thus not necessary, and the local information processing approach remains valid. The major strength of ***LWPR*** is that it uses incremental learning techniques, such that data is discarded after learning and does not need to be stored. This allows continuous learning even if the system is running for a very long time, eventually infinitely. The algorithm can deal with a large number of possibly redundant and / or irrelevant input dimensions, while its projection-technique PLS automatically determines the appropriate number and direction for needed projections. ***LWPR***'s computational complexity is linear in the number of inputs and linear in the number of projected input dimensions (O(n·r)). This makes ***LWPR*** very attractive from a computational point of view.

To my knowledge, ***LWPR*** is the first incremental neural-network learning algorithm that combines all these properties. It is thus very well suited for the high-dimensional real-time learning problems posed by humanoid robots.

5. Evaluation of *LWPR*'s Performance on Artificial Data

5.1 Introduction

I would now like to start a first evaluation of ***LWPR***'s performance. Therefore, a set of data artificially generated from a known function is learned. The algorithm only knows the data, it does not know about the function itself. As has been explained in chapter 3.1, the success of the learning can be measured by the fact how well the function is approximated.

For the evaluation, I have used two functions, which are described in more detail in chapters 5.2 and 5.3. They are composed of Gaussians and sinusoidals, such that no abrupt changes, like steps or spikes, occur. The underlying assumption is that the inverse dynamics that is going to be modeled will also be a smooth function without abrupt changes. Both functions used for evaluation are designed to demonstrate special properties of ***LWPR***, like predictions in almost linear regions or rapidly changing non-linear data.

The values of these functions at randomized positions within a limited interval are used to generate a noise-contaminated set of training data. After learning, a regularly spaced vector of query points is generated and the trained system is asked to predict the function at these points. These predictions are used to reconstruct the function and to estimate the quality of the regression against the original function. The generated training set does not contain any of the query points for function reconstruction: The network might simply have memorized all these points, and thus the estimation of the total error would be influenced. The following two chapters will discuss one of the two training functions each.

5.2 One Dimensional Function Fitting

The first function used for evaluation consists of three separate areas in the total range of $-10 < x < +10$: A linear region from –10 to –2 with a narrow Gaussian bump superposed at $x = -6$ followed by the left half of a steep Gaussian centered at $x = 0$. For $x > 0$, the function is given by a si-function:

$$y = \begin{cases} 0.1 \cdot x + 0.4 \cdot \exp\left(-2 \cdot (x+6)^2\right) & ; -\infty < x \leq -2 \\ 0.1 \cdot x \cdot \sin(-\frac{x \cdot p}{4}) + \exp(-x^2) & ; -2 < x < 0 \\ \frac{\sin(x)}{x} & ; 0 \leq x < +\infty \end{cases} \quad \text{(eq. 5.1)}$$

Equation 5.1 is used to generate two distinct sets: a set of training and a set of testing samples. Additionally, all data in the training data set contain additional noise generated by $N(0,0.01)$.[15] The training set consists of 500 noisy points drawn uniformly from random positions in the range $-10 < x < +10$. The test data set consists of 200 testing points without noise equally spaced in the same interval. The predictions of these training points are used to reconstruct the function after learning. No data point belonging to the test set is contained in the training set. ***LWPR*** is initialized using the following values: The default distance metric $\mathbf{D}_{def} = 8 \cdot \mathbf{I}$,[16] the threshold for generating a new receptive field $w_{gen} = 0.1$, the threshold for pruning two overlapping fields $w_{prune} = 0.9$, and, finally, the penalty for narrow distance matrices is set to $g = 0.0001$ (equation 4.20).[17]

Figure 5.1a shows the original function without noise and the network prediction after learning. Figure 5.1c shows the error of the learned function, i.e. the original function subtracted by the network prediction: $y = y_{original} - y_{lwpr}$. This allows recognizing the difference easily.

[15] N(0, 0.01) denotes a Gaussian noise distribution with mean zero and a variance of 0.01
[16] I being the identity matrix; the size of I appropriate to the context
[17] For a discussion, see Schaal & Atkeson, 1998.

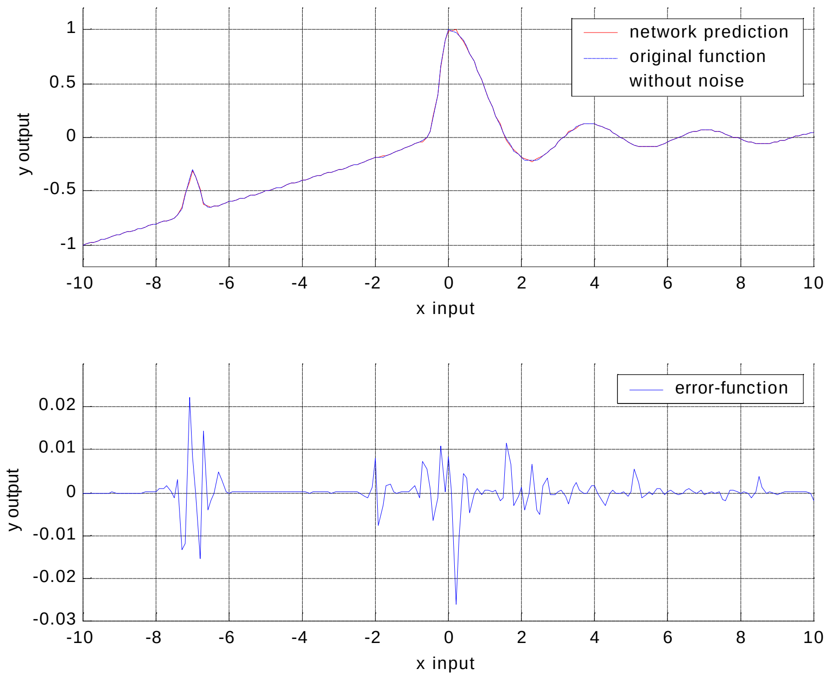

Figure 5.1 Results of Learning a One Dimensional Function

Looking at the error plot in figure 5.1b, one can easily see that the prediction is almost perfect in the linear range of the original function ($x < -8$ and $-6 < x < -2$). The error at $x = -2$ might seem odd; but looking at the function y in equation 5.1, one can see that the function changes at $x = -2$ which causes small perturbations that the algorithm cannot resolve completely. Instead, ***LWPR*** uses the linear function regression as the best chance and assumes that the real data is noise-contaminated. Of course, the error-plot uses the real function and shows these linear predictions as errors in the regression.

The following figure 5.2a shows the learning curve: the decrease of the normalized mean squared error (nMSE[18]) over time on a double logarithmic scale. The nMSE slowly decreases from the initial value until a sharp drop occurs. This is a significant property of learning approaches, which usually happens after roughly one swoop through the training set. Finally, the nMSE settles at an excellent value of roughly $nMSE = 3 \cdot 10^{-5}$

[18] nMSE denotes the mean squared error (MSE) on the test set normalized by the variance of the outputs of the test set. The nMSE can be used for simple comparisons of different regression problems.

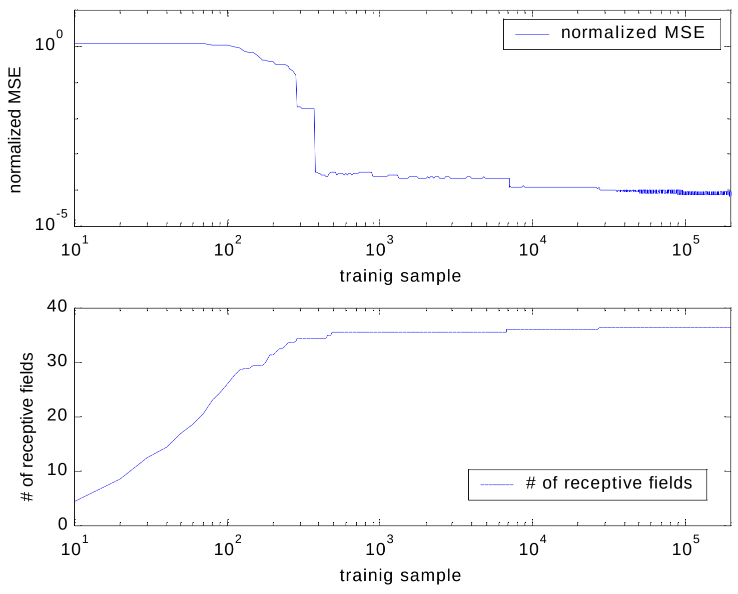

Figure 5.2 Learning Curves for the One Dimensional Regression

Figure 5.2b shows that the number of receptive fields allocated by ***LWPR*** continually grows during training. Ultimately, it settles with #rf=37. One can see that the nMSE decreases by a significant step whenever a new receptive field is added. This happens, because a new receptive field will only be introduced at such positions that have not been modeled well by the existent receptive fields. The new field will model the data with high accuracy exactly at that position, such that the nMSE decreases.

An interesting experiment can be performed forcing the algorithm to use a smaller number of receptive fields. This can be achieved by increasing the penalty for narrow distance matrices, e.g. by keeping all parameters the same and only increasing g by a factor of 100, such that $g_{\text{new}} = 0.01$. Figure 5.3 illustrates the results: Parts a and c show that the function regression becomes significantly worse (note the changed y-axis scale in figure 5.3c!), when using only a total of 23 receptive fields compared to 37 as before. This is not surprising, as a smaller number of linear models will always perform worse in modeling a non-linear function. However, this experiment can be used to display the position and shape of the receptive fields. In figure 5.3b, one can see that a small number of fields spawn a wide range in those areas where the function is almost linear. This means, that good approximation can be achieved in a wide region with very little effort. On the other hand, ***LWPR*** places many receptive fields with a small region of validity where the non-linearity is significant, e.g. at $x = 0$ or $x = -7$. Hence, the function needs a big number of linear approximations to be reasonably represented. With this experiment, one can understand that the learning algorithm adjusts the position and shape (i.e. the width) of the receptive fields to resemble the data distribution appropriately. Hence, the computational complexity (that is correlated to the number of receptive fields) is adapted to the input data distribution.

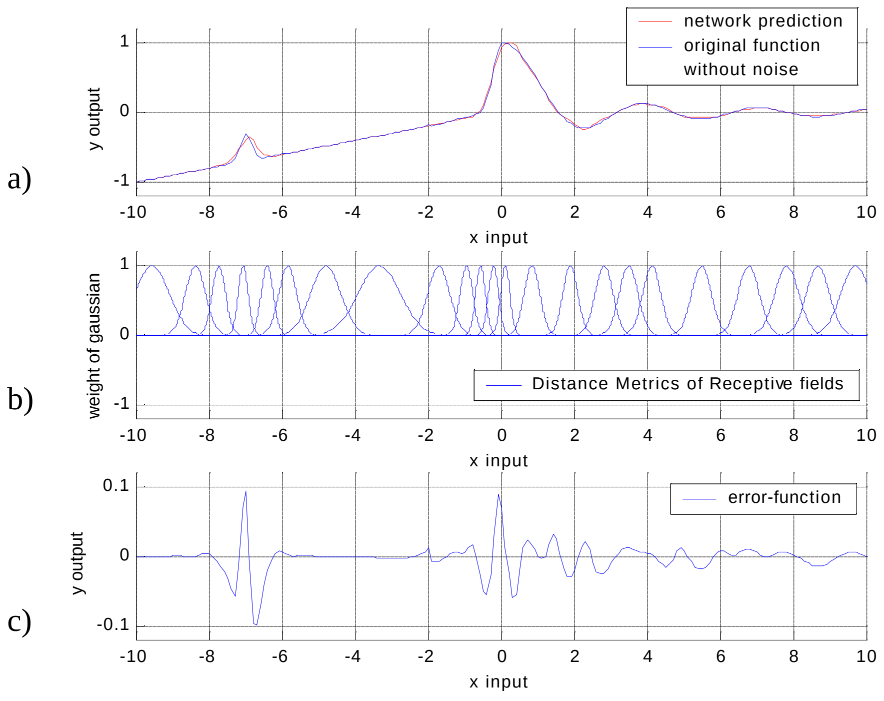

Figure 5.3 Learning by using a Constrained Number of Receptive Fields

5.3 Two-Dimensional Function Fitting with Shifting Input Distributions

A more complex function that is still relatively easy to display is given by equation 5.2. It is a two-dimensional function (along the directions x_1 and x_2), which consists of three Gaussian bumps in the rear and a slowly rising but quickly falling Gaussian slope in the foreground:

$$y_1 = \tfrac{1}{2} \cdot \exp\left(-15 \cdot \left((x_1 + \tfrac{1}{2})^2 + (x_2 - \tfrac{1}{2})^2\right)\right) \quad \text{(eq. 5.2)}$$

$$y_2 = \tfrac{2}{2} \cdot \exp\left(-25 \cdot \left(x_1{}^2 + (x_2 - \tfrac{1}{2})^2\right)\right) \quad \text{(eq. 5.3)}$$

$$y_3 = \tfrac{3}{2} \cdot \exp\left(-35 \cdot \left((x_1 - \tfrac{1}{2})^2 + (x_2 - \tfrac{1}{2})^2\right)\right) \quad \text{(eq. 5.4)}$$

$$\left.\begin{aligned} y_4 = {} & \exp\left(-\ 2 \cdot (x_1 - \tfrac{1}{2})^2\right) \cdot \left(1 + \exp(2 \cdot x_2)\right)^{-1} \\ & + \tfrac{1}{2} \cdot \exp\left(-25 \cdot (x_1 + \tfrac{1}{2})^2\right) \cdot \exp\left(-10 \cdot (x_2 + 1)^2\right) \end{aligned}\right\} \text{ for } x_1 \le \tfrac{1}{2} \quad \text{(eq. 5.5)}$$

$$y_4 = \exp\left(-50 \cdot (x_1 - \tfrac{1}{2})^2\right) \cdot \left(1 + \exp(2 \cdot x_2)\right)^{-1} \quad \text{for } x_1 > \tfrac{1}{2} \quad \text{(eq. 5.6)}$$

$$y = \max(y_1, y_2, y_3, y_4) \quad \text{(eq. 5.7)}$$

The Gaussian bumps in the rear (equations 5.2-5.4) have decreasing width and increasing height from the left to the right. This implies very different slopes, which need to be modeled with linear patches of different sizes. The foreground consists of a slowly rising Gaussian slope with a small Gaussian bump superposed (equation 5.5). Once a threshold is passed ($x_1 = \tfrac{1}{2}$), the function rapidly returns to zero (equation 5.6). A 3-D plot of the function can be seen in fig 5.2a.

In contrast to the 1-D-function in chapter 5.2, this time the training data is drawn from two separate regions of the input space. The first training set consists of data from the region $x_2 \le 0$, whereas the second training set covers the other half of the input space ($x_2 > 0$). Each of these training sets consists of 5000 noisy data samples drawn uniformly over half the unit square. During learning, only one training set is used at a time: At first, the algorithm is trained on the set from the back with the three Gaussian bumps. After learning finished (based on an evaluation using a temporary test data set from the same region), the input data is switched and only the second training set from the front is used to continue learning. This time, test data drawn from all over the unit square is used. The algorithm is, however, not re-trained on data from the back anymore, i.e. it has to remember what was learned before.

The final test data set consists of 4489 testing points without noise, meaning that the output values of the test data are the exact function values. These points correspond to the vertices of a 67x67 grid equally spaced over the unit square that will also be used to display the function. The initial parameters for ***LWPR*** are $\mathbf{D}_{def} = 16 \cdot \mathbf{I}$, $w_{gen} = 0.1$, $w_{prune} = 0.9$, and $g = 0.0001$ - so only the initial size of the distance metric is changed.

For learning this two dimensional function, the smaller distance metric still is wide enough, because with a second dimension, more training points fall into every receptive field's active area.

To evaluate the performance of ***LWPR***, the network prediction and the error plot are shown in figures 5.2b and 5.2c:

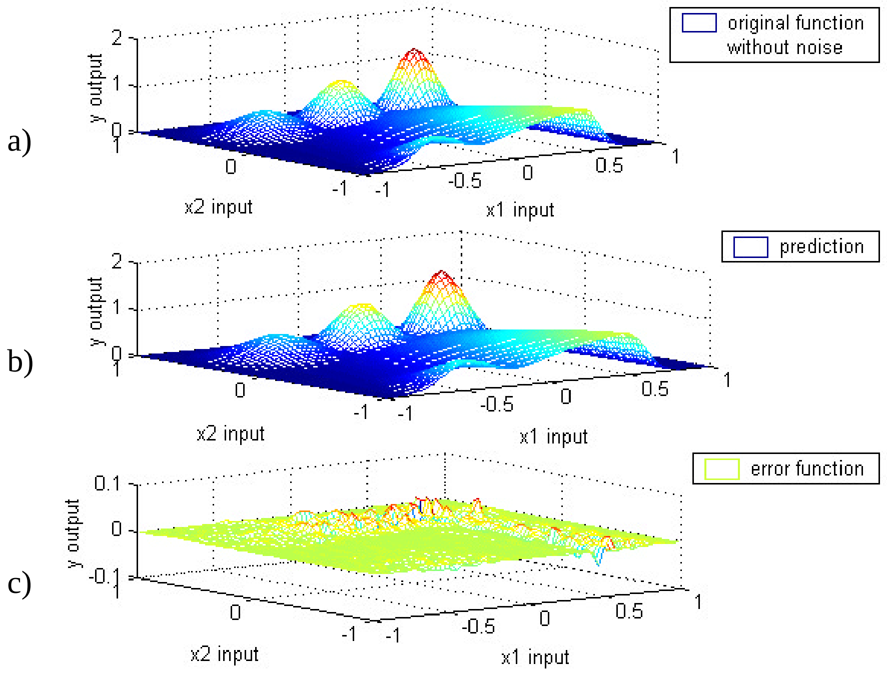

Figure 5.4 Function Approximation for the 2D Example

One can easily see that almost linear regions (like the Gaussian with the small slope in the left corner) result in very accurate predictions. In contrast, regions that cannot be regressed easily by linear functions cause a visible error (like the abrupt fall of the Gaussian in the right corner or the high Gaussian in the rear). It is a remarkable result that although training was split in two different regions, no interference is visible: There is no sharp edge or a significant misclassification in the back. So local learning seems to be robust against shifting input distributions and does not forget previously learned domains while training on new domains.

The maximal error in approximating the function is 0.0606, which is equal to 4% based on the total amplitude of the function. The average error in approximating over all testing points is a much better 0.003, which equals to an excellent 0.2% based on the total amplitude. So using a moderate number of linear models (roughly 400), the algorithm achieves a good approximation of a complex function (cf. equation 5.2-7).

5.4 Limiting the Computational Complexity

There is always a tradeoff between the number of receptive fields allocated by ***LWPR*** and the quality of the prediction (as shown in chapter 5.2). The number of receptive fields is proportional to the computational speed during training. Additionally, the number of receptive fields is proportional to the time needed for generating a prediction. If the later two were not related, an infinite number of locally valid linear models could be used to model arbitrarily complex non-linear functions. So if twice the number of receptive fields were used for the regression, the result would be much better; but also the algorithm would need twice as much time to calculate predictions for given query points.

Luckily, there is a way to deal with this unsatisfying issue when working in motor control. Under the assumptions that the robot behaves well, the query points will not move arbitrarily within the input space, and thus, that consecutive query points will be lying 'close' together. Therefore, we may assume that for little time steps the desired state of the robot only changes slightly. Due to this reasoning, all active receptive fields must be somewhere close to those fields that were active in the previous time step. Using an intelligent structure, the closest neighbors of previously active receptive fields can be found easily. Asking only these fields for predictions on the current query point will limit the computational time needed for a new prediction. As the number of closest receptive fields (n) can be specified by the user, the computational time needed for a prediction will never exceed a user specified time, no matter how many receptive fields have been learned by ***LWPR***. For very fast motion or a small number n, however, one has to be careful not to reduce the quality of the prediction as more receptive fields might be actively contributing to the current prediction than assumed.

5.5 Discussion

The examples of learning artificial data show that truly local learning is a feasible approach that can compete with other state-of-the-art learning algorithms. In this context, it has to be remembered that local learning means learning without competition, without gating nets, and without global regression on top of local receptive fields. Additionally, the algorithm learns in a truly incremental fashion: It can learn from continuously incoming data without knowing the distribution of the input data. Such a carefully designed local learning is robust against negative interference even without having prior assumptions on the input data distribution.

The development of this algorithm followed the assumption that the complexity of the input data is unknown, i.e. that the original data generating function can be of a very complex type. If there was more knowledge about the data generation model, this knowledge should be incorporated in the algorithm design. E.g., regressing a sinusoidal function with a sinusoid is the best approach under the circumstances; no other learning algorithm will outperform this 'regression'. However, in the case of a linear dynamics model for a humanoid robot, prior knowledge about the generation of the data is not available. Therefore, as little assumptions as possible have been made about the data. The only prior assumption that is introduced for the learning is hidden in the penalty term g for small distance metrics. This term introduces an assumption on the smoothness of the function that is going to be regressed.

There are several open research issues connected to the learning approach, e.g. estimating the size of the learning rate a. Another open question is how a good initialization of the distance metric $\mathbf{D}_{def}$ might be found. The initial distance metric, which describes the size of newly created receptive fields, should be appropriate for the data being modeled. Too large receptive fields face the danger to grow until they span the entire input domain, too small metrics tend to over-fit the data and reduce the generalization ability. All these parameters can easily be monitored using just a small number of receptive fields during an initial training phase. As all fields learn independently of the others, the experience gained by training only a few can be used to adjust parameters for all fields.

Further, one might wonder how far such a local learning system has any parallels with neurobiological information processing. The human brain uses locally weighted receptive fields for information processing. However, one major assumption about learning in the brain based on receptive fields is that these fields are broadly tuned and widely overlapping (e.g. Merzenich, Kaas, et al., 1983). Thus, the accuracy of the prediction is achieved by calculating the average of many fields with a reasonable error each. If you compare this with ***LWPR***, you first note that ***LWPR*** rather achieves accuracy by every single or only a few overlapping fields. Whether the learning principles of ***LWPR*** are biologically relevant or not remains speculative. In either way, ***LWPR*** demonstrates that there is an alternative and powerful method to accomplish incremental constructive learning based on local receptive fields.

Using ***LWPR*** for learning problems in the context of robot motion control might be a promising way to gain better performance compared to traditional approaches. The motion data generated with respect to the inverse dynamics problem involves many of the issues that ***LWPR*** can easily deal with. This is especially true for coping with incremental data and changing input distributions as was shown in this thesis. Further examples of evaluating ***LWPR***'s performance can be found in Schaal & Atkeson, 1998. The authors' tests include dealing with redundant or irrelevant input signals and highly noisy data.

Next, I would like to present the application of LWPR on a real robot. The inverse dynamics model used for computed torque feed-forward commands will be learned on generated motion. But first, it might be useful to give a short description of the more technical details of the robot setup.

6. The Humanoid Robot

For testing the algorithm on real data, it is important to have a system that represents ‘human’ properties, i.e. redundancy, flexibility, and the ability of fast motion. As for the test itself, it does not matter whether a robotic arm, a leg, or the whole robot is used for estimating the inverse dynamics, as the concept remains the same. I worked with a robotic arm. This offers a number of advantages: it allows evaluating the performance on a system that has a high number of degrees of freedom and therefore redundancy in the motion. Additionally, there is a significant influence from the motion of one joint to another. A single leg would, in comparison, get on with a smaller number of degrees of freedom and much smaller influence of one joint's motion to another.

The arm of a humanoid robot does not comply with the rigid body dynamics assumptions, such that the rigid body framework cannot be used to generate feed-forward commands. These are needed, as the PD controller shall run on low gains (see chapter 2.7). Therefore, the learning algorithm ***LWPR*** that was described in chapters 4 and 5 will be used to learn the inverse dynamics control. But first, the robot system used will be introduced.

6.1 Hardware for Robot Control

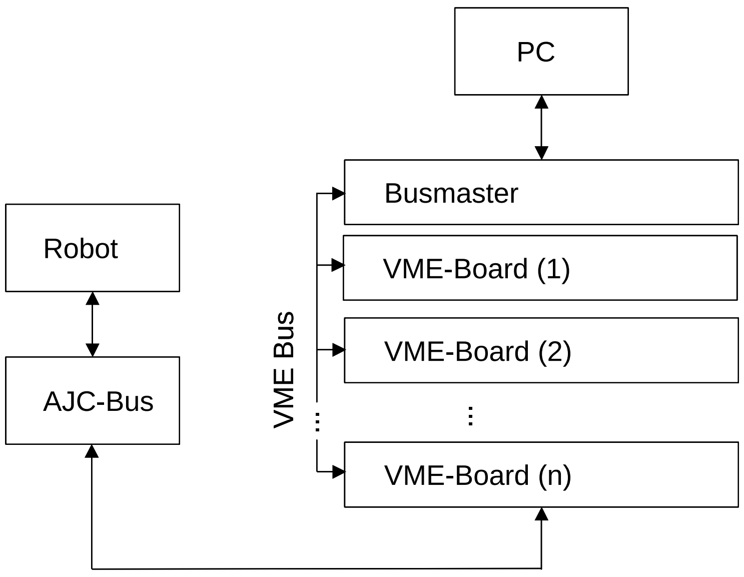

Figure 6.1 Sketch of the Robot Hardware Environment

The vxWorks programming environment from Windriver Systems offers a UNIX-like real time operating system on the target computers (the VME-Boards) and a development environment on a host computer. Figure 6.1 shows the connections in the system. The host computer can address multiple targets and offers target communication from the host to a target or between targets. This also includes shared memory and message passing. Real time linking is allowed, such that objects can dynamically be added or removed from any of the target computers without reloading the whole program. Additionally, the real-time operating system running on the targets offers frequently used features, such as low-level I/O, memory management, process communication, task scheduling, interrupt handling, and many more.

The fact that the VME-Boards are connected to a host computer allows simple integration into an existing TCP/IP computer network and at the same time to keep real time functionalities. The host interface Tornado allows user interactions with the running system. Still, it keeps as much workload as possible on the host computer, freeing the targets to work in real time on their duty, the control of the robot. The host computer is additionally used for developing programs that can run as tasks on the targets. Programs written in C can easily be cross-compiled on a Unix system and downloaded (i.e. linked) to any of the targets for execution.

The targets are Motorola MVME2700 boards each having a single Power PC processor running at 360 MHz. All the targets are mounted in a single VME rack, with only the bus master having a connection to the host computer. The bus master also manages the shared memory and the initialization of all other boards in the systen. The communication between the different targets is performed via the VME backplane. All VME boards run tasks with certain functionality, called Servos, as shown in figure 6.2:

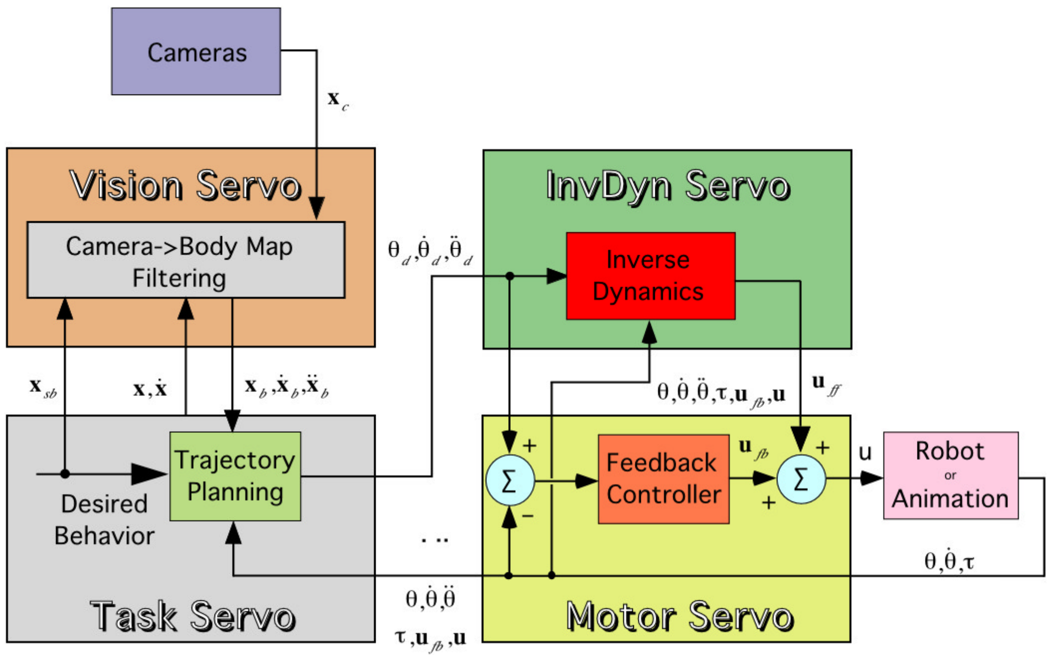

Figure 6.2 General Functionality of the different VME Boards

The Task Servo handles the user interface and plans the desired trajectory in joint space, resulting in $\grave{e}_{\text{des}}$, $\dot{\grave{e}}_{\text{des}}$, and $\ddot{\grave{e}}_{\text{des}}$. The Vision Servo and additional external cameras are not used in this thesis, as the desired trajectory is analytically generated (see chapter 7.1). Therefore, the Vision Servo does not run any task.

In this study, the major focus is on the Inverse Dynamics Servo: Here the inverse dynamics model is used for predicting feed-forward commands, and thus, also learning the model has to happen on this Servo. The Motor Servo deals with low-level motor control, like a PD controller for negative feedback control as discussed in chapter 2.3. It also handles the input and output operations to control the robot. This involves reading sensors and copying the results into shared memory such that all other Servos have access to them. To perform the IO-operations, the Motor Servo is connected to AJC-Boards that will further be described in the following paragraph. In general, further tasks can be added on the existing or additional targets, such that an almost arbitrarily complex control system can be implemented. The main limitation is given by the VME backplane that has to organize the communication between the single targets. Examples of this additional functionality include a separate Inverse Kinematics Servo for trajectory planning or (as shown in figure 6.2) a Vision Servo for processing input from cameras.

In addition to the targets, special VME boards are necessary to generate the appropriate electrical control signals, which operate the robot. An Advanced Joint Controller (AJC) Board is directly connected to the Motor Servo and to every single joint. With seven degrees of freedom, seven of such AJC boards are needed. Each board performs the transformation in both directions; from logical commands to electrical signals to control the robot and the other way, to convert sensor readings into digital values. Open parameters like the slope and the threshold for the conversion can be adjusted using special software supplied with VxWorks.

6.2 Software for Robot Control

The basic software for controlling the robot exists in a software package called 'SL' (Simulation Laboratory) that has been developed at the CLMC laboratory. This software provides tasks for the basic Servos as described in chapter 6.2, such that the robot can start running immediately. Additional functionality that is required by a task has to be programmed on top of the existing modules. For this thesis, the inverse dynamics model using learning with LWPR had to be programmed to replace the original inverse dynamics that is based on rigid body assumptions.

In addition to the software that runs on the VME boards, the SL package also provides a simulator that can be used to test new software packages. Programmed software written for the robot can run in both, the simulator and on the robot. Only the IO interface (handled by the Motor Servo) behaves in a different way: Either commands are sent to the AJC boards and then sensor readings can be retrieved from these boards. Alternatively, using the simulator, the whole software runs on the host computer only and simulates the robot, i.e. displays a figure of the robot on the controlling computer and returns estimated sensor readings. This offers a valuable support for developing and debugging software, as the same program can be used for the simulator and the real robot. The chance of adding errors is minimized, when transferring software from the simulator to the robot.

Another tool contained in the SL packages is a program for data collection with a graphical display. This software can be used to collect motion data (like positions, velocities, and accelerations for the joints but also for the fingertip) from the running system or from the simulator. The data collected can be exported into ASCII standard files and further used with other programs, e.g. MatLab for additional processing.

As described in chapter 6.1, the robot control hardware consists of several targets, which usually run a single task each. The basic tasks for robot control are provided within the SL package. The functionality of these will be described below:

The **Motor Servo** usually runs at high frequency (510Hz in our case), and performs the low level interaction with the robot. It consists of basic feedback control loops that read errors in desired positions, velocities, and accelerations ($\grave{\mathbf{e}}_{\text{des}}$, $\dot{\grave{\mathbf{e}}}_{\text{des}}$, and $\ddot{\grave{\mathbf{e}}}_{\text{des}}$). A feedback command ($\mathbf{u}_{\text{fb}}$) is generated based on the error between desired and actual value (refer to chapter 2.2). In addition, the Motor Servo simply reads a feed-forward command provided by another Servo and adds both, the feedback and the feed-forward command, to retrieve the final control command **u** that is send to the robot. The Motor Servo's mainloop consists of the following steps:

- Reading $\grave{\mathbf{e}}_{\text{des}}$, $\dot{\grave{\mathbf{e}}}_{\text{des}}$, $\ddot{\grave{\mathbf{e}}}_{\text{des}}$, and $\mathbf{u}_{\text{ff}}$ commands from the shared memory
- Computing appropriate motor-commands for the robot
- Sending the new commands to the AJC-Boards or to the simulator
- Reading the sensor of every joint for position, velocity, and torque information
- Updating the sensor information in shared memory

The aim of the **Task Servo** is to create appropriate desired positions, velocities, and feed-forward commands[19] to accomplish a user specified behavior. Therefore, the Task Servo usually runs at the same high frequency as the Motor Servo to offer a new desired state with every update of the Motor Servo's control loop. The Task Servo can move certain computations by transferring them to additional modules, such as the inverse dynamics or the inverse kinematics computation. Usually, the Motor Servo generates a feed-forward command $\mathbf{u}_{\text{ff}}$ based on rigid body dynamics assumptions (refer to chapter 2.5). In this thesis, however, calculating the feed-forward command will be learned by a separate Servo. The remaining steps of the Task Servo's mainloop are summarized below:

[19] The feed-forward computation was handled by a separate processor in my research due to the computational complexity of the neural network.

- Reading the current state of the robot (**è** , $\dot{}$, and $\ddot{}$)
- Reading data about visual information from shared memory (e.g. information of the position of color blobs)
- Computing useful quantities, e.g. the Jacobian **J** of the forward kinematics, the endeffector-position, -velocity and -acceleration in Cartesian space and storing them in shared memory
- Executing a user-specified set of functions to compute the task-specific desired position, velocity and accelerations
- Storing the desired quantities in the shared memory, such that they can be retrieved by the Motor Servo

The **Inverse Dynamics Servo** will only be used for calculating a non-analytical model of the inverse dynamics. If an analytical model is used, the computations are fast enough to run in real-time on the Task Servo. In contrast, for learning the inverse dynamics model and predicting feed-forward commands by means of neural net techniques (such as ***LWPR***), higher performance is needed. A separate target might therefore be necessary to run the inverse dynamics. Depending on the computational complexity, separate models for different joints have to be learned on separate targets to further enhance computational power within a given time. The mainloop of the Inverse Dynamics Servo consists of the following steps:

- Reading desired positions, velocities and accelerations
- Computing the feed-forward command using ***LWPR*** for predictions
- Storing calculated commands in the shared memory, such that they can be retrieved by the Motor Servo
- Every tenth iteration:[20] Reading sensor information from the Motor Servo and updating the internal model.

The **Inverse Kinematics Servo** can be used to learn a complex model of inverse dynamics with additional constraints, such as human-like motion or extremely energy-

[20] Empirical evaluation showed that learning every 10th iteration (i.e. 51Hz) is a good compromise between computational complexity and accuracy

efficient motion. The functional organization can be seen in analogy to the Inverse Dynamics Servo: The Inverse Kinematics Servo reads the desired quantities and calculates the desired positions, velocities and accelerations instead of the Task Servo. The results will be stored in shared memory, such that the Inverse Dynamics and the Motor Servo can retrieve them to compute control commands. The mainloop of the Inverse Kinematics Servo consists of the following steps:

- Reading desired positions, velocities and accelerations in Cartesian coordinated
- Computing the desired quantities in joint space
- Storing these quantities in the shared memory, such that they can be retrieved by the Motor Servo
- Every tenth iteration: Reading state information from the Motor Servo and updating the internal model, if a learning approach like ***LWPR*** is used.

The research for this thesis only dealt with the implementation of the inverse dynamics estimation located on the Inverse Dynamics Servo. Thus, the remaining system was used 'as it is', i.e. without modifications or the addition of further components. It turned out that SL offered an easy to use yet powerful environment. However, as SL still is in an developmental stage, frequent changes in the code happened that also required adaptation of the inverse dynamics software. Whenever a program did not work properly, errors had to be traced back through the whole system, which needed some additional time. However, the provided simulator was a powerful tool, not only to ease the development of software. The possibility to generate desired trajectories and to sample motion data for a first evaluation of ***LWPR***'s performance saved much effort.

6.3 The Sarcos Robotic Arm

The robot that has been used for the experiments was constructed by SARCOS, a company that usually builds tele-operated robots for entertainment purposes. The Sarcos arm is a copy of another robotic arm, which is part of a humanoid robot also built by Sarcos, and which is running in the ATR laboratories, Kyoto, Japan. The design of this robot, and thus of the robotic arm, follows the ideas of humanoid robots, i.e. it is as compliant and lightweight as possible. The arm is hydraulically actuated and has two different modes of operation: low and high pressure. The low-pressure mode can be used to run algorithms without the danger of breaking the robot or hurting anyone. However, this mode does not offer high accuracy or fast motion. Once switched to high-pressure, obviously the dynamic properties of the system completely changes, such that the low-pressure mode is best used for a secure first evaluation of algorithms only.

The robot arm has seven hydraulically actuated rotary joints and, thus, offers seven degrees of freedom (DOF). The names and functionalities of each DOF are described in the following table:

2 DOF in the shoulder	
- Shoulder abduction / adduction	(SAA)
- Shoulder flexion / extension	(SFE)
1 DOF in the upper arm	
- Humeral rotation	(HR)
1 DOF in the elbow	
- Elbow flexion / extension	(EB)
1 DOF in the lower arm	
- Wrist rotate	(WR)
2 DOF in the wrist	
- Wrist abduction / adduction	(WAA)
- Wrist flexion / extension	(WFE)

The hand offers three additional degrees of freedom in the thumb and the indexing finger. These have, however, not been used for learning as they have hardly any influence on the system's motion due to their limited weight. Of the hand, only its position without respect to the fingers was used for evaluation. Please refer to figure 6.3 for a picture and a functional sketch of the robotic arm.

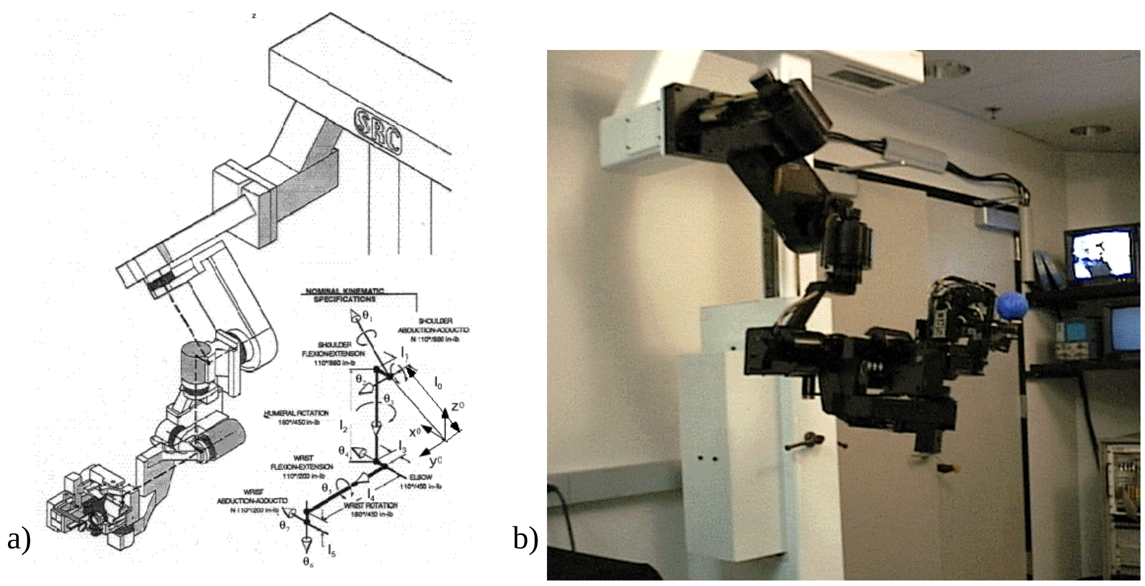
a)
b)
SRC
NOMINAL KINEMATIC SPECIFICATIONS
SHOULDER ABDUCTION-ADDUCTION 110°/880 in-lb
SHOULDER FLEXION-EXTENSION 110°/880 in-lb
HUMERAL ROTATION 180°/450 in-lb
WRIST FLEXION-EXTENSION 110°/200 in-lb
ELBOW 110°/450 in-lb
WRIST ABDUCTION-ADDUCTION 110°/200 in-lb
WRIST ROTATION 180°/450 in-lb
θ_1
θ_2
θ_3
θ_4
θ_5
θ_6
θ_7
l_0
l_1
l_2
l_3
l_4
l_5
z^0
x^0
y^0

Figure 6.3 Sketch (a) and Photograph (b) of the Sarcos Robotic Arm

The main goal of my research is learning the inverse dynamics model for the humanoid robotic arm. As a short reminder, the inverse dynamics model is a typical element of non-linear control (e.g. Sciavicco & Siciliano, 2000), which relates the vectors of position ( ), velocities (˙), and accelerations (¨) of a robot to the torque vector that is necessary to accomplish the acceleration in the given state. Ideal mechanical systems obey inverse dynamics equations from rigid body dynamics

$$\mathbf{B}(\text{è}) \cdot \ddot{\text{è}} + \mathbf{C}(\text{è}, \dot{\text{è}}) \cdot \dot{\text{è}} + \mathbf{G}(\ ) = \qquad \text{(eq. 6.1, same as eq. 2.9)}$$

These equations are highly non-linear and would extend over 1500 lines of C-code for the Sarcos robot arm. Still, rigid body dynamics can be calculated with specialized mathematical packages, and the open parameters (hidden in the parameter matrices **B**, **C**, and **G**) can be estimated from data (e.g., An, Atkeson & Hollerbach, 1988, Featherstone, 1987). The Sarcos robot is hydraulically actuated, and thus, several unknown strong linearities are added to the complexity of the equation. They stem from fluid dynamics, non-linear friction terms of the hydraulic motors, and non-linear saturation of the actuators. From previous experience, it was clear that rigid body dynamics is an insufficient model for the Sarcos Robot Arm. Therefore, the learning algorithm ***LWPR*** was used to estimate the inverse dynamics from the robot's motion. The following chapter will describe the process of learning the inverse dynamics function. It will deal with the trajectory generation, the motion capturing, the learning process, and visualize the results of using ***LWPR*** to predict inverse dynamics commands.

7. Evaluation of the Arm’s Motion

7.1 Generating Trajectories for Learning

To learn the inverse dynamics model, the algorithm needs data samples from real motion. These samples consist of sets of four elements each: position, velocity, acceleration and load. The robot has to be run along a variety of trajectories, such that these data samples can be acquired by measuring the sensors in each joint. As described above, every joint is equipped with a sensor for the position, the velocity and the load. The load sensor is used to measure torques; so only the joint acceleration needs to be estimated using differentiation techniques. The differentiation is performed on the Motor Servo, running at a high frequency of 510 Hz. The data samples are retrieved at a much lower frequency of 50 Hz, leaving enough time and samples to run a low-pass filter on the data. This low-pass filter ensures smooth data compared to the direct differentiation of the sensor readings.

A highly significant question for successful learning is how the imposed trajectories have to be designed. If the robot performs simple repetitive motion, all data samples will be generated by this limited motion. The learning algorithm cannot generalize because it does not have data samples from other areas. To ensure a rich set of data, several trajectories have been developed ranging from continuous motion in joint space to discrete motion in task space, also ranging from single joint motion to all joint motion. As a further significant criterion, the speed of motion can be varied: slow and fast motion has to be represented in the learning data. In addition, it has to be ensured that the whole range of motion is covered. Special care needs to be taken that no joint limits are hit during the process of sampling data. This would lead to false data that the learning algorithm tries to model.

Data samples with respect to the above-mentioned criteria have been created using the following tasks:

Reaching Task (in Joint Space)

This task is designed to explore different joint motions without defining the motion of the fingertip. Therefore, the position of the fingertip is the result of randomized motion in all joints. This task explores motion of the joints in all possible constellations, consequently leaving some joints without any motion. The task iterates through a loop, starting with all joints in their default position. During the loop, a random subset of all joints is chosen (e.g. 1, 4, 5 and 7 of all seven joints). These joints are moved close to either their left or right maximal value within a fixed time using a 5th order spline to generate smooth motions from start to end. After a short delay at the final position, the joints are returned to their original position using the same spline in opposite direction. After another short delay, the iteration starts again with a new random set of joints. The maximal elongation of each joint (in percent of its absolute maximal elongation) and the time for the motion can be varied as parameters to gain variety in the motion. The elongation typically reaches in the order of 70-90% of the maximal elongation, whereas the time for motion varies from 0.6 to 3 seconds for a single motion.

Reaching Task (in Cartesian Space)

The idea of the second reaching task is very similar to the above-mentioned task in joint space. This time, however, the target for this motion is expressed in form of a randomized Cartesian fingertip position. This means, that the desired final joint states are calculated with the inverse kinematics function supplied by SL. This task makes different joints move different widths and at very different speeds, depending on the random target position. An addition to this task is to follow geometrical patterns like lines, pyramids, or boxes. These patterns can be seen as training for expected motion during later operation of the robot. The learning approach will result in better performance if the input data contains data close to the trajectory that will later be used during operation.

Sinusoidal Task (in Joint Space)

The goal of the sinusoidal task is to explore continuous motion in all joints at the same time. During execution, every joint is moved according to a sinusoidal function (with different phase, frequency and amplitude). Some joints are moved according to the superposition of two sinusoidal functions, which changes the motion significantly. The design of the task has to ensure that the patterns of motion do not repeat during execution. Thus, the coefficients of the sinusoidal functions have to be numbers that will not become multiples of each other easily. Additionally, an overall frequency multiplier is permutated during repetitive executions of the task to achieve a broader variety of motion data.

Figure-Eight Task (in Cartesian Space)

The figure-eight task is designed to serve as a verification of the learned inverse dynamics. It draws a figure-eight in the x-z-plane of the fingertip, keeping the y position constant (in Cartesian coordinates). The figure-eight consists of two sinusoidal functions for x and z, with one having double the frequency but half the amplitude of the other: $x = \frac{1}{2}\sin(2 \cdot t \cdot p)$, $z = \sin(t \cdot p)$, $y = y_{def}\,(const)$**.** The overall time to draw one figure-eight was is adjustable in a range from 0.7s to 4s. Thus, recording the x and z position and plotting the trajectory, one can easily evaluate the quality of the motion.

Figure-Eight Task with a Wiggling Component

This is a slightly more sophisticated version for generating training data. It is designed to explore the area close to the desired test trajectory but not exactly along the desired trajectory. For this, small sinusoidal perturbations are added to the original desired fingertip positions of the figure-eight. These perturbations are independent for x, y, and z-directions:

$$\begin{aligned} x_{des} &= \tilde{x}_{des} + x_1 \cdot \sin(x_2 \cdot t + x3) \\ y_{des} &= \tilde{y}_{des} + y_1 \cdot \sin(y_2 \cdot t + y3) \\ z_{des} &= \tilde{z}_{des} + z_1 \cdot \sin(z_2 \cdot t + z3) \end{aligned}$$

During multiple succeeding tasks, these perturbations are varied in amplitude (x_1, y_1, z_1), in frequency (x_2, y_2, z_2), and in phase (x_3, y_3, z_3). The following figure gives an impression of the area of motion covered by the tasks described above.

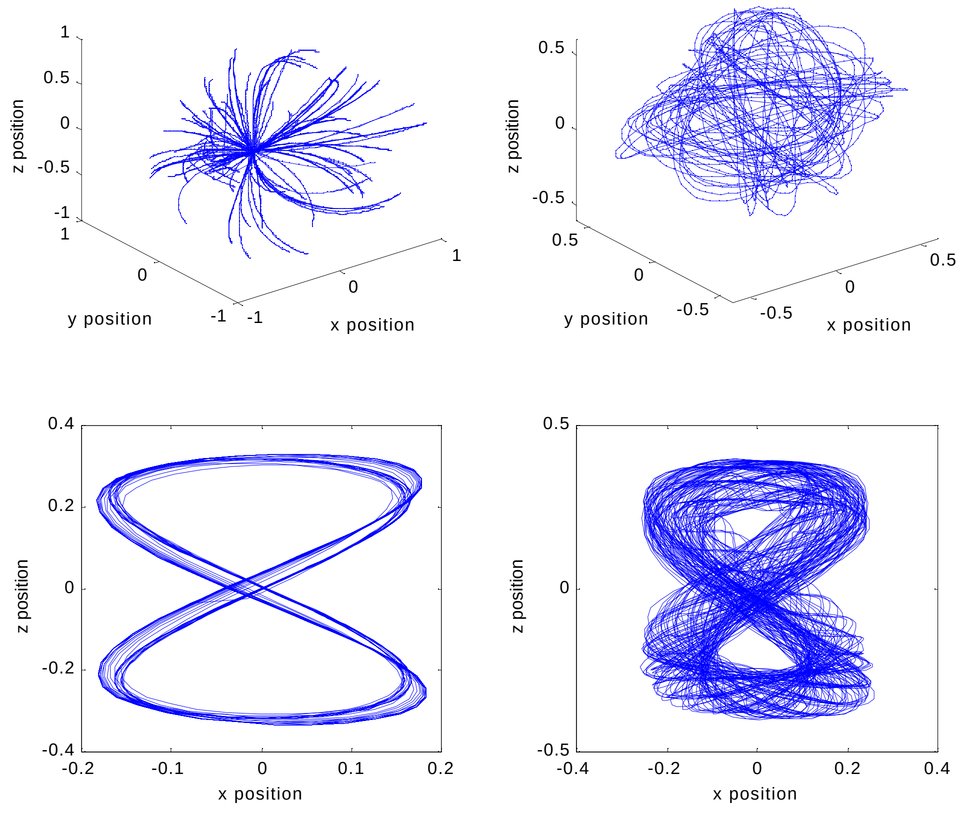

Figure 7.1 Range of Motion (Fingertip Position) covered by Sample Tasks
(all values are shown in meters)

The upper left figure shows a two minutes section of motion described by the reaching task in joint space. One can easily recognize the smooth trajectories resulting from trajectory planning in joint space. The upper right figure shows the fingertip position in Cartesian space during the sinusoidal motion. One can also recognize smooth motion, but this time almost the entire workspace is covered. The lower left figure displays the motion during the plain figure-eight task (only x and z directions are shown) with varying speed of motion. Thus, the figure-eight slowly 'grows' due to uncertainty in the control. The graph in the lower right corner shows the figure-eight task with imposed perturbations, such that the position of the figure-eight and the trajectory of the figure change over time. Again, only the x and z directions are shown. All units of the graphs are meters.

The range of motion has to be carefully checked when sampling data. It needs to be well balanced between covering the interesting area completely, and not exceeding any joint limits. If a joint limit is exceeded, the motion in this joint will stop abruptly and thus, false data is added to the learning set. Obviously, hitting the joint limits might also seriously damage the robot. If this constraint limits the area that can be covered by motion, further exploration will have to take place once the robot has learned the basic area. This means that if the robot later explores previously unseen areas, the learning will continue. Finally, the algorithm knows the inverse dynamics for the whole appropriate range of motion. A second concern must be the maximal torque that any joint can generate. If the torque is exceeded, the desired command cannot be executed as planned. The learning algorithm on the other hand cannot verify if the desired command was executed. Thus, it has to assume the correct execution. As a result, it learns an invalid mapping from command to motion.

7.2 Learning Inverse Dynamics

For learning the inverse dynamics model, different data sets are generated according to the tasks described in chapter 7.1. While collecting data, the feed-forward commands are generated by an analytical model with estimated parameters according to An & Atkeson, et al., 1988 (refer to chapter 2.5). This method returns roughly correct results. Then, a command is executed to move the robot along a desired trajectory. The training samples used as input to the learning algorithm consist of the actual applied command and the resulting state of the robot. Thus, a mapping between a command that was sent and the resulting state of the robot is learned. It does not matter how accurate the analytical inverse dynamics model is, as ***LWPR*** only learns what happens as reaction to an (arbitrarily) applied command. Initially, data were collected for off-line training of ***LWPR*** models in order to gain better insights about the learning process of ***LWPR***, and to allow comparing ***LWPR*** to parametric models based on rigid body dynamics. Later experiments used direct online learning without memorizing any data.

The data collected consists of 21 input variables (joint position, velocity and acceleration for seven joints) and 7 output variables (the joint torques measured by the load cells). The frequency for sampling data was set to 50 Hz; this was found to be fast enough to capture the 'essence' of the motion, but also slow enough not to get too similar data (and thus waste resources, as very similar data does not represent new information)

All tasks are executed in multiple trials with varied parameters, like the maximal elongation or the time for a repetitive motion. Once the initialization of the robot is completed (i.e. a few seconds after the task starts), data is collected for 120 seconds. This results in $120s \cdot 50Hz = 6000$ data points of each single task. Data from the different tasks is then merged into separate sets: One set contains all data from both reaching tasks and the sinusoidal task. This set is assumed to represent 'general motion'. The other set contains all data from the figure-eight tasks, with and without wiggles. This set is later being used to check if learning on data close to the desired trajectory results in superior performance compared to learning on general data. As additional tests, the figure-eight task data again is split into two subsets, one containing data with wiggles and the other containing only data without wiggles around the original figure-eight trajectory. In a way, these training sets are somehow 'cheating' with real learning, as a humanoid robot cannot be trained on exactly what they will do later. This would eventually include all motion possible. Thus, the appropriate learning task is one without the figure-eight data.

Of each of the data sets generated, a random subset (20% of all data) is extracted, i.e. removed from the original data and kept as a separate test set. The remaining 80% of the original data are used as input to train the learning algorithm ***LWPR***. The extracted test set is used to evaluate the performance of ***LWPR***: For this evaluation, every single data in the test set is given to ***LWPR*** to predict the torque needed to reach this state. ***LWPR***'s prediction is then checked against the real torque that has been used to reach this state, which is stored with the test data set. This complex procedure ensures that the algorithmic performance can be evaluated based on data that it has never seen before. Thus, ***LWPR*** needs to generalize - it is not sufficient to memorize all data. The difference between prediction and previously measured torque of all data is summarized in the mean-squared-error and is displayed in graphs as function over the time. These graphs allow easy comparison with other techniques, like the analytical model introduced above.

It should be mentioned that for the seven required torque predictions (one for each joint), seven independent models have to be trained. A single ***LWPR*** model could also generate seven predictions, but in this case, the position and the size of the receptive fields would be the same for all seven predictions. There is no reason to believe that the local models have the same distance metrics for all joints. Therefore, all seven joints must be learned by separate models. Seven independent models offer another advantage: they can be computed in parallel on multiple boards, which will speed up the processing time for both, learning and predicting.

7.3 Learning Results on Sampled Data

As described in chapter 7.2, each of the seven networks needed for predicting the inverse dynamics model of the robot is trained independently on the 'general data' set. The general data set consists of 57.600 training samples, what responds to roughly 20 minutes of actual training. As an example of the resulting learning curves, the diagram for the SFE joint is shown in figure 7.2:

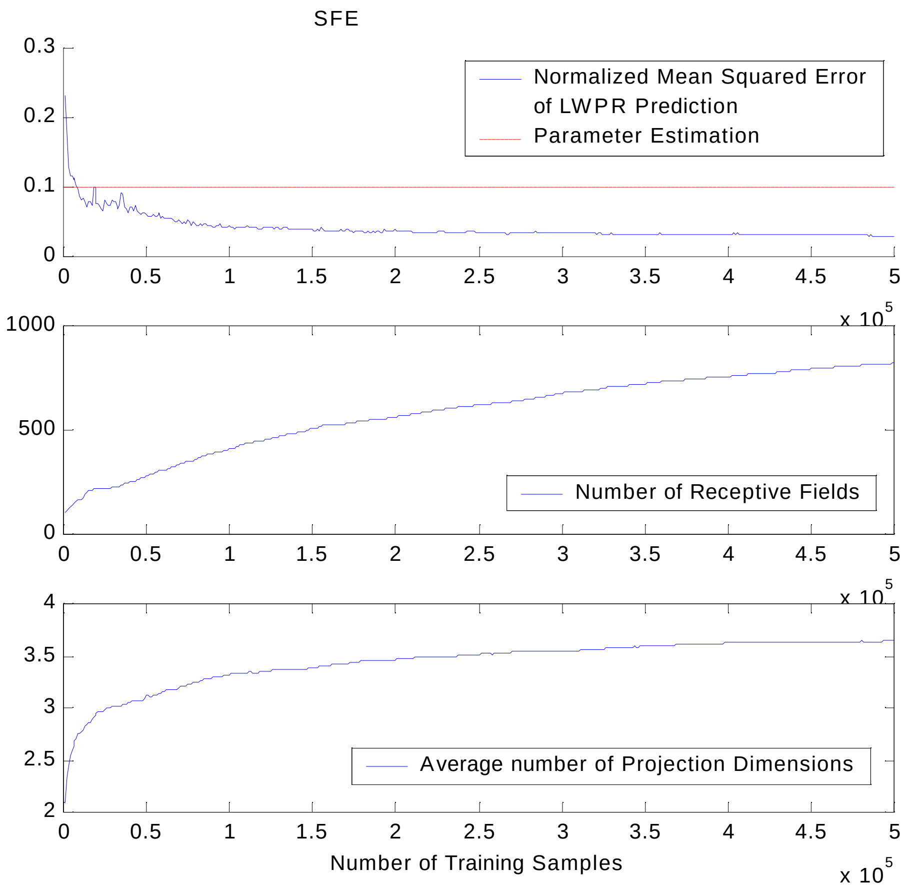

Figure 7.2 The SFE learning Curves as an Example for Learning

The upper part of figure 7.2 shows the decrease of the normalized mean squared error (nMSE) on the separate test set while learning the inverse dynamics of the shoulder flexion/extension joint (SFE). The nMSE is the mean squared error that is found when predicting the test set, normalized by the variance of the test set's output data. The normalization allows easy comparison with other joints or other data sets, which might operate in a very different output range. For example, an absolute error of 0.5 might be an excellent or a catastrophic result, depending on the variance of the output range. In figure 7.2, the nMSE settles at 0.0353 after seeing 500.000 training samples, which is a final error of 3,53 % of the overall output distribution. The 500.000 training samples presented to the learning algorithm represent roughly 10 iterations on the available training data.

To compare this result to another approach, parameter estimation was carried out using standard estimation methods (An, Atkeson, et. al, 1988 and Conradt, Tevatia, et al., 2000) on the same data set. This estimation resulted in an analytical rigid body dynamics model with estimated inertial and geometric parameters. The analytical model that was generated incorporates a large amount of prior knowledge about the physics of the robot system. When comparing both results, the parameter estimation only achieves a nMSE of 0.1 on the same training data (shown as a dotted line in the same diagram). Thus, ***LWPR*** outperforms the analytical approach after very few training samples only.

The diagram in the center of figure 7.2 shows the number of receptive fields that are allocated during learning. One can easily recognize that the remaining error decreases only marginally, once roughly 500 receptive fields have been allocated. The lower diagram in figure 7.2 shows the average number of projection directions used by all receptive fields. This number settles at 3.7, meaning that even though the number of input dimensions was 21, locally less than 4 dimensions are used for predictions. This is a remarkable result, which also strongly supports the research by Vijayakumar (1997) & Schaal about motion data lying in very few dimensions locally.

The learning curves of the other six joints are very similar. For convenience reasons, the final learning results will be summarized in the following table:

Joint Name	Parameter Estimation	LWPR
SAA	0.1952	0.0560
SFE	0.1002	0.0353
HR	0.1539	0.0398
EB	0.2031	0.0492
WR	0.2409	0.0678
WAA	1.1722	0.0342
WFE	0.5535	0.0262

Table 7.1 nMSE Learning Results for the General Data Set

Looking at the table, one will realize that the parameter estimation approach performs significantly worse on the WAA and WFE joints compared to the other joints. This can be explained by the fact that these joints are at the end of the kinematic chain and are especially compliant and lightweight. Therefore, the rigid body dynamics assumptions match these joints poorly. In strong contrast, ***LWPR*** does not perform in any way different on these joints, as it does not incorporate prior knowledge about the joints when learning.

An interesting experiment can be designed with the figure-eight tasks to generate data. In this test, data for training ***LWPR*** is generated using the figure-eight with superposed wiggles, i.e. data close to the desired figure-eight trajectory but not exactly on the trajectory. The data for evaluation, however, is exactly the desired figure-eight trajectory. This test will show how well the different approaches can generalize from a narrow area of data close to the desired data. The training data set used for this test is constructed of nine different sample sets (with slow, medium and fast wiggles) containing 6000 data points each. Thus, the complete training set consists of 54.000 training examples. The test set, however, consists of only 1020 data samples, as the figure-eight for evaluation was drawn within 2 seconds and the Motor Servo was running at a frequency of 510 Hz. The resulting learning curves show the nMSE, the number of receptive fields and the average number of projection directions in figure 7.3:

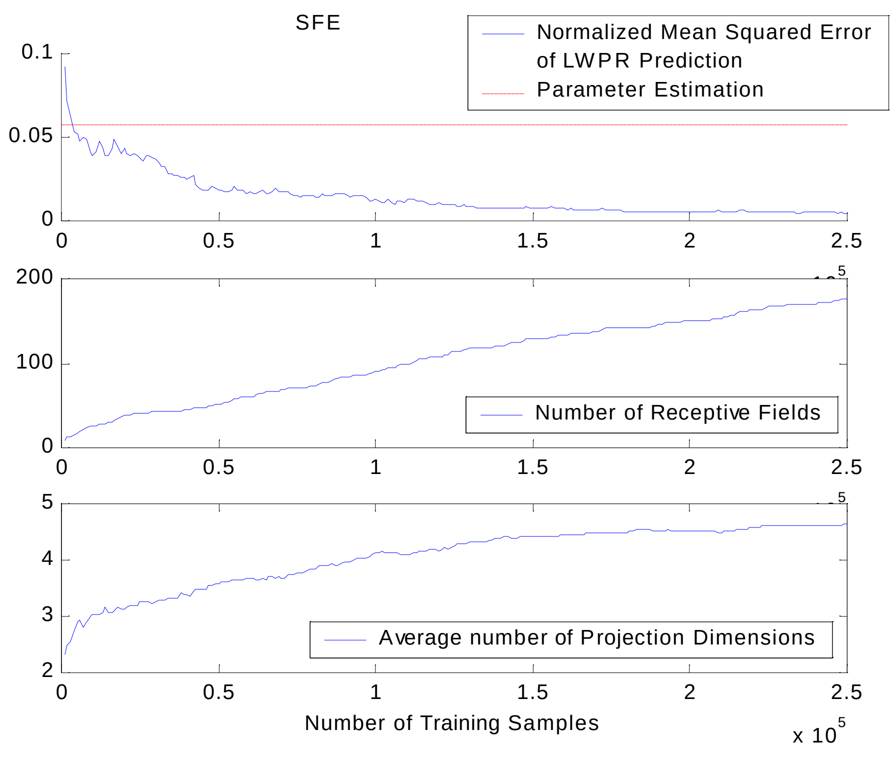

Figure 7.3 The SFE learning Curves on the Figure-Eight Data

One can easily realize that ***LWPR*** outperforms the parametric model within very little time. In figure 7.3, only half the total training input compared to the general data set (250.000 vs. 500.000 data points) is shown. Ultimately, ***LWPR*** converges to a nMSE that is about 12 times lower than the nMSE achieved with the parametric model on this set of data. This is not surprising, as ***LWPR*** can model the presented data samples in the bounded range much better than an analytical model. The analytical model always has to represent the entire range of motion. ***LWPR*** uses fewer receptive fields than in the first learning approach on general data; these receptive fields are, however, located in a much narrower region, such that the overall accuracy of the data prediction becomes better. Additionally, the average number of projection direction increases by almost one, indicating that more features of the input data are used for classification. This happens, because all data is lying much closer together compared to the general data set used before.

Again, the learning curves for the other six joints are very similar to the one presented, such that only the final learning results will be displayed in the table below.

Joint Name	Parameter Estimation	LWPR
SAA	0.1132	0.0054
SFE	0.0581	0.0048
HR	0.1010	0.0017
EB	0.0947	0.0048
WR	0.4584	0.0135
WAA	0.9118	0.0065
WFE	1.7762	0.0070

Table 7.2 nMSE Learning Results for the Figure-Eight Training Set

Training the network for 500.000 iterations took about 2 ½ hours on a 500MHz personal computer running Linux, indicating that LWPR achieves approximately real-time updating (given that the data was collected at 50Hz). Additionally, every network can be trained independently on separate computers, which will increase the performance by a factor of seven. There is hardly any communication overload in this context.

7.4 Verifying the Results on Real Motion

The normalized mean squared error (nMSE) shown as a result in chapter 7.3 allows a first evaluation of the learning quality. However, important is the performance of the robot on desired tasks, not the value of the nMSE. To show these results, the desired states (positions, velocities and accelerations for all seven joints) of two tasks are generated. The feed-forward commands to achieve the desired positions are predicted by ***LWPR***. In a second trial, the analytical model with estimated parameters calculates feed-forward commands. Then, the tasks are executed on the real robot using desired positions and predicted feed-forward commands. During execution, the fingertip position is recorded. The recorded positions serve as a criterion for the evaluation.

The first test is performed using the figure-eight task in Cartesian space as described in chapter 7.1. The trajectory of the figure-eight is executed as a repetitive task using the feed-forward commands predicted by ***LWPR*** and a low gain PD controller. In this test, ***LWPR*** was trained on the 'general data set'. After a few repetitions of the figure-eight to wait for initial disturbances to settle, the fingertip position is recorded for 20 iterations. Figure 7.4 shows the fingertip position in the x-z-plane. For both feed-forward control strategies, the ***LWPR*** predictions and the analytical model with estimated parameters, 20 loops through the figure-eight are displayed, showing that the control stays on track.

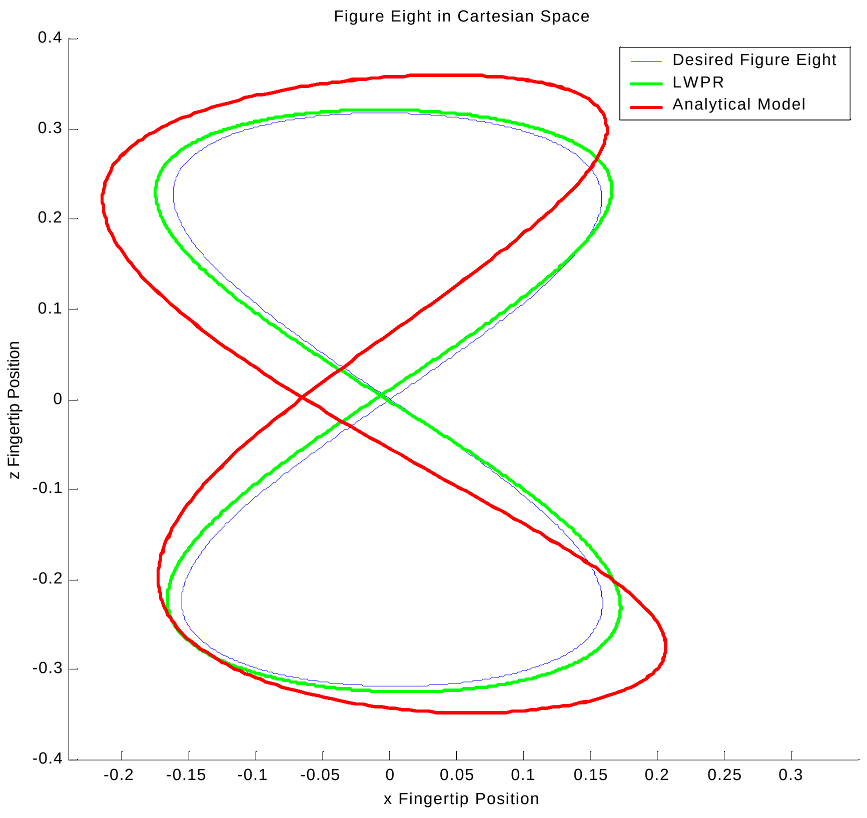

Figure 7.4 Executing the Figure-Eight Task with Feed-forward Control
(all values are shown in meters)

As can easily be seen, ***LWPR*** performs by far superior compared to the analytical model. The trajectory achieved using ***LWPR***'s prediction as feed-forward commands is much closer to the desired trajectory than the one accomplished by the analytical model. Both methods, ***LWPR*** and the analytical model, have used the same input data to estimate the model. The superior performance of ***LWPR*** clearly demonstrates that rigid body dynamics cannot model the humanoid robot arm well.

During the first evaluation, the total time for a figure-eight was set to two seconds. When varying the time, one expects different results. Indeed, longer execution time (i.e. slower drawing of the figure-eight) improves the performance. This is not a property of the prediction or the learning approach; it happens because slower motion is better controlled by the PD controller. The influence of the feed-forward control is significantly reduced when moving slower. Therefore, both approaches improve the quality in roughly the same way.

A more interesting comparison is using the alternate training data set. This set was constructed using motion close to the desired figure-eight (please refer to figure 7.3 and table 7.2). The nMSE based on the figure-eight trajectory was significantly smaller compared to the nMSE achieved on the general data. This leads to the expectation of improved performance on the execution of the task. However, also the nMSE of the analytical model was smaller. So again the desired trajectory is used to predict feed-forward commands and the resulting fingertip trajectories are shown in figure 7.5:

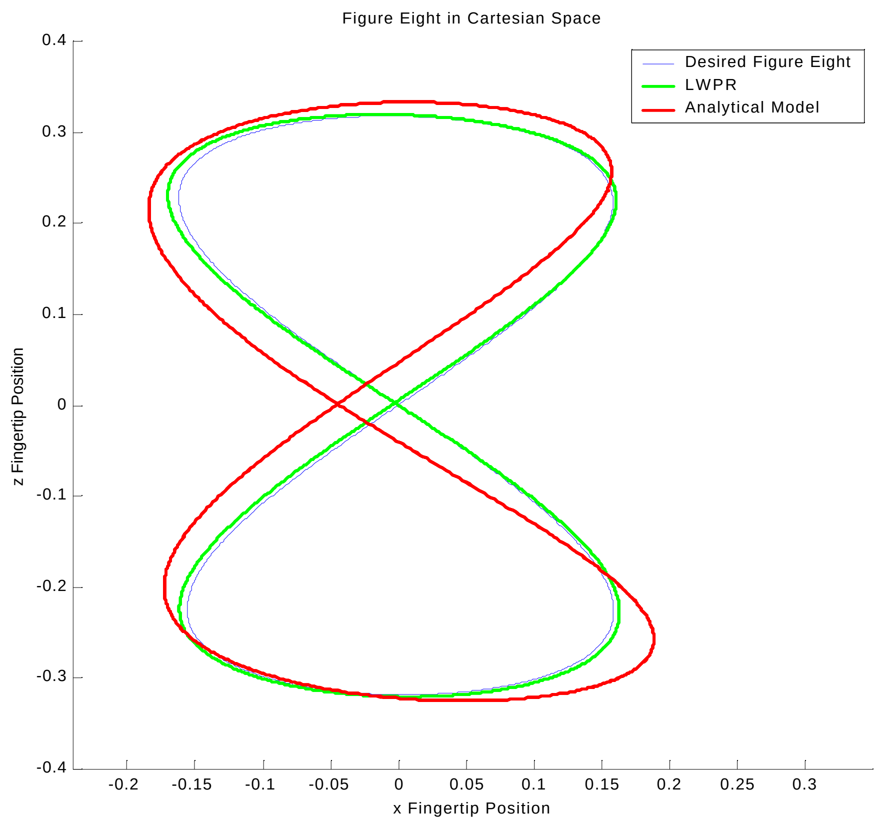

Figure 7.5 Executing the Figure-Eight Task with Feed-forward Control based on Figure-Eight Training Data (all values are shown in meters)

Comparing figure 7.5 to figure 7.4, one can easily recognize that the performance has improved. As expected, both approaches result in trajectories that lie closer to the desired trajectory. Still, ***LWPR*** outperforms the analytical model by far. One important thing to stress is the fact that learning based on data close to the desired trajectory is not a valid approach to solving the inverse kinematics problem. One cannot assume that the desired trajectory is known in advance; so finally, the robot has to perform well on any arbitrary trajectory.

A third and last comparison between ***LWPR***'s predictions and the analytical model was carried out using the reaching task in joint space. The task performs smooth motion in a randomly chosen subset of all joints. After a short delay at the target position, the arm returns to its default position and starts again choosing a different subset of all joints. The total time for a single iteration was set to four seconds, giving almost two seconds to move the arm to the target and again almost two seconds to return. A more detailed description of the task can be found in chapter 7.1. This task is very different compared to the previously evaluated figure-eight task, since it consists of discrete motion in joint space compared to continuous motion in Cartesian space. Again, ***LWPR*** trained on the general data set was used for predicting feed-forward commands. The specialized version (trained on the figure-eight) cannot be assumed to perform accurately, as the limited input data does not cover the range of motion used in the reaching task.

For evaluating the performance, the feed-forward commands of all joints are predicted by ***LWPR***. During a second trial, feed-forward commands are generated by the analytical model estimated on the same data used for training ***LWPR***. Figure 7.6 shows the displacement of the fingertip position compared to the desired trajectory. The actual trajectory cannot be displayed well on paper; therefore, the l2-norm position error (Cartesian distance from desired to real fingertip position) is shown.

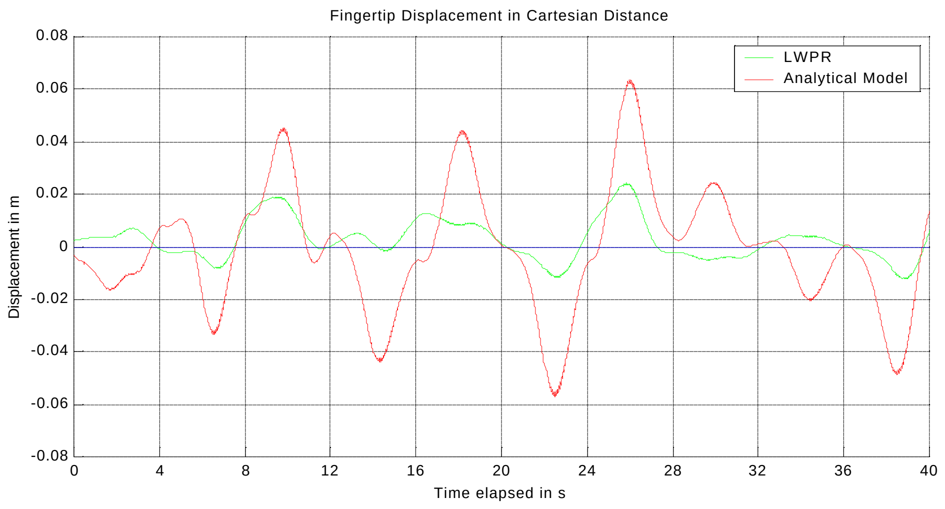

Figure 7.6 Fingertip Displacements during the Reaching Task

This figure again demonstrates the superior performance of ***LWPR*** compared to the analytical parameter estimation technique. ***LWPR*** hardly misses the desired position by more than two centimeters, whereas the analytical estimation is often more than four centimeters off. One can realize that the displacement usually becomes significantly worse when the joints are moved far away from their rest positions (at the positions between the grid lines on the time axis). On the other hand, the displacement usually decreases when the joints are operated close to their rest position (at the positions of the grid lines in figure 7.6). This can be explained by the fact that both algorithms have been exposed to little data from areas far away from the default position. Hence, the approximation ability is significantly reduced in these areas. The global parameter approximation approach, which is building a single model that is valid for the entire space, particularly under-represents these areas. Due to the local learning in ***LWPR***, the little data available is used to generate a model that is valid in this region only. And, as can be seen, this method leads to far superior results for the data given. Again, the local learning model used in ***LWPR*** again by far superior compared to the analytical approach.

8. Conclusion

8.1 Discussion

This thesis presents a new learning algorithm, Locally Weighted Projection Regression (***LWPR***), a non-linear function approximation network that is particularly suited for problems of online incremental motor learning. The essence of ***LWPR*** is to achieve function approximation with piecewise linear models and to find efficient local projections to reduce the dimensionality of the input space. High-dimensional learning problems can thus be dealt with efficiently: updating one projection direction has linear computational cost in the number of inputs, and since the algorithm accomplishes good approximation results with only 2-5 projections irrespective of the number of input dimensions, the overall computational complexity remains linear in the inputs. Moreover, ***LWPR*** selects low dimensional projections and is thus able to exclude irrelevant and redundant dimensions from the input data.

LWPR leads to excellent function approximation for classical problems in robot learning. In this thesis, I concentrate on the problem of estimating the inverse dynamics model of a highly compliant, lightweight humanoid robot. Finding the inverse dynamics model of such a robot is a nontrivial problem that has to be dealt with in new ways. Because of the design of humanoid robots, traditional approaches do not offer high quality. ***LWPR***, in contrast, outperformed the traditional parameter identification method by a large margin. This result by itself is not surprising since it was known in advance that the humanoid robot arm does not conform to the traditional rigid body assumption. What is remarkable, however, is that ***LWPR*** achieves good learning results from data that represented less than one hour of actual robot movement, and that only 2-3 local projections were needed on average. Thus, the 21-dimensional input space of the learning task was drastically reduced.

The major advantages of using the local online learning algorithm ***LWPR*** for estimating the inverse dynamics model are:

- the high learning speed
- the limited prior knowledge about the system needed to achieve good results

- the high accuracy achieved with a limited training data set
- the real-time performance of the system
- the possibility of parallel processing for each joint, such that computational complexity can be transferred to multiple computers for speed-up purposes
- the build-in dimensionality reduction (projection regression) which allows local learning methods to work
- the offline pre-trainability to further speed-up learning on the real robot

The only known disadvantage is the remaining parameter adjustment, which is necessary to achieve accurate learning results.

The main purpose for inventing the new algorithm was to estimate the inverse dynamics model of modern humanoid robots that are compliant and lightweight compared to traditional robots. However, not only such robots can improve their accuracy using ***LWPR***. Traditional robots also suffer from uncertainties in their dynamics model, and the learning approach offers a new way to deal with this problem. Additionally, as traditional robots age, their mechanical properties change. An online learning approach will always keep the model accurate, no matter how the mechanical system changes (as long as the wear remains moderate). One step further, a learning approach will estimate the appropriate inverse dynamics model for any robot it is used on, with almost no human interaction. This can drastically simplify the design process of new robots.

On the other hand, not only the inverse dynamics problem can be solved by ***LWPR***. Other regression problems with similar constraints can also be dealt with. An example is shown is the following chapter, using ***LWPR*** to estimate inverse dynamics.

8.2 Brief Introduction to Using *LWPR* for Inverse Kinematics Estimation

One of the core issues of robot control is movement planning, as outlined in chapter 2.1. Most movement tasks are defined in coordinate systems that are different from the actuator space of the robot. Hence, a coordinate transformation from task to actuator space must be performed before motor commands can be computed. On a system with redundant degrees of freedom, this transformation from external plans to internal coordinates is often ill posed and is known as the inverse kinematics problem. For redundant manipulators like the Sarcos robot arm, solutions of the inverse dynamics equation are usually non-unique.

Traditional pseudo-inverse methods, like Resolved Motion Rate Control, suffer from the problem of singular postures and need additional optimization criteria to resolve the redundancy of the system. At singular postures, i.e. when the Jacobian becomes rank deficient, these methods cannot yield a solution and suffer from numerical explosions. Given these problems, a learning system might offer an appropriate solution: it can only reproduce the solutions it was trained on, i.e. problems with singularities of the Jacobian **J** cannot arise since it is impossible to encounter 'singular training data'.

Due to the redundancy of the robot arm, learning the inverse problems is usually not possible if the redundant solutions $\dot{}$ for one $\dot{\mathbf{x}}$ form a non-convex set (Jordan & Rumelhart, 1992). This problem can be avoided by a specific input representation to the learning network. Consider two solutions for $\dot{}$ from the set of solution vectors that produce the same end-effector velocity:

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{è}) \cdot \dot{\mathbf{è}}_1$$

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{è}) \cdot \dot{\mathbf{è}}_2$$

Since the Jacobian relates the $\dot{\mathbf{x}}$ and $\dot{\mathbf{è}}$ in linear form, even for a redundant system the average of the two solutions will result in the desired $\dot{\mathbf{x}}$, i.e.:

$$\dot{\mathbf{x}} = \frac{\mathbf{J}(\mathbf{è}) \cdot (\dot{\mathbf{è}}_1 + \dot{\mathbf{è}}_2)}{2} = \mathbf{J}(\mathbf{è}) \cdot \dot{\mathbf{è}}_{AVE} \qquad \text{(eq. 8.1)}$$

Thus, if one considers only a small local region of the space, the set of solutions of joint velocity vectors for one particular $\dot{\mathbf{x}}$ form a convex set. Averaging over this local convex set - this is what neural network learning and thus ***LWPR*** essentially does - will lead to a valid solution for the inverse kinematics problem (equation 8.1). Thus, a specially localized learning system like ***LWPR*** can learn the inverse mapping function based on the input/output representation $(\dot{\mathbf{x}}, \mathbf{è}) \rightarrow (\dot{\mathbf{è}})$.

This approach will automatically resolve the redundancy problem without resorting to any other optimization approach; the inverse solution is simply the local average over solutions previously experienced. The algorithm will also perform well near singular posture since it cannot generate joint movement that it has never experienced.

Please refer to Conradt, Tevatia, et al., 2000, for a more detailed description of the learning process for the inverse kinematics problem, and graphs that demonstrate the performance of this approach. We also discuss the problem of overcome previously inexperienced areas by adding a small default joint velocity.

8.2 Future Research

In general, ***LWPR*** opens a very promising approach towards new motion abilities of lightweight compliant robots. Definitely, further experiments are needed to evaluate the robot's performance by using ***LWPR***. Also, more different types of robots need to be tested. One such test will be performed this summer in Japan on the real humanoid robot (shown in figure 8.1) with additional redundant input dimensions. This addresses the question of scalability into higher dimensions with more redundant inputs.

Current research focuses on creating a complete real-time implementation of the two robot-learning tasks in a multi-processor environment: The inverse dynamics model is implemented, and all the benchmark tests indicate that it will be unproblematic to additionally apply ***LWPR*** in real-time to inverse kinematics. To my knowledge, no other research groups have accomplished similar fast and accurate learning results for such high-dimensional learning problems in the context of humanoid robotics.

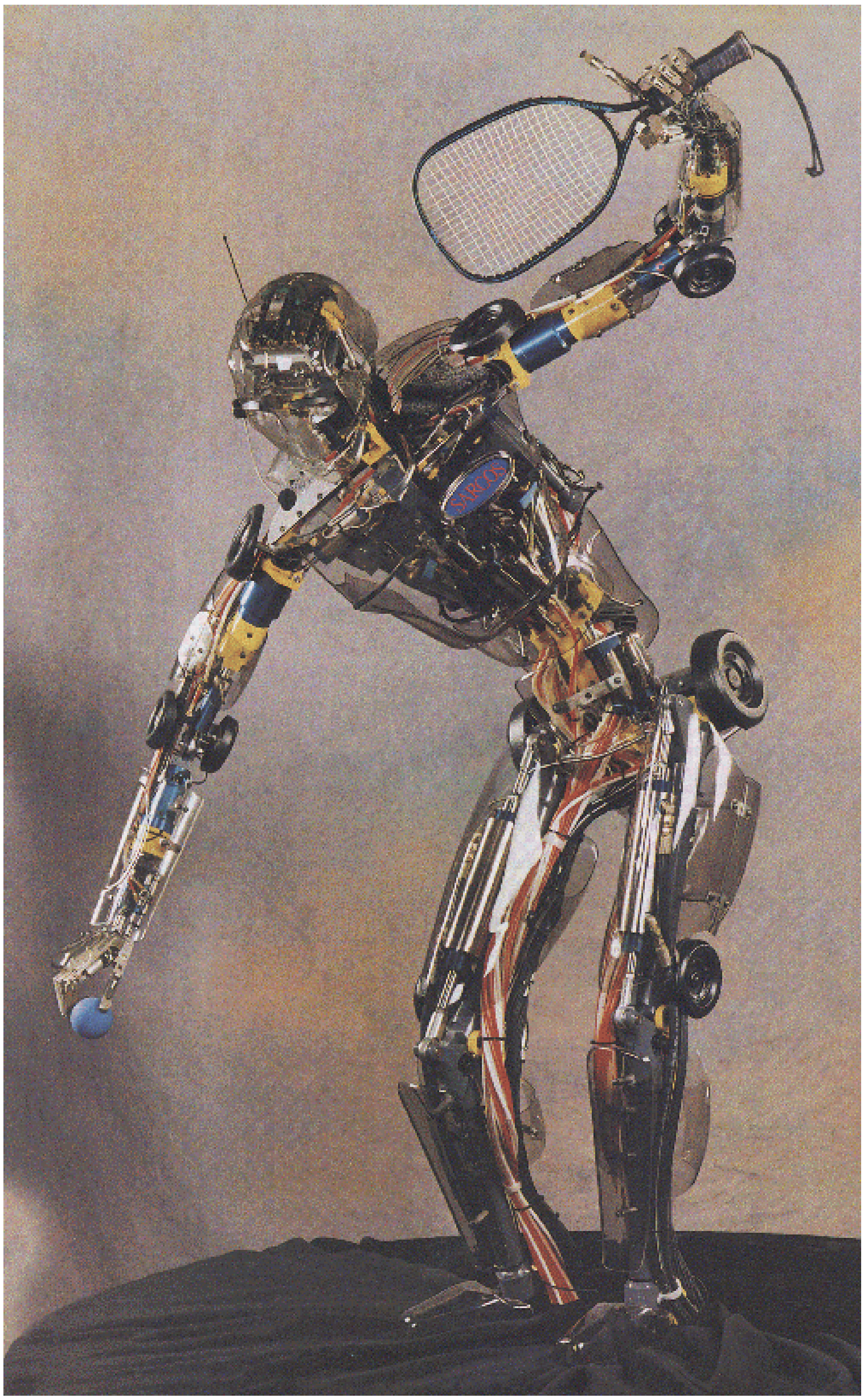

Figure 8.1 Photograph of DB playing Squash

References

An, C., Atkeson, C. & Hollerbach, J. (1988), *Model-Based Control of a Robot Manipulator*. Cambridge, MA, MIT-press.

Atkeson, C., Moore, A. & Schaal, S. (1997), *Locally weighted learning*. Artificial Intelligence Review, 11, 1-5, pp. 11-73.

Atkeson, C. (1989), *Using local models to control movement*. Advances in Neural Processing Information Systems 1, pp. 79-86, San Mateo, CA. Morgan Kaufmann.

Atkeson, C. & Schaal, S. (1995), *Memory-based neural networks for robot learning*. Neurocomputing, 9, pp. 243-269.

Baillieul, J. (1985), *Kinematic programming alternatives for redundant manipulators*. In: IEEE International Conference on Robotics and Automation.

Belsley, D., Kuh, E. & Welsch, R. (1980), *Regression diagnostics: Identifying influential data and sources of collinearity*. New York, Wiley.

Cleveland, W. (1979), *Robust locally weighted regression and smoothing scatterplots*. Journal of the American Statistical Association, 74, pp. 829-836.

Cleveland, W. & Loader, C. (1995), *Smoothing by local regression: Principles and methods*. Technical Report, AT&T Bell Laboratories, Murray Hill, NY.

Conradt, J., Tevatia, G., Vijayakumar, S. & Schaal, S. (2000), *On-line Learning in Humanoid Robot Systems*. Proceedings of the Seventeenth International Conference on Machine Learning, San Francisco.

Craig, J. (1986), *Introduction to robotic*s. Readings, MA, Addison-Wesley.

Daugman, J. & Downing, C. (1995), *Gabor wavelets for statistical pattern recognition*. In: Arbib, M.: The Handbook of Brain Theory and Neural Networks, pp. 414-420, Cambridge, MA, MIT press.

Deco, G. & Obradovic, D. (1996), *An information-theoretic approach to neural computation*. New York, Springer.

Fan, J. & Gijbels, I. (1996), *Local polynomial modeling and its application*. London, Chapman & Hall.

Farmer, J. & Sidorowich, D. (1987), *Predicting chaotic time series*. Physical review letters, 59 (8), pp. 845-848.

Farmer, J. & Sidorowich, D. (1988), *Exploiting chaos to predict the future and reduce noise*. In Lee, Y.: Evolution, Learning, and Cognition, p.27, Singapore, World Scientific.

Field, D. (1994), *What is the goal of sensory coding?* Neural Computation, 6, pp. 559-601.

Frank, I. & Friedman, J. (1993), *A statistical view of some chemometric regression tools*. Technometrics, 35(2), pp. 109-135.

Friedman, J. & Stutzle, W. (1981), *Projection pursuit regression*. Journal of the American Statistics Association, 76, pp. 817-823.

Friedman, J. (1984), *A variable span smoother*. Technical report No. 5, Department of Statistics, Stanford University.

Geman, S., Bienenstock, E. & Doursat, R. (1992), *Neural Networks and the Bias/ Variance Dilemma,* Neural Computation, 4, pp. 1-58.

Georgopoulos, A. (1991), *Higher order motor control*. Annual Review of Neuroscience, 14, pp. 361-377.

Hastie, T. & Loader, C. (1993), *Local regression: Automatic kernel carpentry*. Statistical Science, 8, pp. 120-143.

Hirai, K. (1997), *Current and future perspective of Honda humanoid robot*. Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 500-508, Grenoble, France, IEEE, New York.

Hirai, K., Hirose, M., Haikawa, Y. & Takenaka, T. (1998), *The development of Honda humanoid robot*, IEEE International Conference on Robotics and Automation, pp. 1321-1326, Leuven, Belgium, IEEE, New York.

Hubel, D. & Wiesel, T. (1959), *Receptive fields of single neurons in the cat's striate cortex*. Journal of Neurophysiology, 148, pp. 574-591.

Jordan, M. & Rumelhart R. (1992), *Supervised learning with a distal teacher*. Cognitive Science, pp. 307-354.

Jordan, M. & Jacobs, R. (1994), *Hierarchical mixtures of experts and the EM algorithm*. Neural Computation, 3, pp. 79-87.

Kovács, P. (1993), *Rechnergestützte symbolische Roboterkinematik*. In: Fortschritte der Robotik, 18, Vieweg Verlag, 1993.

Lee, C., Rohrer, W. & Sparks, D. (1988), *Population coding of saccadic eye movement by neurons in the superior colliculus*. Nature, 332, pp. 357-360.

Liegeois, A. (1977), *Automatic supervisory control of the configuration and behavior of multibody mechanisms.* IEEE Transactions on Systems, Man, and Cybernetics, 7(12), pp. 868-871.

Ljung, L. & Söderström, T. (1986), *Theory and practice of recursive identification*. Cambridge, MIT press.

Merzenich, M., Kaas, J., Nelson, R., Sur, M. & Fellemann, D. (1983) *Topographic reorganization of somatosensory cortical areas 3b and 1 in adult monkeys following restricted deafferentation*. Neuroscience, 8, pp. 33-55.

Moody, J. & Darken, C. (1988), *Learning with localized receptive fields*. In Touretzky, D.: Proceedings of the 1988 Connectionists Summer School, pp. 133-143, San Mateo, CA, Morgan Kaufmann.

Moore, A. (1991), *Fast, robust, and adaptive control by learning only forward models*. In Moody, J.: Advances in Neural Information Processing Systems 4, San Mateo, CA, Morgan Kaufmann.

Mountcastle, V. (1957), *Modality and topographic properties of single neurons of cat's somatic sensory cortex*. Journal of Neurophysiology, 20, pp. 408-434.

Nadaraya, E. (1964), *On estimating regression*. Theoretical Probabilistics Applied, 9, pp. 141-142.

Olshausen, B. & Field, D. (1996), *Emergence of simple-cell receptive field properties by learning a sparse code for natural images*. Nature, 381, pp. 607-609.

Perrone, M. & Cooper, L. (1993), *When networks disagree: Ensemble methods for hybrid neural networks*. In Mammone, R.: Neural Networks for Speech and Image processing, Chapman Hall.

Pfeiffer, F. & Reithmeier, E. (1987), *Roboterdynamik*. Stuttgart, Teubner Verlag.

Poggio, R. & Girosi, F. (1990), *Regularization algorithms for learning that are equivalent to multilayer networks*. Science, 247, 4945, pp. 978-982.

Powell, M. (1987), *Radial basis functions for multivariable interpolation: A review*. In Mason, J.: Algorithms for approximation, pp. 143-167, Oxford, Clarendon Press.

Rieseler, H. (1992), *Roboterkinematik – Grundlagen, Invertierung und symbolische Berechnung*. In Fortschritte der Robotik, 16, Vieweg Verlag.

Schaal, S. & Atkeson, C. (1994), *Assessing the quality of learned local models*. In Cowan, J.: Advances in Neural Information Processing Systems 6, San Mateo, CA, Morgan Kaufmann.

Schaal, S. & Atkeson, C. (1998), *Constructive Incremental Learning From Only Local Information*. Neural Computation, 10, 8, pp. 2047-2084.

Schwinn, W. (1992), *Grundlagen der Roboterkinematik*, LS-Verlag.

Sciavicco, L. & Siciliano, B. (2000), *Modeling and Control of Robot Manipulators*. New York, Springer Verlag.

Scott, D. (1992), *Multivariate Density Estimation*. New York, Wiley.

Vijayakumar, S. & Schaal, S. (1997), *Local dimensionality reduction for locally weighted learning*. IEEE International Symposium on Computational Intelligence in Robotics and Automation, pp. 220-225, Monterey, CA.

Vijayakumar, S. & Schaal, S. (2000a), *Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space*. Proceedings of the Seventeenth International Conference on Machine Learning, San Francisco.

Vijayakumar, S. & Schaal, S. (2000b), *Real Time Learning in Humanoids: A challenge for Scalability of Online Algorithms*. IEEE International Conference on Humanoid Robotics, Boston.

Wahba, G. & Wold, S. (1975), *A completely automatic french curve: Fitting spline functions by cross-validation*. Communications in Statistics, 4(1).

Watson, G. (1964), *Smooth regression analysis*. In Sankhaya, G.: The Indian Journal of Statistics A, 26, pp. 359-372.

Whitney, D. (1969), *Resolved motion rate control of manipulators and human prostheses*. IEEE Transactions on Man-Machine Systems, 10(2), pp. 47-53.

Wold, H. (1975), *Soft modeling by latent variables: the nonlinear iterative partial least squares approach*. Perspectives in Probability and Statistics.

Short German Summary

Das Ziel der vorliegenden Diplomarbeit ist es, ein neues Verfahren zur Bewegungssteuerung menschenähnlicher Roboter (siehe Schaubild 9.1a) zu entwickeln und zu bewerten. Solche menschenähnlichen Roboter unterscheiden sich von traditionellen Robotern dadurch, daß sie aus leichtgewichtigen Materialien gebaut sind und eine hohe Flexibilität in ihren Gelenken aufweisen. Durch diese Änderungen können klassische Methoden zur Bewegungssteuerung nicht mehr akkurat arbeiten. Insbesondere war es bisher nicht möglich, gleichzeitig Genauigkeit der Bewegungen und Flexibilität in den Gelenken zu erzielen.

Ein möglicher Ansatz für eine neue Steuerung besteht in der Anwendung von biologisch inspirierten Lernalgorithmen. Diese versuchen, durch nur lokal gültige lineare Funktionen die unbekannte Dynamik eines Roboters zu beschreiben. Der hier vorgestellte und benutzte Algorithmus ***LWPR*** ist bestens dafür geeignet, eine solche Steuerung zu lernen.

Um den Lernalgorithmus zu trainieren, benutze ich einen Roboterarm, der sich entlang vielgestaltiger Trajektorien bewegt. Durch die gleichzeitige Messung der Armbewegung entsteht ein Datensatz, der die Bewegungsdynamik beschreibt. Ein ***LWPR***-Netzwerk kann aus diesen Daten das Dynamikmodell des Roboters lernen und anschließend zur Generierung gewünschter Bewegungsbefehle eingesetzt werden. Im Vergleich zu traditionellen Methoden erhöht dieses neue Verfahren die Genauigkeit der Bewegung von menschenähnlichen Robotern um ein Vielfaches, ohne deren Flexibilität einzuschränken. Ein Beispiel für die Auswertung einer Bewegung ist in Schaubild 9.1b wiedergegeben. Dort ist die Trajektorie einer Figur-8 des Endeffektors aufgezeichnet. Man kann deutlich sehen, daß die von ***LWPR*** generierte Bahn wesentlich näher an der gewünschten Bahn liegt als die aus traditionellen Methoden resultierende.

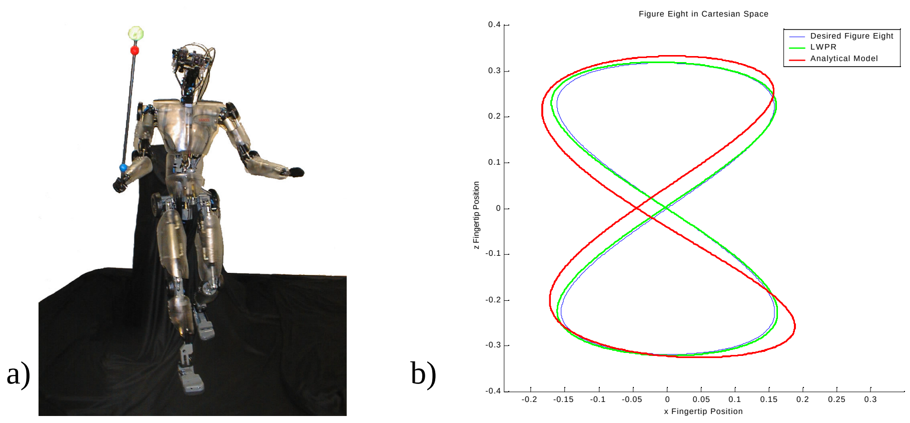

Schaubild 9.1 a) Menschenähnlicher Roboter DB, b) Aufgezeichnete Trajektorie der Figur-8

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Berlin, den 12. 2. 2001