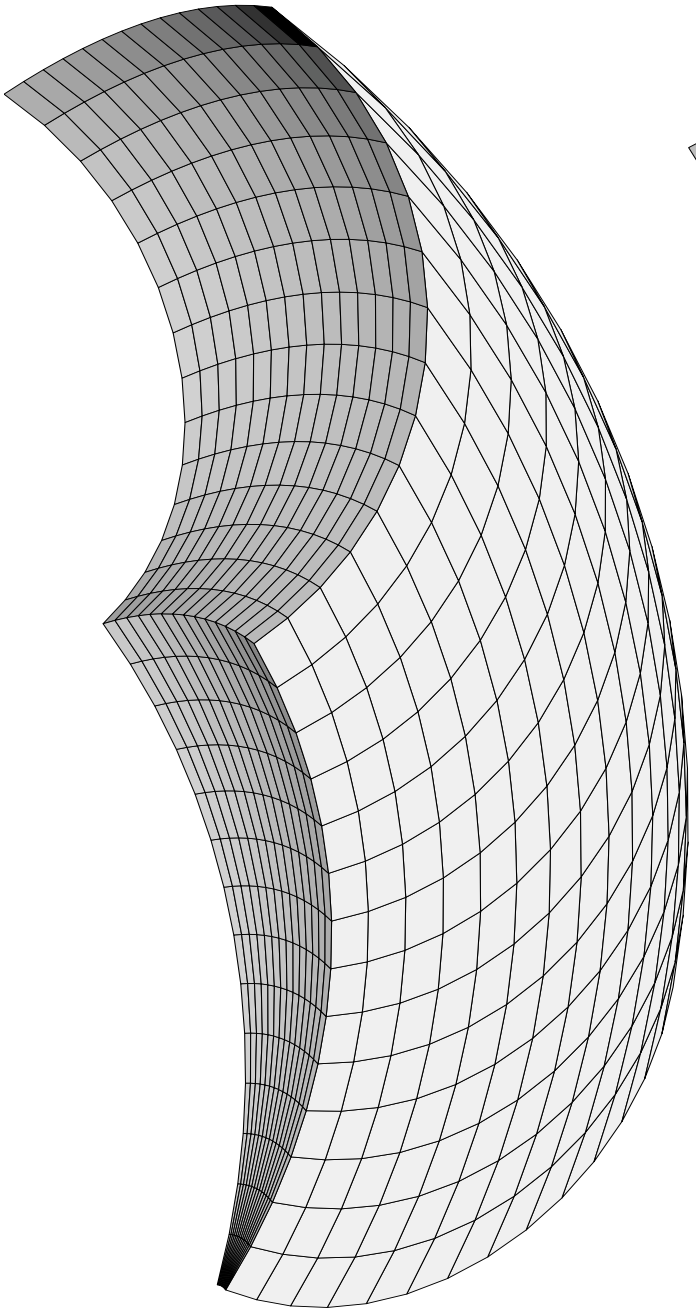
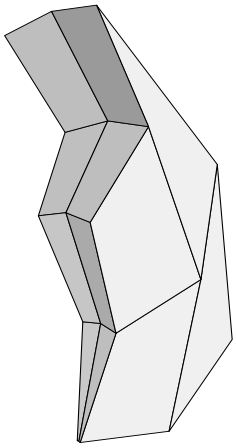
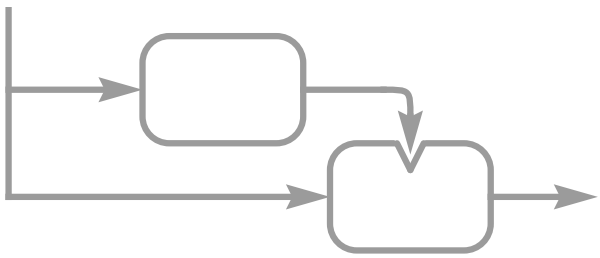
Rapid Learning in Robotics







Jörg Walter

Die Deutsche Bibliothek — CIP Data

Walter, Jörg
Rapid Learning in Robotics / by Jörg Walter, 1st ed.
Göttingen: Cuvillier, 1996
Zugl.: Bielefeld, Univ., Diss. 1996
ISBN 3-89588-728-5

Jörg A. Walter

Rapid Learning in Robotics

Robotics deals with the control of actuators using various types of sensors and control schemes. The availability of precise sensorimotor mappings – able to transform between various involved motor, joint, sensor, and physical spaces – is a crucial issue. These mappings are often highly non-linear and sometimes hard to derive analytically. Consequently, there is a strong need for rapid learning algorithms which take into account that the acquisition of training data is often a costly operation.

The present book discusses many of the issues that are important to make learning approaches in robotics more feasible. Basis for the major part of the discussion is a new learning algorithm, the *Parameterized Self-Organizing Maps*, that is derived from a model of neural self-organization. A key feature of the new method is the rapid construction of even highly non-linear variable relations from rather modestly-sized training data sets by exploiting topology information that is not utilized in more traditional approaches. In addition, the author shows how this approach can be used in a modular fashion, leading to a learning architecture for the acquisition of basic skills during an "investment learning" phase, and, subsequently, for their rapid combination to adapt to new situational contexts.

Foreword

The rapid and apparently effortless adaptation of their movements to a broad spectrum of conditions distinguishes both humans and animals in an important way even from nowadays most sophisticated robots. Algorithms for rapid learning will, therefore, become an important prerequisite for future robots to achieve a more intelligent coordination of their movements that is closer to the impressive level of biological performance.

The present book discusses many of the issues that are important to make learning approaches in robotics more feasible. A new learning algorithm, the *Parameterized Self-Organizing Maps*, is derived from a model of neural self-organization. It has a number of benefits that make it particularly suited for applications in the field of robotics. A key feature of the new method is the rapid construction of even highly non-linear *variable relations* from rather modestly-sized training data sets by exploiting topology information that is unused in the more traditional approaches. In addition, the author shows how this approach can be used in a modular fashion, leading to a learning architecture for the acquisition of basic skills during an "investment learning" phase, and, subsequently, for their rapid combination to adapt to new situational contexts.

The author demonstrates the potential of these approaches with an impressive number of carefully chosen and thoroughly discussed examples, covering such central issues as learning of various kinematic transforms, dealing with constraints, object pose estimation, sensor fusion and camera calibration. It is a distinctive feature of the treatment that most of these examples are discussed and investigated in the context of their actual implementations on real robot hardware. This, together with the wide range of included topics, makes the book a valuable source for both the specialist, but also the non-specialist reader with a more general interest in the fields of neural networks, machine learning and robotics.

Helge Ritter
Bielefeld

Acknowledgment

The presented work was carried out in the connectionist research group headed by Prof. Dr. Helge Ritter at the University of Bielefeld, Germany.

First of all, I'd like to thank Helge: for introducing me to the exciting field of learning in robotics, for his confidence when he asked me to build up the robotics lab, for many discussions which have given me impulses, and for his unlimited optimism which helped me to tackle a variety of research problems. His encouragement, advice, cooperation, and support have been very helpful to overcome small and larger hurdles.

In this context I want to mention and thank as well Prof. Dr. Gerhard Sagerer, Bielefeld, and Prof. Dr. Sommer, Kiel, for accompanying me with their advises during this time.
Thanks to Helge and Gerhard for refereeing this work.

Helge Ritter, Kostas Daniilidis, Ján Jokusch, Guido Menkhaus, Christof Dücker, Dirk Schwammkrug, and Martina Hasenjäger read all or parts of the manuscript and gave me valuable feedback. Many other colleagues and students have contributed to this work making it an exciting and successful time. They include Jörn Clausen, Andrea Drees, Gunther Heidemannn, Hartmut Holzgraefe, Ján Jockusch, Stefan Jockusch, Nils Jungclaus, Peter Koch, Rudi Kaatz, Michael Krause, Enno Littmann, Rainer Orth, Marc Pomplun, Robert Rae, Stefan Rankers, Dirk Selle, Jochen Steil, Petra Udelhoven, Thomas Wengereck, and Patrick Ziemeck. Thanks to all of them.

Last not least I owe many thanks to my Ingrid for her encouragement and support throughout the time of this work.

Contents

List of Figures

* (10/207) Illustrations contributed by Dirk Selle [2.5], Ján Jockusch [2.8, 2.9], and Bernd Fritzke [6.8].

Chapter 1

Introduction

In school we *learned* many things: e.g. vocabulary, grammar, geography, solving mathematical equations, and coordinating movements in sports. These are very different things which involve declarative knowledge as well as procedural knowledge or skills in principally all fields. We are used to subsume these various processes of obtaining this knowledge and skills under the single word *"learning"*. And, we learned that learning is important. Why is it important to a living organism?

Learning is a crucial capability if the effective environment cannot be foreseen in all relevant details, either due to complexity, or due to the non-stationarity of the environment. The mechanisms of learning allow nature to create and re-produce organisms or systems which can evolve — with respect to the later given environment — optimized behavior.

This is a fascinating mechanism, which also has very attractive technical perspectives. Today many technical appliances and systems are standardized and cost-efficient mass products. As long as they are non-adaptable, they require the environment and its users to comply to the given standard. Using learning mechanisms, advanced technical systems can adapt to the different given needs, and locally reach a satisfying level of helpful performance.

Of course, the mechanisms of *learning* are very old. It took until the end of the last century, when first important aspects were elucidated. A major discovery was made in the context of physiological studies of animal digestion: Ivan Pavlov fed dogs and found that the inborn ("unconditional") salivation reflex upon the taste of meat can become accompanied by a *conditioned reflex* triggered by other stimuli. For example, when a bell

was rung always before the dog has been fed, the *response* salivation became *associated* to the new *stimulus*, the acoustic signal. This fundamental form of associative learning has become known under the name *classical conditioning*. In the beginning of this century it was debated whether the conditioning reflex in Pavlov's dogs was a stimulus–response (S-R) or a stimulus–stimulus (S-S) association between the perceptual stimuli, here taste and sound. Later it became apparent that at the level of the nervous system this distinction fades away, since both cases refer to associations between neural representations.

The fine structure of the nervous system could be investigated after staining techniques for brain tissue had become established (Golgi and Ramón y Cajal). They revealed that neurons are highly interconnected to other neurons by their tree-like extremities, the dendrites and axons (comparable to input and output structures). D.O. Hebb (1949) postulated that the synaptic junction from neuron A to neuron B was strengthened each time A was activated simultaneously, or shortly before B. Hebb's rule explained the conditional learning on a qualitative level and influenced many other, mathematically formulated learning models since. The most prominent ones are probably the *perceptron*, the *Hopfield model* and the *Kohonen map*. They are, among other neural network approaches, characterized in chapter 3. It discusses learning from the standpoint of an approximation problem. How to find an efficient mapping which solves the desired learning task? Chapter 3 explains Kohonen's "Self-Organizing Map" procedure and techniques to improve the learning of continuous, high-dimensional output mappings.

The appearance and the growing availability of computers became a further major influence on the understanding of learning aspects. Several main reasons can be identified:

First, the computer allowed to isolate the mechanisms of learning from the wet, biological substrate. This enabled the testing and developing of learning algorithms in simulation.

Second, the computer helped to carry out and evaluate neuro-physiological, psychophysical, and cognitive experiments, which revealed many more details about information processing in the biological world.

Third, the computer facilitated bringing the principles of learning to technical applications. This contributed to attract even more interest and opened important resources. Resources which set up a broad interdisci-

plinary field of researchers from physiology, neuro-biology, cognitive and computer science. Physics contributed methods to deal with systems constituted by an extremely large number of interacting elements, like in a ferromagnet. Since the human brain contains of about 10^{10} neurons with 10^{14} interconnections and shows a — to a certain extent — homogeneous structure, stochastic physics (in particular the Hopfield model) also enlarged the views of neuroscience.

Beyond the phenomenon of "learning", the rapidly increasing achievements that became possible by the computer also forced us to re-think about the before unproblematic phenomena "machine" and "intelligence". Our ideas about the notions "body" and "mind" became enriched by the relation to the dualism of "hardware" and "software".

With the appearance of the computer, a new modeling paradigm came into the foreground and led to the research field of *artificial intelligence*. It takes the digital computer as a prototype and tries to model mental functions as processes, which manipulate symbols following logical rules – here fully decoupled from any biological substrate. Goal is the development of algorithms which emulate cognitive functions, especially human intelligence. Prominent examples are chess, or solving algebraic equations, both of which require of humans considerable mental effort.

In particular the call for practical applications revealed the limitations of traditional computer hardware and software concepts. Remarkably, traditional computer systems solve tasks, which are distinctively hard for humans, but fail to solve tasks, which appear "effortless" in our daily life, e.g. listening, watching, talking, walking in the forest, or steering a car.

This appears related to the fundamental differences in the information processing architectures of brains and computers, and caused the renaissance of the field of *connectionist* research. Based on the *von-Neumann-architecture*, today computers usually employ one, or a small number of central processors, working with high speed, and following a sequential program. Nevertheless, the tremendous growth in availability of cost-efficiency computing power enables to conveniently investigate also parallel computation strategies in simulation on sequential computers.

Often learning mechanisms are explored in computer simulations, but studying learning in a complex environment has severe limitations - when it comes to *action*. As soon as learning involves responses, acting on, or inter-acting with the environment, simulation becomes too easily unreal-

istic. The solution, as seen by many researchers is, that "learning must meet the real world". Of course, simulation can be a helpful technique, but needs realistic counter-checks in real-world experiments. Here, the field of robotics plays an important role.

The word "robot" is young. It was coined 1935 by the playwriter Karl Capek and has its roots in the Czech word for "forced labor". The first modern industrial robots are even younger: the "Unimates" were developed by Joe Engelberger in the early 60's. What is a robot? A robot is a mechanism, which is able to move in a given environment. The main difference to an ordinary machine is, that a robot is more versatile and multi-functional, and it can be programmed, or commanded to perform functions normally ascribed to humans. Its mechanical structure is driven by actuators which are governed by some controller according to an intended task. Sensors deliver the required feed-back in order to adjust the current trajectory to the commanded motion and task.

Robot tasks can be specified in various ways: e.g. with respect to a certain reference coordinate system, or in terms of desired proximities, or forces, etc. However, the robot is governed by its own actuator variables. This makes the availability of precise mappings from different sensory variables, physical, motor, and actuator values a crucial issue. Often these *sensorimotor mappings* are highly non-linear and sometimes very hard to derive analytically. Furthermore, they may change in time, i.e. drift by wear-and-tear or due to unintended collisions. The effective learning and adaption of the sensorimotor mappings are of particular importance when a precise model is lacking or it is difficult or costly to recalibrate the robot, e.g. since it may be remotely deployed.

Chapter 2 describes work done for establishing a hardware infrastructure and experimental platform that is suitable for carrying out experiments needed to develop and test robot learning algorithms. Such a laboratory comprises many different components required for advanced, sensor-based robotics. Our main actuated mechanical structures are an industrial manipulator, and a hydraulically driven robot hand. The perception side has been enlarged by various sensory equipment. In addition, a variety of hardware and software structures are required for command and control purposes, in order to make a robot system useful.

The reality of working with real robots has several effects:

- It enlarges the field of problems and relevant disciplines, and includes also material, engineering, control, and communication sciences.
- The time for gathering training data becomes a major issue. This includes also the time for preparing the learning set-up. In principle, the learning solution competes with the conventional solution developed by a human analyzing the system.
- The faced complexity draws attention also towards the efficient structuring of *re-usable building blocks* in general, and in particular for learning.
- And finally, it makes also technically inclined people appreciate that the complexity of biological organisms requires a rather long time of adolescence for good reasons;

Many learning algorithms exhibit stochastic, iterative adaptation and require a large number of training steps until the learned mapping is reliable. This property can also be found in the biological brain.

There is evidence, that learned associations are gradually enhanced by repetition, and the performance is improved by practice - even when they are learned insightfully. The *stimulus-sampling* theory explains the *slow learning* by the complexity and variations of environment (context) stimuli. Since the environment is always changing to a certain extent, many trials are required before a response is associated with a relatively complete set of context stimuli.

But there exits also other, *rapid* forms of associative learning, e.g. *"one-shot learning"*. This can occur by insight, or triggered by a particularly strong impression, by an exceptional event or circumstances. Another form is *"imprinting"*, which is characterized by a *sensitive period*, within which learning takes place. The timing can be even genetically programmed. A remarkable example was discovered by Konrad Lorenz, when he studied the behavior of chicks and mallard ducklings. He found, that they imprint the image and sound of their mother most effectively only from 13 to 16 hours after hatching. During this period a duckling possibly accepts another moving object as mother (e.g. man), but not before or afterwards.

Analyzing the circumstances when *rapid learning* can be successful, at least two important prerequisites can be identified:

- First, the importance and correctness of the learned *prototypical association* is clarified.
- And second, the correct *structural context* is known.

This is important in order to draw meaningful inferences from the prototypical data set, when the system needs to *generalize* in new, previously unknown situations.

The main focus of the present work are learning mechanisms of this category: *rapid learning* – requiring only a small number of training data. Our computational approach to the realization of such learning algorithm is derived form the "Self-Organizing Map" (SOM). An essential new ingredient is the use of a continuous parametric representation that allows a rapid and very flexible construction of manifolds with intrinsic dimensionality up to 4. . .8 i.e. in a range that is very typical for many situations in robotics.

This algorithm, is termed "Parameterized Self-Organizing Map" (PSOM) and aims at continuous, smooth mappings in higher dimensional spaces. The PSOM manifolds have a number of attractive properties.

We show that the PSOM is most useful in situations where the structure of the obtained training data can be correctly inferred. Similar to the SOM, the structure is encoded in the topological order of prototypical examples. As explained in chapter 4, the discrete nature of the SOM is overcome by using a set of basis functions. Together with a set of prototypical training data, they build a continuous mapping manifold, which can be used in several ways. The PSOM manifold offers auto-association capability, which can serve for completion of partial inputs and simultaneously mapping to multiple coordinate spaces.

The PSOM approach exhibits unusual mapping properties, which are exposed in chapter 5. The special construction of the continuous manifold deserves consideration and approaches to improve the mapping accuracy and computational efficiency. Several extensions to the standard formulations are presented in Chapter 6. They are illustrated at a number of examples.

In cases where the topological structure of the training data is known beforehand, e.g. generated by actively sampling the examples, the PSOM "learning" time reduces to an immediate construction. This feature is of particular interest in the domain of robotics: as already pointed out, here

the cost of gathering the training data is very relevant as well as the availability of adaptable, high-dimensional sensorimotor transformations.

Chapter 7 and 8 present several PSOM examples in the vision and the robotics domain. The flexible association mechanism facilitates applications: feature completion; dynamical sensor fusion, improving noise rejection; generating perceptual hypotheses for other sensor systems; various robot kinematic transformation can be directly augmented to combine e.g. visual coordinate spaces. This even works with redundant degrees of freedom, which can additionally comply to extra constraints.

Chapter 9 turns to the next higher level of *one-shot learning*. Here the learning of prototypical mappings is used to rapidly adapt a learning system to new context situations. This leads to a hierarchical architecture, which is conceptually linked, but not restricted to the PSOM approach.

One learning module learns the context-dependent skill and encodes the obtained *expertise* in a (more-or-less large) set of parameters or weights. A second *meta*-mapping module learns the association between the recognized context stimuli and the corresponding mapping expertise. The learning of a set of prototypical mappings may be called an *investment learning* stage, since effort is invested, to train the system for the second, the *one-shot* learning phase. Observing the context, the system can now adapt most rapidly by "mixing" the expertise previously obtained. This *mixture-of-expertise* architecture complements the *mixture-of-experts* architecture (as coined by Jordan) and appears advantageous in cases where the variation of the underlying model are continuous within the chosen mapping domain.

Chapter 10 summarizes the main points.

Of course the full complexity of learning and the complexity of real robots is still unsolved today. The present work attempts to make a contribution to a few of the many things that still can be and must be improved.

Chapter 2

The Robotics Laboratory

This chapter describes the developed concept and set-up of our robotic laboratory. It is aimed at the technically interested reader and explains some of the hardware aspects of this work.

A real robot lab is a testbed for ideas and concepts of efficient and intelligent controlling, operating, and learning. It is an important source of inspiration, complication, practical experience, feedback, and cross-validation of simulations. The construction and working of system components is described as well as ideas, difficulties and solutions which accompanied the development.
For a fuller account see (Walter and Ritter 1996c).

Two major classes of robots can be distinguished: *robot manipulators* are operating in a bounded three-dimensional workspace, having a fixed base, whereas *robot vehicles* move on a two-dimensional surface – either by wheels (mobile robots) or by articulated legs intended for walking on rough terrains. Of course, they can be mixed, such as manipulators mounted on a wheeled vehicle, or e.g. by combining several finger-like manipulators to a dextrous *robot hand*.

2.1 Actuation: The Puma Robot

The domain for setting up this robotics laboratory is the domain of manipulation and exploration with a 6 degrees-of-freedom *robot manipulator* in conjunction with a *multi-fingered robot hand*.

The compromise solution between a mature robot, which is able to

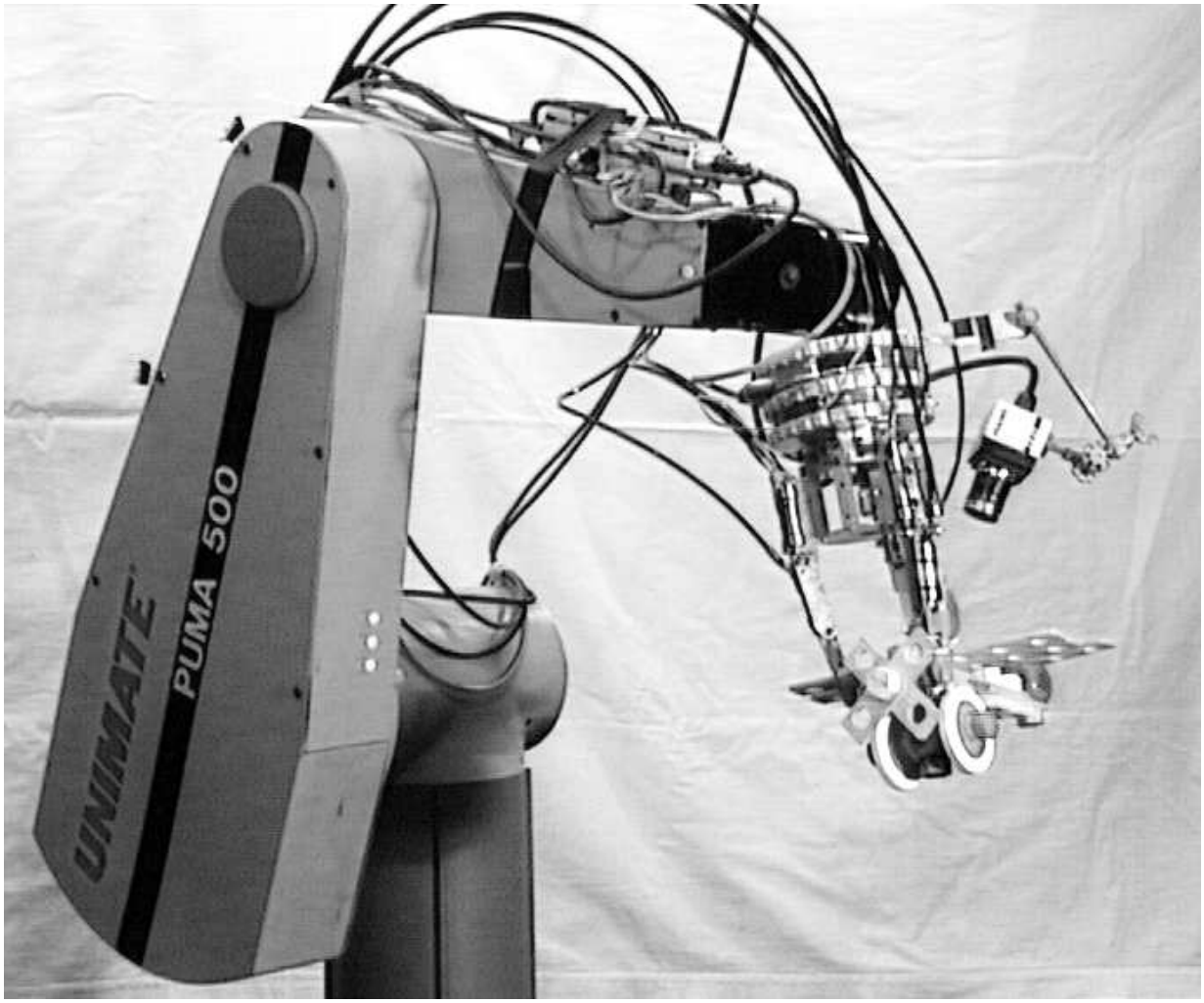


Figure 2.1: The six axes Puma robot arm with the TUM multi-fingered hand fixating a wooden "Baufix" toy airplane. The 6 D force-torque sensor (FTS) and the end-effector mounted camera is visible, in contrast to built-in proprioceptive joint encoders.

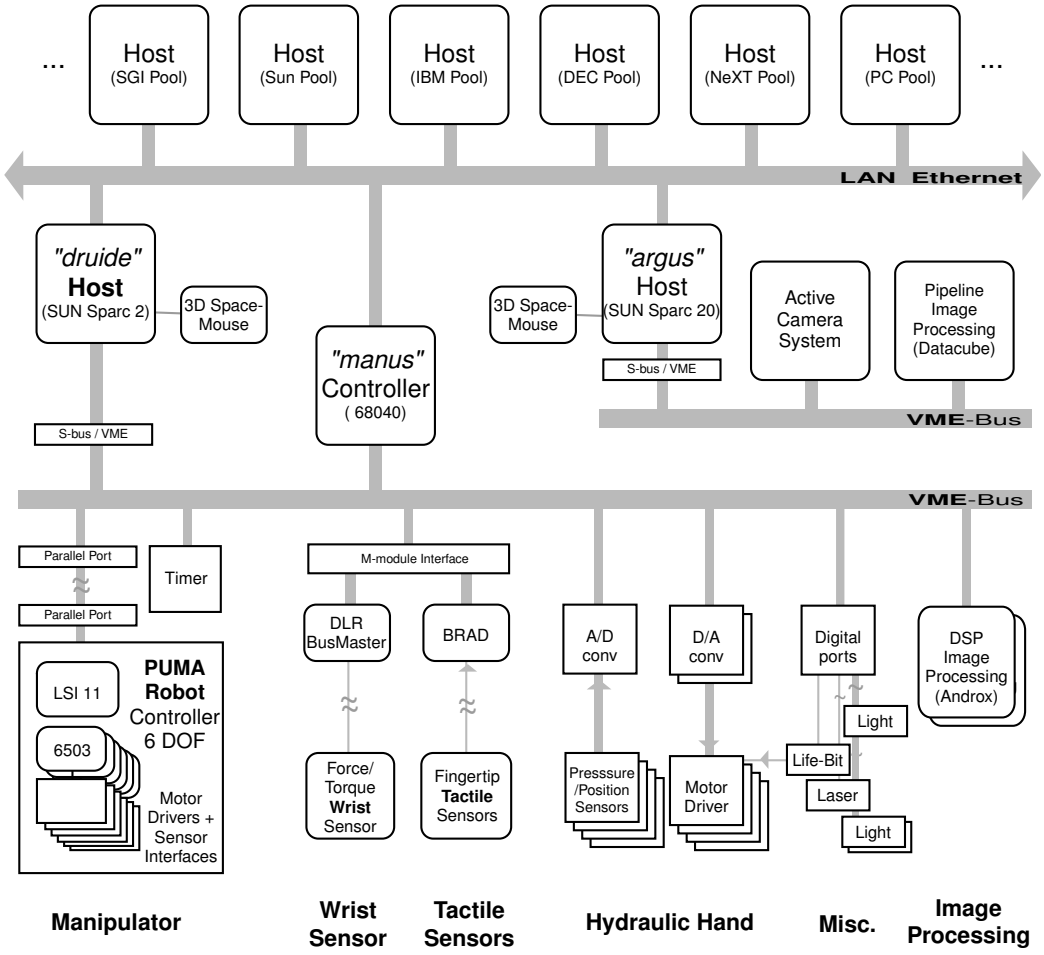


Figure 2.2: The Asymmetric Multiprocessing "Road Map". The main hardware "roads" connect the heterogeneous system components and lay ground for various types of communication links. The LAN Ethernet ("Local Area Network" with TCP/IP and max. throughput 10 Mbit/s) connects the pool of Unix computer workstations with the primary "robotics host" *"druide"* and the "active vision host" *"argus"* . Each of the two Unix SparcStation is bus master to a VME-bus (max 20 MByte/s, with 4 MByte/s S-bus link). *"argus"* controls the active stereo vision platform and the image processing system (Datacube, with pipeline architecture). *"druide"* is the primary host, which controls the robot manipulator, the robot hand, the sensory systems including the force/torque wrist sensor, the tactile sensors, and the second image processing system. The hand sub-system electronics is coordinated by the *"manus"* controller, which is a second VME bus master and also accessible via the Ethernet link. (Boxes with rounded corners indicate semi-autonomous sub-systems with CPUs enclosed.)

carry the required payload of about 3 kg *and* which can be turned into an *open, real-time* robot, was found with a Puma 560 Mark II robot. It is probably "the" classical industrial robots with six revolute joints. Its geometry and kinematics[1] is subject of standard robotics textbooks (Paul 1981; Fu, Gonzalez, and Lee 1987). It can be characterized as a medium fast (0.5 m/s straight line), very reliable, robust "work horse" for medium pay loads. The action radius is comparable to the human arm, but the arm is stronger and heavier (radius 0.9 m; 63 kg arm weight). The Puma Mark II controller comprises the power supply and the servo electronics for the six DC motors. They are controlled by six parallel microprocessors and coordinated by a DEC LSI-11 as central controller. Each joint microprocessor (Rockwell 6503) implements a digital PD controller, correcting the commanded joint position periodically. The *decoupled joint position control* operates with 1 kHz and originally receives command updates (setpoints) every 28 ms by the LSI-11.

In the standard application the Puma is programmed in the interpreted language VAL II, which is considered a flexible programming language by industrial standards. But running on the main controller (LSI-11 processor), it is not capable of handling high bandwidth sensory input itself (e.g., from a video camera) and furthermore, it does not support flexible control by an auxiliary computer. To achieve a tight real-time control directly by a Unix workstation, we installed the software package RCI/RCCL (Hayward and Paul 1986; Lloyd 1988; Lloyd and Parker 1990; Lloyd and Hayward 1992).

The acronym RCI/RCCL stands for *Real-time Control Interface* and *Robot Control C Library*. The package provides besides the reprogramming of the robot controller a library of commands for issuing high-level motion commands in the C programming language. Furthermore, we patched the Sun operating system OS 4.1 to sufficient real-time capabilities for serving a reliable control process up to about 200 Hz. *Unix* is a multitasking operating system, sequencing several processes in short time slices. Initially, Unix was not designed for real-time control, therefore it provides a regular process only with timing control on a coarse time scale. But *real-time* processing requires, that the system reliably responds within a certain time frame. RCI succeeded here by anchoring the synchronous trajectory control task

[1] Designed by Joe Engelberger, the founder of Unimation, sometimes called the father of robotics. Unimation was later sold to Westinghouse Inc., AEG and last to Stäubli.

(a special thread) at a special device driver serving the interrupts from a timer card. The *control task* is thus running independently and outside the *planning task*. By this means, sensory information (e.g. camera or force sensors) can be processed and feedback in a very effective and convenient manner.

For example, by default our DLR 6 D wrist sensor is read out about the currently exerted force and torque vector (3+3=6 D) between the robot arm and the robot hand (Fig. 2.1, 2.4). The DLR Force-Torque-Sensor (FTS) was developed by the robotics group of Prof. Hirzinger of the DLR, Oberpfaffenhofen, and is a spin-off from the ROTEX Spacelab mission D2 (Hirzinger, Brunner, Dietrich, and Heindl 1994). As indicated in Fig. 2.2, the FTS is an micro-controller based sensory sub-system, which communicates via a special field-bus with the VME-bus.

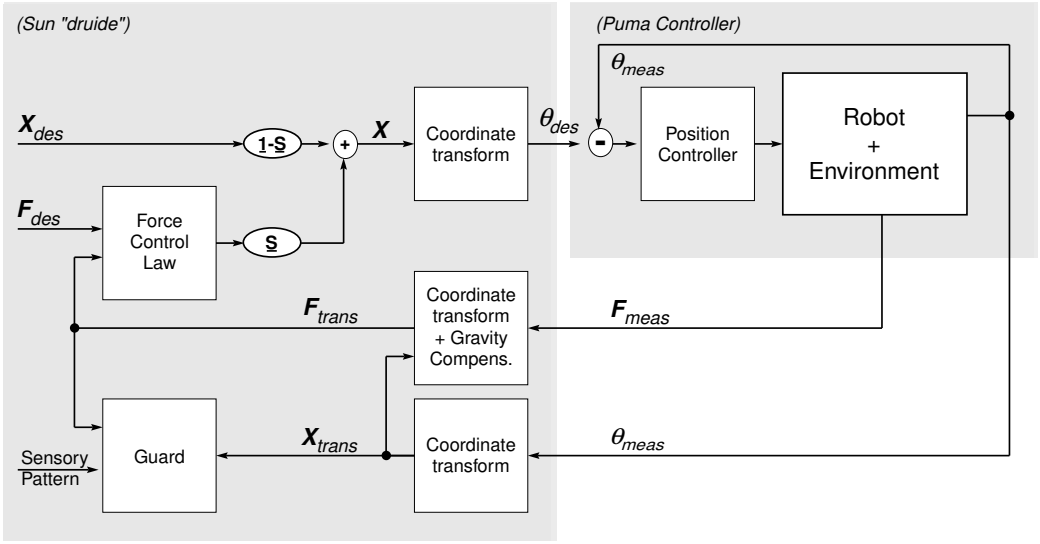


Figure 2.3: A two-loop control scheme for the mixed force and position control. The inner, fast loop runs on the joint micro controller within the Puma controller, the outer loop involves the control task on *"druide"*.

The resulting robot control system allows us to implement hybrid control architectures using the position control interface. This includes multi-sensor compliant motions with mixed force controlled motions as well as controlling an artificial spring behavior. The main restriction is the difficulty in controlling forces with high robot speeds. High speed motions

with environment interaction need quick response and therefore require, a very high frequency of the digital force control loop. The bottleneck is given by the Puma controller structure. The realizable force control includes a fast inner position loop (joint micro controller) with a slower outer force loop (involving the Sun *"druide"*). But still, by generating the robot trajectory setpoints on the external Sun workstation, we could double the control frequency of VAL II and establish a stable outer control loop with 65 Hz.

Fig. 2.3 sketches the two-loop control scheme implemented for the mixed force and position control of the Puma. The inner, fast loop runs on the joint micro controller within the Puma controller, the outer loop involves the control task on the Sun workstation *"druide"*. The desired position $\mathbf{X}_{des}$ and forces $\mathbf{F}_{des}$ are given for a specified coordinate system (here written as generalized 6 D vectors: position and orientation in roll, pitch, yaw (see also Fig. 7.2 and Paul 1981) $\mathbf{X}_{des} = (p_x, p_y, p_z, \phi, \theta, \psi)$ and generalized force $\mathbf{F}_{des} = (f_x, f_y, f_z, m_x, m_y, m_z)$). The control law transforms the force deviation into a desired position. The diagonal selection matrix elements in $\mathbf{S}$ choose force controls (if 1) or position control (if 0) for each axis, following the idea of Cartesian sub-space control[2]. The desired position is transformed and signaled to the joint controllers, which determine appropriate motor power commands. The results of the robot - environment interaction $\mathbf{F}_{meas}$ is monitored by the force-torque sensor measurement and transformed to the net acting force $\mathbf{F}_{trans}$ after the gravity force computation. The guard block checks on specified sensory patterns, e.g., force-torque ranges for each axes and whether the robot is within a safe-marked work space volume. Depending on the desired action, a suitable controller scheme and sets of parameters must be chosen, for example, $\mathbf{S}$, gains, stiffness, safe force/position patterns). Here the efficient handling and access of parameter sets, suitable for run-time adaptation is an important issue.

[2]Examples for suitable selection matrices are: $\mathbf{S}$=diag(0,0,1,0,0,0) for a compliant motion with a desired force in z direction, or $\mathbf{b}$ S=diag(0,0,1,1,1,0) for aligning two flat surfaces (with surface normal in z). A free translation and z-rotational follow controller in Cartesian space can be realized with $\mathbf{S}$=diag(1,1,1,0,0,1). See (Mason and Salisbury 1985; Schutter 1986; Dücker 1995).

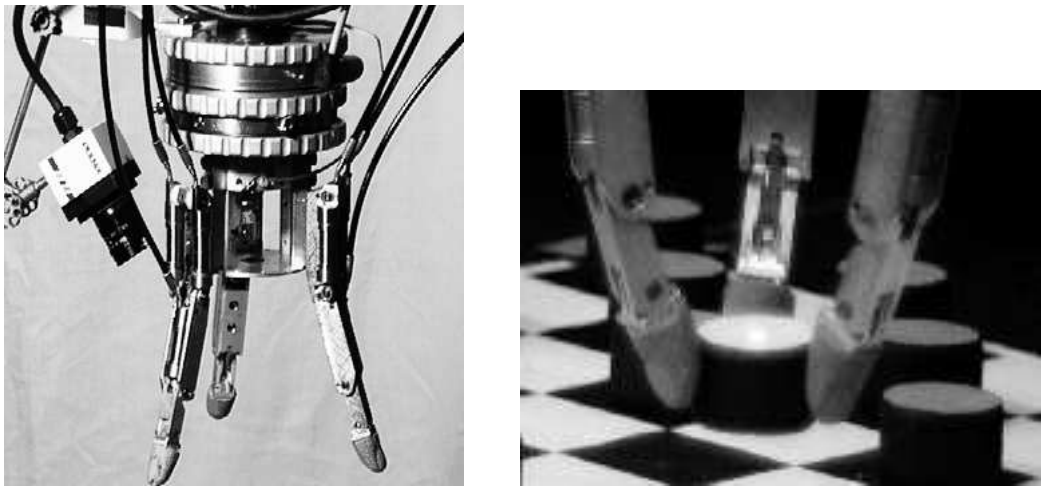


Figure 2.4: The endeffector. *(left:)* Between the arm and the hydraulic hand, the cylinder shaped FTS device can measure current 6 D force torque values. The three finger modules are mounted here symmetrically at the 12 sided regular prism base. On the left side, the color video camera looks at the scene from an end-effector fixed position. Inside the flat palm, a diode laser is directed in tool axis, which allows depth triangulation in the viewing angle of the camera.

2.2 Actuation: The Hand "Manus"

For the purpose of studying dextrous manipulation tasks, our robot lab is equipped with an hydraulic robot hand with (up to) four identical 3-DOF fingers modules, see Fig. 2.4. The hand prototype was developed and built by the mechanical engineering group of Prof. Pfeiffer at the Technical University of Munich ("TUM-hand"). We received the final hand prototype comprising four completely actuated fingers, the sensor interface, and motor driver electronics. The robot finger's design and its mobility resembles that of the human index finger, but scaled up to about 110 %.

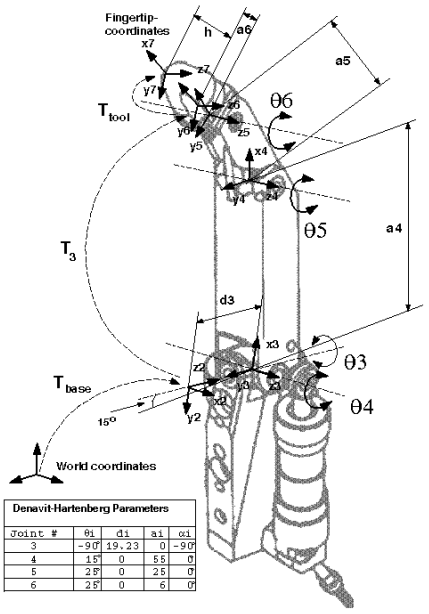


Figure 2.5: The kinematics of the TUM robot finger. The cardanic base joint allows 15° sidewards gyring (θ_3) and full adduction (θ_4) together with two coupled joints ($\theta_5 = \theta_6$). (after Selle 1995)

Fig. 2.5 displays the kinematics of one finger. The particular kinematic mapping (from piston location to joint angles and Cartesian position) of the cardanic joint configuration is very hard to invert analytically. Selle (1995) describes an iterative numerical procedure. This sensorimotor mapping is a challenging task for a learning algorithm. In section 8.1 we will take up this problem and present solutions which achieve good accuracy with a fairly small number of training examples.

2.2.1 Oil model

The finger joints are driven by small, spring loaded, hydraulic cylinders, which connect each actuator to the base station by a oil hose. In contrast to the more standard hydraulic system with a central power supply and valve controlled bi-directional powered cylinder, here, each finger cylinder is one-way powered from a corresponding cylinder at the base station. Unfortunately, the finger design does not foresee integrated sensors directly at the fingers.

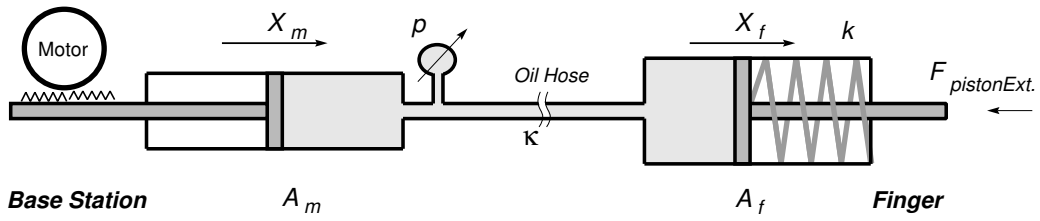


Figure 2.6: The hydraulic oil system.

The control system has to rely on indirect feedback sensing through the oil system. Fig. 2.6 displays the location of the two feedback sensors. In each degree of freedom (i) the piston position x_m of the motor cylinder (linear potentiometer) and (ii) the pressure p in the closed oil system (membrane sensor with semi-conductor strain-gauge) is measured at the base station. The long oil hose is not perfectly stiff, which makes this oil system component significantly expandable (4 m, large surface to volume ratio). This bears the advantage of a naturally compliant and damped system but bears also the disadvantage, that even pure position control must consider the force - position coupled oil model (Menzel et al. 1993; Selle 1995; Walter and Ritter 1996c).

2.2.2 Hardware and Software Integration

The modular concept of the TUM-hand includes its interface electronics. Each finger module has its separate motor servo electronics and sensor amplifiers, which we connected to analog converter cards in the VME bus system as illustrated in the lower right part of Fig. 2.2. The digital hand control process is running at *"manus"*, a VME based embedded 68040 pro-

cessor board. Following the example of RCCL, the "Manus Control C Library" (MCCL) was developed and implemented (Rankers 1994; Selle 1995). To facilitate an arm-hand unified planning level, the Unix workstation *"druide"* is set up to issue finger motion (piston, joint, or Cartesian position) and force control requests to the *"manus"* controller (Fig. 2.2).

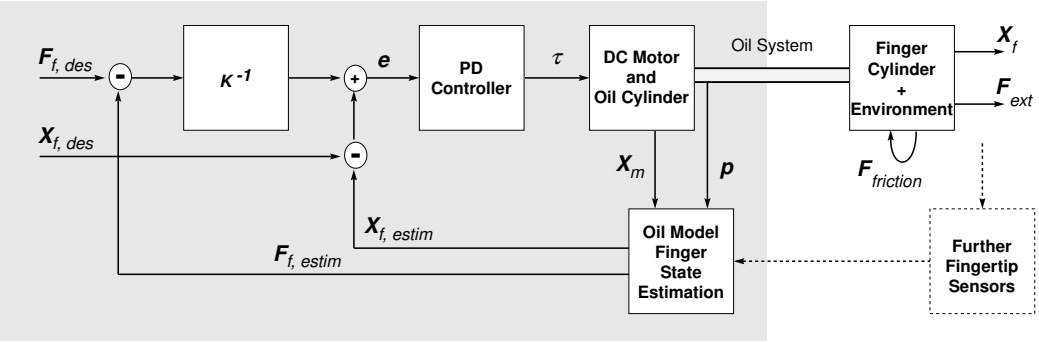


Figure 2.7: A control scheme for the mixed force and position control running on the embedded VME-CPU *"manus"*. The original robot hand design allows only indirect estimation of the finger state utilizing a model of the oil system. Certain kinds of influences, especially friction effects require extra information sources to be satisfyingly accounted for – as for example tactile sensors, see Sec. 2.3.

The achieved performance in dextrous finger control is a real challenge and led to the development of a simulator package for a more detailed study of the oil system (Selle 1995). The main sources of uncertainty are friction effects in combination with the lack of direct sensory feedback. As illustrated in Fig. 2.7, extra sensory information is required to fill this gap. Particularly promising are different kinds of tactile sense organs. The human skin uses several types of neural receptors, sensitive to static and dynamic pressure in a remarkable versatile manner.

In the following section extensions to the robot's senses are described. They are the prerequisite for more intelligent, semi-autonomous robotic systems. As already mentioned, todays robots are usually restricted to the proprioceptors of their actuator positions. For environment interaction two categories can be distinguished: *(i)* remote senses, which are mediated, e.g. by light, and *(ii)* direct senses in case parts of the robot are in contact. Measurements to obtain force-torque information are the FTS-wrist sensor and the finger state estimation as mentioned above.

2.3 Sensing: Tactile Perception

Despite the explained importance of good sensory feedback sub-systems, no suitable tactile sensors are commercially available. Therefore we focused on the design, construction and making of our own multi-purpose, compound sensor (Jockusch 1996). Fig. 2.8 illustrates the concept, achieved with two planar film sensor materials: *(i)* a slow piezo-resistive FSR material for detection of the contact force and position, and *(ii)* a fast piezoelectric PVDF foil for incipient slip detection. A specific consideration was the affordable price and the ability to shape the sensors in the particular desired forms. This enables to seek high spatial coverage, important for fast and spatially resolved contact state perception.

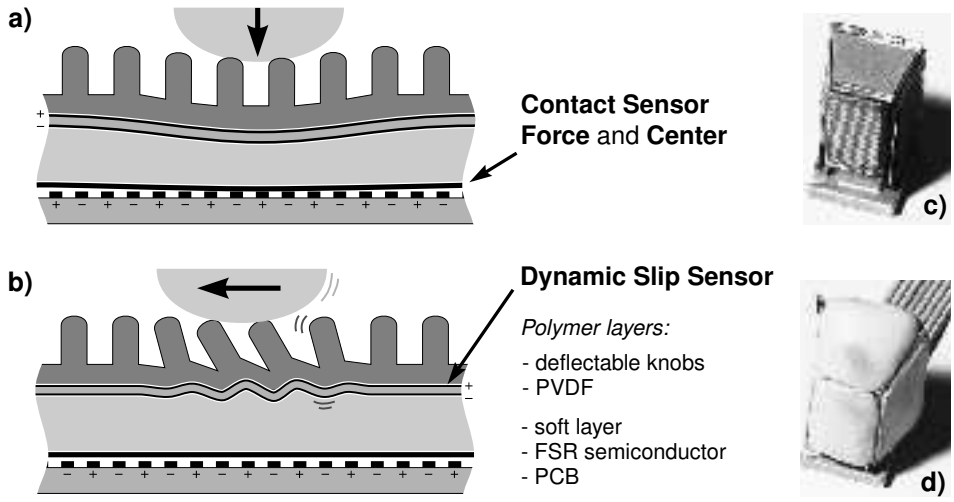


Figure 2.8: The sandwich structure of the multi-layer tactile sensor. The FSR sensor measures normal force and contact center location. The PVDF film sensor is covered by a thin rubber with a knob structure. The two sensitive layers are separated by a soft foam layer transforming knob deflection into local stretching of the PVDF film. By suitable signal conditioning, slippage induced oscillations can be detected by characteristic spike trains. *(c–d:)* Intermediate steps in making the compound sensor.

Fig. 2.8cd shows the prototype. Since the kinematics of the finger involves a moving contact spot during object manipulation, an important requirement is the continuous force sensitivity during the rolling motion

on an object surface, see Jockusch, Walter, and Ritter (1997).

Efficient system integration is provided by a dedicated, 64 channel signal pre-conditioning and collecting micro-computer based device, called "MASS" (= *Multi channel Analog Signal Sampler*, for details see Jockusch 1996). MASS transmits the configurable set of sensor signals via a high-speed link to its complementing system "BRAD" – the **Buffered Random Access Driver** hosted in the VME-bus rack, see Fig. 2.2. BRAD writes the time-stamped data packets into its shared memory in cyclic order. By this means, multiple control and monitor processes can conveniently access the most recent sensor data tuple. Furthermore, entire records of the recent history of sensor signals are readily available for time series analysis.

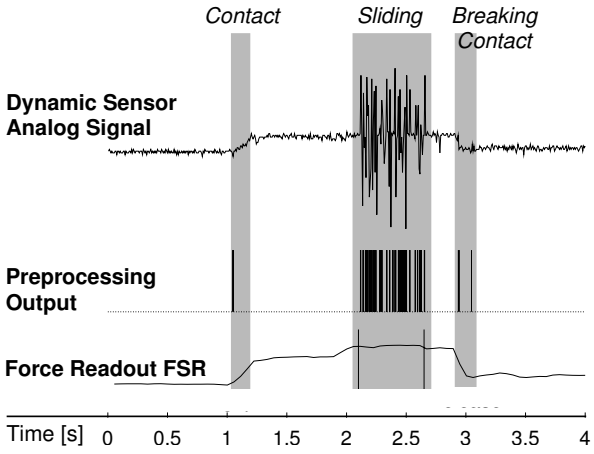


Figure 2.9: Recordings from the raw and pre-processed signal of the dynamic slippage sensor. A flat wooden object is pressed against the sensor, and after a short rest tangentially drawn away. By band-pass filtering the slip signal of interest can be extracted: The middle trace clearly shows the sudden contact and the slippage phase. The lower trace shows the force values obtained from the second sensor.

Fig. 2.9 shows first recordings from the sensor prototype. The raw signal of the PVDF sensors (upper trace) is bandpass filtered and thresholded. The obtained spike train (middle trace) indicates the critical, characteristic signal shapes. The first contact with a flat wood piece induces a short signal. Together with the simultaneously recorded force information (lower trace) the interesting phases can be discriminated.

These initial results from the new tactile sensor system are very promising. We expect to *(i)* fill the present gap in proprioceptive sensory information on the oil cylinder friction state and therefore better finger fine control; *(ii)* get fast contact state information for task-oriented low-level grasp reflexes; *(iii)* obtain reliable contact state information for signaling higher-level semi-autonomous robot motion controllers.

2.4 Remote Sensing: Vision

In contrast to the processing of force-torque values, the information gained by the image processing system is of very high-dimensional nature. The computational demands are enormous and require special effort to quickly reduce the huge amount of raw pixel values to useful task-specific information.

Our vision related hardware currently offers a variety of CCD cameras (color and monochrome), frame grabbers and two specialized image processors systems, which allow rapid pre-processing. The main subsystems are *(i)* two Androx ICS-400 boards in the VME bus system of *"druide"*(see Fig. 2.2), and *(ii)* A MaxVideo-200 with a DigiColor frame grabber extension from Datacube Inc.

Each system allows simultaneous frame grabbing of several video channels (Androx: 4, Datacube: 3-of-6 + 1-of-4), image storage, image operations, and display of results on a RGB monitor. Image operations are called by library functions on the Sun hosts, which are then scheduled for the *parallel processors*. The architecture differs: each Androx system uses four DSP operating on shared memory, while the Datacube system uses a collection of special pipeline processors working easily in frame rate (max 20 MByte/s). All these processors and crossbar switches are register programmable via the VME bus. Fortunately there are several layers of library calls, helping to organize the pipelines and their timely switching (by pipe altering threads).

Specially the latter machine exhibits high performance if it is well adapted to the task. The price for the speed is the sophistication and the complexity of the parallel machines and the substantial lack of debugging information provided in the large, parallel, and fast switching data streams. This lack of debug tools makes code development somehow tedious.

However, the tremendous growth in *general-purpose computing power* allows to shift already the entire exploratory phase of vision algorithm development to general-purpose high-bandwidth computers. Fig. 2.2 exposes various graphic workstations and high-bandwidth server machines at the LAN network.

2.5 Concluding Remarks

We described work invested for establishing a versatile robotics hardware infrastructure (for a more extended description see Walter and Ritter 1996c). It is a testbed to explore, develop, and evaluate ideas and concepts. This investment was also prerequisite of a variety of other projects, e.g. (Littmann et al. 1992; Kummert et al. 1993a; Kummert et al. 1993b; Wengerek 1995; Littmann et al. 1996).

An experimental robot system comprises many different components, each exhibiting its own characteristics. The integration of these sub-systems requires quite a bit of effort. Not many components are designed as intelligent, open sub-systems, rather than systems by themselves.

Our experience shows, that good design of re-usable building blocks with suitably standardized software interfaces is a great challenge. We find it a practical need in order to achieve rapid experimentation and economical re-use. An important issue is the sharing and interoperating of robotics resources via electronic networks. Here the hardware architecture must be complemented by a software framework, which complies to the special needs of a complex, distributed robotics hardware. Efforts to tackle this problem are beyond the scope of the present work and therefore described elsewhere (Walter and Ritter 1996e; Walter 1997).

In practice, the time for gathering training data is a significant issue. It includes also the time for preparing the learning set-up, as well as the training phase. Working with robots in reality clearly exhibits the need for those learning algorithms, which work efficiently also with a small number of training examples.

Chapter 3

Artificial Neural Networks

This chapter discusses several issues that are pertinent for the PSOM algorithm (which is described more fully in Chap. 4). Much of its motivation derives from the field of neural networks. After a brief historic overview of this rapidly expanding field we attempt to order some of the prominent network types in a taxonomy of important characteristics. We then proceed to discuss learning from the perspective of an approximation problem and identify several problems that are crucial for rapid learning. Finally we focus on the so-called "Self-Organizing Maps", which emphasize the use of topology information for learning. Their discussion paves the way for Chap. 4 in which the PSOM algorithm will be presented.

3.1 A Brief History and Overview of Neural Networks

The field of artificial neural networks has its roots in the early work of McCulloch and Pitts (1943). Fig. 3.1a depicts their proposed model of an idealized biological neuron with a binary output. The neuron "fires" if the weighted sum $\sum_j w_{ij} x_j$ (synaptic weights w) of the inputs x_j (dendrites) reaches or exceeds a threshold w_i. In the sixties, the **Adaline** (Widrow and Hoff 1960), the **Perceptron**, and the **Multi-Layer Perceptron** ("MLP", see Fig. 3.1b) have been developed (Rosenblatt 1962). Rosenblatt demonstrated the convergence conditions of an early *learning algorithm* for the one-layer Perceptron. The learning algorithm described a way of iteratively changing the weights.

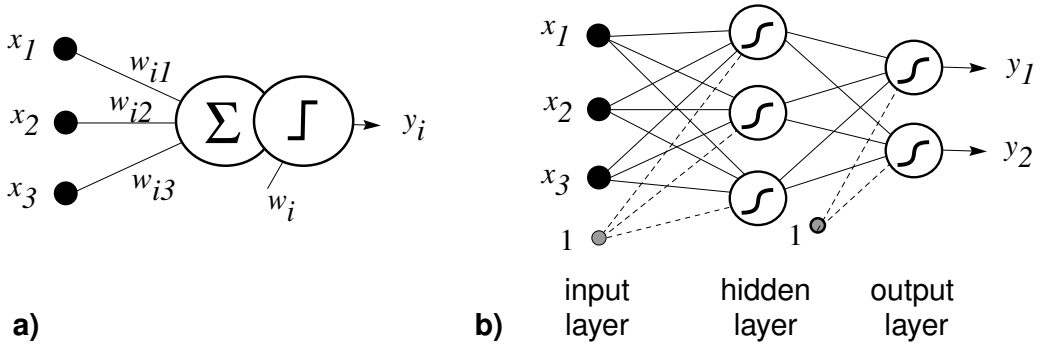


Figure 3.1: *(a)* The **McCulloch-Pitts neuron** "fires" (output y_i=1 else 0) if the weighted sum $\sum_j w_{ij}x_j$ of its inputs x_j reaches or exceeds a threshold w_i. If this binary threshold function is generalized to a non-linear sigmoidal transfer function $g(\sum_j w_{ij}x_j - w_i)$ (also called *activation*, or *squashing function*, e.g. $g(\cdot)$=tanh$(\cdot)$), the neuron becomes a suitable processing element of the standard *(b)* **Multi-Layer Perceptron (MLP)**. The input values x_i are made available at the "input layer". The output of each neural unit is *feed forward* as input to all neurons of the next layer. In contrast to the standard or single-layer perceptron, the MLP has typically one or several, so-called *hidden layers* of neurons between the input and the output layer.

In (1969) Minsky and Papert showed that certain classes of problems, e.g. the "exclusive-or" problem, cannot be learned with the simple perceptron. They doubted that learning rules could be found for computationally more powerful multi-layered networks and recommended to focus on the symbolic oriented learning paradigm, today called *artificial intelligence* ("AI"). The research funding for artificial neural networks was cut, and it took twenty years until the field became viable again.

An important stimulus for the field was the multiple discovery of the **error back-propagation** algorithm. Its has been independently invented in several places, enabling iterative learning for multi-layer perceptrons (Werbos 1974, Rumelhart, Hinton, and Williams 1986, Parker 1985). The MLP turned out to be a *universal approximator*, which means that using a sufficient number of hidden units, any function can be approximated arbitrarily well. In general two hidden layers are required - for *continuous* functions one layer is sufficient (Cybenko 1989, Hornik et al. 1989). This property is of high theoretical value, but does not guarantee efficiency of any kind.

Other important developments where made: e.g. v.d. Malsburg and Willshaw (1977, 1973) modeled the ordered formation of connections between neuron layers in the brain. A strongly related, more formal algorithm was formulated by Kohonen for the development of a topographically ordered map from a general space of input stimuli to a layer of abstract neurons. We return to Kohonen's work later in Sec. 3.7.

Hopfield (1982, 1984) contributed a famous model of the content-addressable **Hopfield network**, which can be used e.g. as *associative memory* for image completion. By introducing an *energy function*, he opened the mathematical toolbox of statistical mechanics to the class of *recurrent neural networks* (mean field theory developed for the physics of magnetism). The **Boltzmann machine** can be seen as a generalization of the Hopfield network with *stochastic* neurons and symmetric connection between the neurons (partly visible – input and output units – and partly hidden units). "Stochastic" means that the input influences the probability of the two possible output states ($y \in \{-1, +1\}$) which the neuron can take (*spin glass* like).

The **Radial Basis Function Networks ("RBF")** became popular in the connectionist community by Moody and Darken (1988). The RFB belong to the class of local approximation schemes (see p. 33). Similarities and

differences to other approaches are discussed in the next sections.

3.2 Network Characteristics

Meanwhile, a large variety of neural network types have emerged. In the following we present a (certainly incomplete) taxonomic ordering and point out several distinguishable axes:

Supervised versus Unsupervised and Reinforcement Learning: In supervised learning paradigm, the training input signal is given with a pairing output signal from a supervisor or teacher knowing the correct answer. Unsupervised networks (e.g. competitive learning, vector quantization, SOM, see below) draw information from redundancies in the input data distribution.

An intermediate form is the *reinforcement learning*. Here the system receives a "reward" or "quality" signal, indicating whether the network output was more or less successful. A major problem is the meaningful **credit assignment** to the responsible network parts. The *structural* problem is extended by the *temporal* credit assignment problem if the quality signal is delayed and a sequence of decisions contributed to the overall result.

Feed-forward versus Recurrent Networks: In feed-forward networks the information flow is unidirectional from the input to the output layer. In contrast, recurrent networks also connect neuron outputs back as additional feedback inputs. This enables a network intern dynamic, controlled by the given input and the learned network characteristics.

A typical application is the *associative memory*, which can iteratively recall incomplete or noisy images. Here the recurrent network dynamics is built such, that it leads to a settling of the network. These relaxation endpoints are fix-points of the network dynamic. Hopfield (1984) formulated this as an energy minimization process and introduced the statistical methods known e.g. in the theory of magnetism. The goal of learning is to place the set of *point attractors* at the desired location. As shown later, the PSOM approach will uti-

lize a form of recurrent network dynamic operating on a *continuous attractor manifold*.

Hetero-association and Auto-association: The ability to evaluate the given input and recall the desired output is also called *association*. *Hetero-association* is the common (one-way) input to output mapping (function mapping). The capability of *auto-association* allows to infer different kinds of desired outputs on the basis of an incomplete pattern. This enables the learning of more general relations in contrast to function mapping.

Local versus Global Representation: For a network with local representation, the output of a certain input is produced only by a *localized part* of the network (which is pin-pointed by the notion of a "grandmother cell"). Using global representation, the network output is assembled of information distributed over the entire network. A global representation is more *robust against single neuron failures*. Here, as a result the network performance *degrades gracefully*, like the biological brain usually does. The local representation of knowledge is easier to interpret and not endangered by the so-called *"catastrophic interference"*, see "on-line learning" below.

Batch versus Incremental Learning: Calculating the network weight updates under consideration of all training examples at once is called "batch-mode" learning. For a linear network, the solution of this learning task can be shown to be equivalent to finding the pseudo-inverse of a matrix, that is formed by the training data. In contrast, incremental learning is an iterative weight update that is often based on some gradient descent for an "error function". For good convergence this often requires the presentation of the training examples in a stochastic sequence. Iterative learning is usually more efficient, particularly w.r.t. memory requirements.

Off-line versus On-line Learning and Interferences: Off-line learning allows easier control of the training procedure and validity of the data (identification of *outliers*). On-line, incremental learning is very important, since it provides the ability to dynamically adapt to new or changing situations. But it generally bears the danger of undesired "interferences" (*"after-learning"* or *"life-long learning"*).

Consider the case of a network, which is already well trained with the data set A. When a new data set B gets available, the knowledge about "skill" A can be deteriorated (interference) mainly in the following ways:

(i) due to re-allocation of the computational resources to new mapping domains the old skill (A) becomes less accurate ("*stability – plasticity*" problem).

(ii) Further data sets A and B might be inconsistent due to a change in the mapping task and require a re-adaptation.

(iii) Beyond these two principal, problem-immanent interferences, a *global* learning process can cause **"catastrophic interference"**: when the weight update to new data is global, it is hard to control, how this influences knowledge previously learned. A popular solution is to memorize the old dataset A, and retrain the network based on the merged dataset A and B.

One of the main challenges in on-line learning is the proper control of the current context. It is crucial in order to avoid wrong generalization for other contexts - analog to the human "traumatic experiences" (see also localized representations above, mixture-of-experts below and Chap. 9 for the problem of context oriented learning).

Fixed versus adaptable network structures As pointed out before, the suitable network (model) structure has significant influence on the efficiency and performance of the learning system. Several methods have been proposed for tackling the combined problem of adapting the network weights and dynamically deciding on the structural adaptation (e.g. growth) of the network (additive models). Strategies on selecting the network size will be later discussed in Sec. 3.6.

For a more complete overview of the field of neural networks we refer the reader to the literature, e.g. (Anderson and E. Rosenfeld 1988; Hertz, Krogh, and Palmer 1991; Ritter, Martinetz, and Schulten 1992; Arbib 1995).

3.3 Learning as Approximation Problem

In this section *learning* tasks are considered from the perspective of basic representation types and their relation to methods of other disciplines.

Usually, the *process of learning* is based on a certain amount of apriori knowledge and a set of training examples. Its *goal* is two-fold:

- the learner should be able to recognize the re-occurance of a previously seen situation (stimuli or input) and **associate** the correct answer (response or output) as learned before;
- in new, previously un-experienced situations, the learner should **generalize** its knowledge and infer the answer appropriately.

The primary problem of learning is to find an appropriate *representation* of the learned knowledge or skill: its input (stimuli) and output (responses). The reason is rather simple and fundamental: no system can learn, what it cannot represent.

We can distinguish three basic types of task describing variables x:

1. the **orderable, continuous valued** representations (e.g. length, speed, temperature, force) with a defined order relation ($x_i < x_j$) and an existing distance metric $|x_i - x_j|$;
2. a **periodic or circular** variable representation (e.g. azimuth angles, hour of the day) with defined distance, but without a clear order relation ("wrap-around"; is Monday before or after Saturday?);
3. the **symbolic and categorical** representation $x \in \{c_1, \ldots, c_k\}$, like cities, attributes and generally names. Here, nor an order, neither a distance relation between x_i and x_j does exist, just the binary equivalence relation is defined ($x_i = x_j$, or $x_i \neq x_j$).

Sometimes variables are represented depending on the context, e.g. "red" color may get coded as category, as circular hue value, or as orderable location in the color triangle.

A desired skill can be modeled as the mapping between an input space and an output space. The input space (later notated X^{in}) captures all relevant system (observable and non-observed) variables. The problem of learning such a mapping is then equivalent to the problem of synthesizing an *associative memory* (Poggio and Girosi 1990). The appropriate output is retrieved when the input is presented and the system *generalizes* when a *new* input is presented. Different frameworks of this problem depend strongly on the type of input and output variables.

Most methods prefer orderable, variables as input variables type (neural nets), some others prefer categorical variables (artificial intelligence and machine learning approaches). Depending on the type of output variables different frameworks offer methods called regression, approximation, classification, system identification, system estimation, pattern recognition, or learning. Tab. 3.1 compares names for learning task, common in different domains of research.

Output Type vs. **Framework**	**Continuous Values** Orderable Variables	**Symbolic Values** Categorical Variables
Neural Networks	Learning	Learning
Machine Learning	Sub-symbolic & Fuzzy Learning	Learning
Mathematics	Approximation	Quantization
Statistics	Regression	Classification
Engineering	System Identification & Estimation	Pattern Recognition

Table 3.1: Creating and refining a model in order to solve a learning task has various common names in different disciplines.

In the following we mainly focus on the variable type continuous and orderable. It can be considered as the most general case, since periodic variables (2) can transformed by the trick of mapping the phase information into a pair of sine and cosine values (of course the topology is unchanged). Categorical output values (3) are often prepared by a competitive component which selects the dominating component in a multi-dimensional output ("winner takes all"). It is interesting to notice, that *Fuzzy Systems* work the opposite way. Continuous valued inputs are examined on their probability to belong to a particular class (fuzzy membership). All combinations are propagated through a symbolic rule set (if-then-else type) and the output "de-fuzzificated" into a continuous output. The attractive point is the simplicity how to insert categorical "expert knowledge" into the system.

We consider a system that generates data and is presumed to be described by the multivariate function $f(\mathbf{x})$ (possible perturbed by noise). With continuous valued variables, the *learning task* is to model the system by the function $F(\mathbf{w}, \mathbf{x})$ that can serve as a *reasonable approximation* of f over the domain D of interest. The regression is based on a set of given

training or design points D_{train} within the domain of interest $D_{train} \subset D \subset X$. $\mathbf{x}$ denotes the input variable set $\mathbf{x} = \{x_1, \ldots, x_d\} \in X$, $\mathbf{w}$ a parameter set $\mathbf{w} = (w_1, w_2, \ldots)$.

A good measure for the quality of the approximation will depend on the intended task. However, in most cases accuracy is of importance and is measured employing a distance function $dist(f(\mathbf{x}), F(\mathbf{w}, \mathbf{x}))$ (e.g. L2 norm). The lack of accuracy or *lack-of-fit* $LOF(F)$ is often defined by the expected error

$$LOF(F, D) = \langle dist\left(f(\mathbf{x}), F(\mathbf{w}, \mathbf{x})\right)\rangle_D \tag{3.1}$$

the $dist$ function averaged over the domain of interest D. The approximation problem is to determine the parameter set $\mathbf{w}^*$, that minimizes $LOF(F, D)$. The ultimate solution depends strongly on F. If it exists, it is called "best approximation" (see e.g. Davis 1975; Poggio and Girosi 1990; Friedman 1991).

Summarizing, several main problems in building a learning system can be distinguished:

(i) encoding the problem in a suitable representation $\mathbf{x}$;

(ii) finding a suitable approximation function F;

(iii) choosing the algorithm to find optimal values for the parameters W;

(iv) the problem of efficiently implementing the algorithm.

The proceeding chapter 4 will present the PSOM approach with respect to *(ii)–(iv)*. Numerous examples for *(i)* are presented in the later chapters. The following section discusses several common methods for *(ii)*

3.4 Approximation Types

In the following we consider some prominent examples of approximating functions $F(\mathbf{w}, \mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$ – for the moment simplified to one-dimensional outputs.

The classical linear case is

$$F(\mathbf{w}, \mathbf{x}) = \mathbf{w} \cdot \mathbf{x} \tag{3.2}$$

the dot product with input $\mathbf{x}$, usually augmented by a constant one. This *linear regression* scheme corresponds to a linear, single layer network, compare Fig. 3.1.

The classical approximation scheme is a linear combination of a suitable set of basis functions $\{B_i\}$ on the original input $\mathbf{x}$

$$F(\mathbf{w}, \mathbf{x}) = \sum_{i=1}^{m} w_i \, B_i(\mathbf{x}) \tag{3.3}$$

and corresponds to a network with one hidden layer. This representation includes global polynomials (B_i are products and powers of the input components; compare polynomial classifiers), as well as expansions in series of orthogonal polynomials and splines.

Nested sigmoids schemes correspond to the *Multi-Layer Perceptron* and can be written as

$$F(\mathbf{w}, \mathbf{x}) = g\left(\sum_k u_k \, g\left(\sum_j v_{kj} \, g\left(\cdots g\left(\sum_i w_{ji} \, x_i\right)\cdots\right)\right)\right) \tag{3.4}$$

where $g(\cdot)$ is a sigmoid transfer function, and $\mathbf{w} = (w_{ji}, v_{kj}, u_k, \cdots)$ denote the synaptic input weights of the neural unit. This scheme of nested non-linear functions is unusual in the classical theory of approximation (Poggio and Girosi 1990).

Projection Pursuit Regression uses approximation functions, which are a sum of univariant functions B_i of linear combinations of the input variables:

$$F(\mathbf{w}, \mathbf{x}) = \sum_{i=1}^{m} B_i(\mathbf{w}_i \cdot \mathbf{x}) \tag{3.5}$$

The interesting advantage is the straight forward solvability for affine transformations of the given task (scaling and rotation) (Friedman 1991).

Regression Trees: The domain of interest is recursively partitioned in hyper-rectangular subregions. The resulting subregions are stored e.g. as binary tree in the CART method ("Classification and Regression Tree" Breimann, Friedman, Olshen, and Stone 1984). Within each subregion, f is approximated - often by a constant - or by piecewise

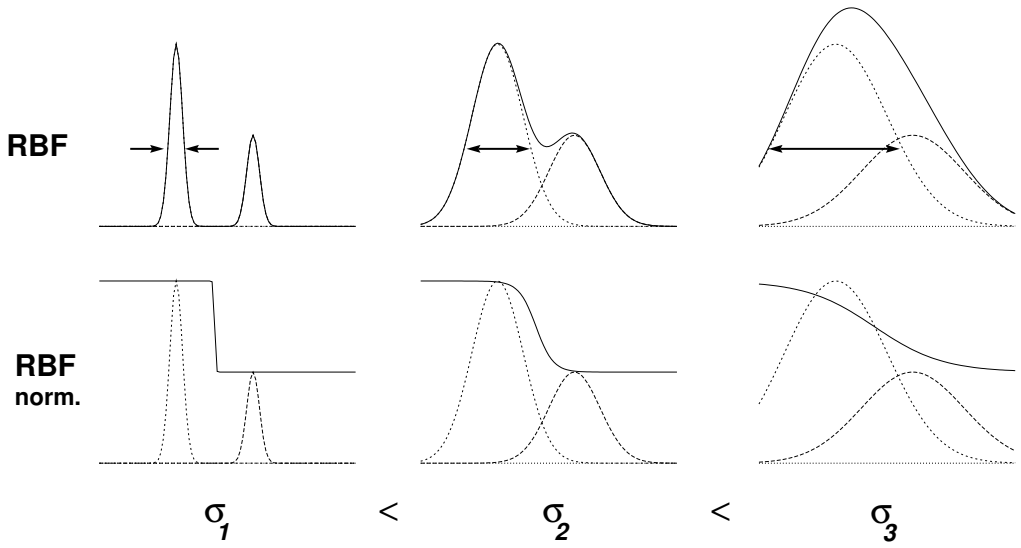


Figure 3.2: Two RBF units constitute the approximation model of a function. The *upper row* displays the plain RBF approach versus the results of the normalization step in the *lower row*. From *left* to *right* three basis radii $\sigma_{1...3}$ illustrate the smoothing impact of an increasing σ to distance ratio.

polynomial regression splines - but only for one or two components of x (MARS algorithm ("Multivariate Adaptive Regression Splines"; Friedman 1991).

Normalized Radial Basis Function (RBF) networks take the form of a weighted sum over reference points $\mathbf{w}_i$ located in the input space at $\mathbf{w}_i$:

$$F(\mathbf{w}, \mathbf{x}) = \frac{\sum_i \rho(|\mathbf{x} - \mathbf{u}_i|)\ w_i}{\sum_i \rho(|\mathbf{x} - \mathbf{u}_i|)} \tag{3.6}$$

The radial basis functions $\rho : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ usually decays to zero with growing argument and is often represented by the Gaussian bell function $\rho(r) = e^{-(r/\sqrt{2}\sigma)^2}$, characterized by the width σ (therefore the RBF is sometimes called a *kernel* function). The division by the unweighted sum takes care on normalization and flat extrapolation as illustrated in Fig. 3.2. The learning is often split in two phases: *(i)* the placement of the centers is learned by an unsupervised method, for example by *k-means* clustering, *learning vector quantization* (LVQ2), or competitive learning; *(ii)* the width σ is set, often

ad-hoc to the half mean distance of the base centers. The output values w_i are learned supervised. The RBF net can be combined with a *local linear mapping* instead of a constant w_i (Stokbro, Umberger, and Hertz 1990), as described below. RBF network algorithms which generalize the constant sized radii σ (sphere) to individually adaptable tensors (ellipses) are called "Generalized Radial Basis Function networks" (GRBF), or "Hyper-RBF" (see Powell 1987; Poggio and Girosi 1990).

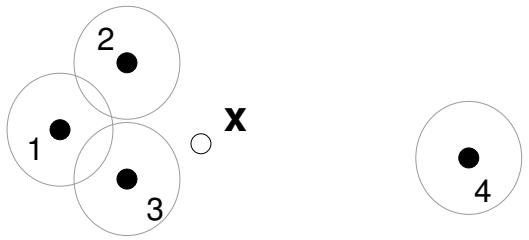


Figure 3.3: Distance versus topological distance. Four RBF unit center points $\mathbf{u}_i$ (denoted 1...4) around a test point $\mathbf{x}$ (the circles indicate the width σ). Accounting only for the distance $|\mathbf{x} - \mathbf{u}_i|$, the RBF output (Eq. 3.6) weights $\mathbf{u}_1$ stronger than $\mathbf{u}_4$. Considering the triangles spanned by the points 123 versus 234 reveals that $\mathbf{x}$ is far outside the triangle 123, but in the middle of the triangle 234. Therefore, $\mathbf{x}$ can be considered closer to point 4 than to point 1 — with respect to their topological relation.

Topological Models and Maps are schemes, which build dimension reducing mappings from a higher dimensions input space to a low-dimensional set. A very successful model is the so-called "feature map" or "Self-Organizing Map" (SOM) introduced by Kohonen (1984) and described below in Sec. 3.7. In the presented taxonomy the SOM has a special role: it has a localized knowledge representation where the location in the neural layer encodes *topological information* beyond Euclidean distances in the input space (see also Fig. 3.3). This means that input signals which have similar "features" will map to neighboring neurons in the network ("feature map"). This topological preserving effect works also in higher dimensions (famous examples are Kohonen's *Neural Typewriter* for spoken Finnish language, and the *semantic map*, where the similarity relationships of a set of 16 animals

could be extracted from a sequence of three-word sentences (Kohonen 1990; Ritter and Kohonen 1989). The topology preserving properties enables *cooperative learning* in order to increase speed and robustness of learning, studied e.g. in Walter, Martinetz, and Schulten (1991) and compared to the so-called *Neural-Gas Network* in Walter (1991) and Walter and Schulten (1993).

The **Neural-Gas Network** shows in contrast to the SOM not a fixed grid topology but a "gas-like", dynamic definition of the neighborhood function, which is determined by (dynamic) ranking of closeness in the input space (Martinetz and Schulten 1991). This results in advantages for applications with inhomogeneous or unknown topology (e.g. prediction of chaotic time series like the Mackey-Glass series in Walter (1991) and later also published in Martinetz et al. (1993)).

The choice of the type of approximation function introduces *bias*, and restricts the *variance* of the of the possible solutions. This is a fundamental relation called the *bias–variance* problem (Geman et al. 1992). As indicated before, this bias and the corresponding variance reduction can be good or bad, depending on the suitability of the choice. The next section discusses the problem over-using the variance of a chosen approximation ansatz, especially in the presence of noise.

3.5 Strategies to Avoid Over-Fitting

Over-fitting can occur, when the function f gets approximated in the domain D, using only a too limited number of training data points D_{train}. If the ratio of free parameter versus training points is too high, the approximation fits to the noise, as illustrated by Fig. 3.4. This results in a reduced generalization ability. Beside the proper selection of the appropriate network structure, several strategies can help to avoid the over-fitting effect:

Early stopping: During incremental learning the approximation error is systematically decreased, but at some point the expected error or lack-of-fit $LOF(F, D)$ starts to increase again. The idea of *early stopping* is to estimate the LOF on a separate test data set D_{test} and determine the optimal time to stop learning.

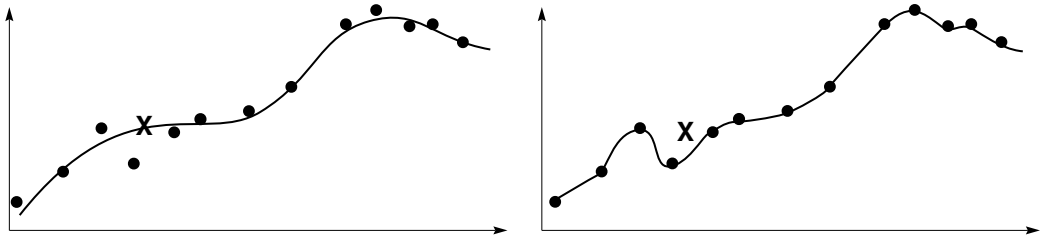


Figure 3.4: *(Left)* A meaningful fit to the given cross-marked noisy data. *(Right)* **Over-fitting** of the same data set: It fits well to the training set, but is performing badly on the indicated (cross-marked) position.

More training data: Over-fitting can be avoided when sufficient training points are available, e.g. by learning on-line. Duplicating the available training data set and adding a small amount of noise can help to some extent.

Smoothing and Regularization: Poggio and Girosi (1990) pointed out that learning from a limited set of data is an ill-posed problem and needs further assumptions to achieve meaningful generalization capabilities. The most usual presumption is *smoothness*, which can be formalized by a stabilizer term in the cost function Eq. 3.1 (regularization theory). The roughness penalty approximations can be written as

$$F(\mathbf{w}, \mathbf{x}) = \text{argmin}_F \left(LOF\left(F, D\right) + \lambda R\left(F\right) \right), \tag{3.7}$$

where $R(F)$ is a functional that describes the roughness of the function $F(\mathbf{w}, \mathbf{x})$. The parameter λ controls the tradeoff between the fidelity to the data and the smoothness of F. A common choice for R is the integrated squared Laplacian of F

$$R(F) = \sum_{i=1}^{n} \sum_{j=1}^{n} \int_D \left| \frac{\partial^2 F}{\partial x_i \partial x_j} \right|^2 d\mathbf{x} \tag{3.8}$$

which is equivalent to the *thin-plate spline* (for $n \leq 3$; coined by the energy of a bended thin plate of finite extent). The main difficulty is the introduction of a very influential parameter λ and the computation burden to carry out the integral.

For the topology preserving maps the smoothing is introduced by a parameter, which determines the range of learning coupling be-

tween neighboring neurons in the map. This can be interpreted as a regularization for the SOM and the "Neural-Gas" network.

3.6 Selecting the Right Network Size

Beside the accuracy criterion (LOF, Eq. 3.1) the simplicity of the network is desirable, similar to the idea of *Occam's Razor*. The formal way is to augment the cost function by a *complexity cost term*, which is often written as a function of the number of non-constant model parameters (additive or multiplicative penalty, e.g. the Generalized Cross-Validation criterion GCV; Craven and Wahba 1979).

There are several techniques to select the right network size and structure:

Trial-and-Error is probably the most prominent method in practice. A particular network structure is constructed and evaluated, which includes training and testing. The achieved *lack-of-fit* (LOF) is estimated and minimized.

Genetic Algorithms can automize this optimization method, in case of a suitable encoding of the construction parameter, the *genome* can be defined. Initially, a set of *individuals* (network genomes), the *population* is constructed by hand. During each *epoch*, the individuals of this *generation* are evaluated (training and testing). Their *fitnesses* (negative cost function) determine the probability of various ways of replication, including mutations (stochastic genome modifications) and cross-over (sexual replication with stochastic genome exchange). The applicability and success of this method depends strongly on the complexity of the problem, the effective representation, and the computation time required to simulate *evolution*. The computation time is governed by the product of the (non-parallelized) population size, the fitness evaluation time, and the number of simulated generations. For an introduction see Goldberg (1989) and, e.g. Miller, Todd, and Hegde (1989) for optimizing the coding structure and for weights determination Montana and Davis (1989).

Pruning and Weight Decay: By including a suitable non-linear complexity penalty term to the iterative learning cost function, a fraction of

the available parameters are forced to decay to small values (*weight decay*). These redundant terms are afterwards removed. The disadvantage of *pruning* (Hinton 1986; Hanson and Pratt 1989) or *optimal brain damage* (Cun, Denker, and Solla 1990) methods is that both start with rather large and therefore slower converging networks.

Growing Network Structures (*additive model*) follow the opposite direction. Usually, the learning algorithm monitors the network performance and decides when and how to *insert* further network elements (in form of data memory, neurons, or entire sub-nets) into the existing structure. This can be combined with outliers removing and pruning techniques, which is particularly useful when the growing step is generous (one-shot learning and forgetting the unimportant things). Various unsupervised algorithms have been proposed: additive models building local regression models (Breimann, Friedman, Olshen, and Stone 1984; Hastie and Tibshirani 1991), dynamic memory based models (Atkeson 1992; Schaal and Atkeson 1994), and RBF net (Platt 1991); the *tiling algorithm* (for binary outputs; Mézard and Nadal 1989) has similarities to the recursive partitioning procedure (MARS) but allows also non-orthogonal hyper-planes. The (binary output) *upstart algorithm* (Frean 1990) shares similarities with the continuous valued *cascade correlation* algorithm (Fahlman and Lebiere 1990; Littmann 1995). Adaptive topological models are studied in (Jockusch 1990), (Fritzke 1991) and in combination with the Neural-Gas in (Fritzke 1995).

3.7 Kohonen's Self-Organizing Map

Teuvo Kohonen formulated the (*Self-Organizing Map*) (SOM) algorithm as a mathematical model of the self-organization of certain structures in the brain, the *topographic maps* (e.g. Kohonen 1984).

In the cortex, neurons are often organized in two-dimensional sheets with connections to other areas of the cortex or sensor or motor neurons somewhere in the body. For example, the somatosensory cortex shows a *topographic map* of the sensory skin of the body. Topographic map means that neighboring areas on the skin find their neural connection and representation to neighboring neurons in the cortex. Another example is the

retinotopic map in the primary visual cortex (e.g. Obermayer et al. 1990).

Fig. 3.5 shows the basic operation of the Kohonen feature map. The map is built by a m (usually two) dimensional lattice $\mathbf{A}$ of formal neurons. Each neuron is labeled by an index $\mathbf{a} \in \mathbf{A}$, and has reference vectors $\mathbf{w_a}$ attached, projecting into the input space X (for more details, see Kohonen 1984; Kohonen 1990; Ritter et al. 1992).

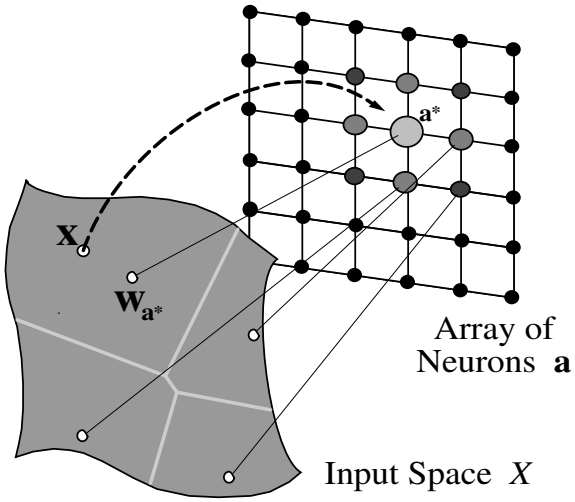


Figure 3.5: The "Self-Organizing Map" ("SOM") is formed by an array of processing units, called formal *neurons*. Here the usual case, a two-dimensional array is illustrated at the right side. Each neuron has a reference vector $\mathbf{w_a}$ attached, which is a point in the embedding input space X. A presented input $\mathbf{x}$ will select that neuron with $\mathbf{w_a}$ closest to it. This competitive mechanism tessellates the input space in discrete patches - the so-called *Voronoi cells*.

The response of a SOM to an input vector $\mathbf{x}$ is determined by the reference vector $\mathbf{w_{a^*}}$ of the discrete **"best-match" node** $\mathbf{a}^*$. The "winner" neuron $\mathbf{a}^*$ is defined as the node which has its reference vector $\mathbf{w_a}$ closest to the given input

$$\mathbf{a}^* = \underset{\forall \mathbf{a} \in \mathbf{A}}{\operatorname{argmin}} \; \|\mathbf{w_{a'}} - \mathbf{x}\| \,. \tag{3.9}$$

This competition among neurons can be biologically interpreted as a result of a lateral inhibition in the neural layer. The distribution of the reference vectors, or "weights" $\mathbf{w_a}$, is iteratively developed by a sequence of training vectors $\mathbf{x}$. After finding the best-match neuron $\mathbf{a}^*$ *all* reference vectors

are updated $\mathbf{w}_{\mathbf{a}}^{(new)} := \mathbf{w}_{\mathbf{a}}^{(old)} + \Delta\mathbf{w}_{\mathbf{a}}$ by the following adaption rule:

$$\Delta\mathbf{w}_{\mathbf{a}} = \epsilon\, h(\mathbf{a}, \mathbf{a}^*)\, (\mathbf{x} - \mathbf{w}_{\mathbf{a}}) \tag{3.10}$$

Here $h(\mathbf{a}, \mathbf{a}^*)$ is a bell shaped function (Gaussian) centered at the "winner" $\mathbf{a}^*$ and decaying with increasing distance $|\mathbf{a} - \mathbf{a}^*|$ in the neuron layer. Thus, each node or "neuron" in the neighborhood of the "winner" $\mathbf{a}^*$ participates in the current learning step (as indicated by the gray shading in Fig. 3.5.)

The networks starts with a given node grid $\mathbf{A}$ and a random initialization of the reference vectors. During the course of learning, the width of the neighborhood bell function $h(\cdot)$ and the learning step size parameter ϵ is continuously decreased in order to allow more and more specialization and fine tuning of the (then increasingly) individual neurons.

This particular cooperative nature of the adaptation algorithm has important advantages:

- it is able to generate *topological order* between the $\mathbf{w}_{\mathbf{a}}$;
- as a result, the convergence of the algorithm can be *sped up* by involving a whole group of neighboring neurons in each learning step;
- this is additionally valuable for the learning of output values with a higher degree of *robustness* (see Sect. 3.8 below).

By means of the Kohonen learning rule Eq. 3.10 an m–dimensional feature map will select a (possibly locally varying) subset of m independent features that capture as much of the variation of the stimulus distribution as possible. This is an important property that is also shared by the method of *principal component analysis* ("PCA", e.g. Jolliffe 1986). Here a linear sub-space is oriented along the axis of the maximum data variation, where in contrast the SOM can optimize its "best" features locally. Therefore, the feature map can be viewed as the non-linear extension of the PCA method.

The emerging tessellation of the input and the associated encoding in the node location code exhibits an interesting property related to the task of data compression. Assuming a noisy data transmission (or storage) of an encoded data set (e.g. image) the data reconstruction shows errors depending on the encoding and the distribution of noise included. Feature

map encoding (i.e. node Location in the neural array) are advantageous when the distribution of stochastic transmission errors is decreasing with distance to the original data. In case of an error the reconstruction will restore neighboring features, resulting in a more "faithful" compression.

Ritter showed the strict monotonic relationship between the stimulus density in the m-dimensional input space and the density of the matching weight vectors. Regions with high input stimulus density $P(\mathbf{x})$ will be represented by more specialized neurons than regions with lower stimulus density. For certain conditions the density of weight vectors could be derived to be proportional to $P(\mathbf{x})^{\kappa}$, with the exponent $\kappa = m/(m+2)$ (Ritter 1991).

3.8 Improving the Output of the SOM Schema

As discussed before, many learning applications desire continuous valued outputs. How can the SOM network learn smooth input–output mappings?

Similar to the *binning* in the hyper-rectangular recursive partitioning algorithm (CART), the original output learning strategy was the supervised teaching of an attached constant $y_{\mathbf{a}}$ (or vector $\mathbf{y_a}$) for every winning neuron $\mathbf{a}^*$

$$F(\mathbf{x}) = y_{\mathbf{a}^*}. \tag{3.11}$$

The next important step to increase the output precision was the introduction of a locally valid mapping around the reference vector. Cleveland (1979) introduced the idea of locally weighted linear regression for uni-variate approximation and later for multivariate regression (Cleveland and Devlin 1988). Independently, Ritter and Schulten (1986) developed the similar idea in the context of neural networks, which was later coined the **Local Linear Map** ("LLM") approach.

Within each subregion, the *Voronoi cell* (depicted in Fig. 3.5), the output is defined by a tangent hyper-plane described by the additional vector (or matrix) $\mathbf{B}$

$$F(\mathbf{x}) = y_{\mathbf{a}^*} + \mathbf{B}_{\mathbf{a}^*}(\mathbf{x} - \mathbf{w}_{\mathbf{a}^*}). \tag{3.12}$$

By this means, a univariate function is approximated by a set of tangents. In general, the output $F(\mathbf{x})$ is *discontinuous*, since the hyper-planes do not match at the Voronoi cell borders.

The next step is to smooth the LLM-outputs of several neurons, instead of considering one single neuron. This can be achieved by replacing the *"winner-takes-all"* rule (Eq. 3.9) with a *"winner-takes-most"* or *"softmax"* mechanism. For example, by employing Eq. 3.6 in the index space of lattice coordinates $\mathbf{A}$. Here the distance to the best-match $\mathbf{a}^*$ in the neuron index space determines the contribution of each neuron. The relative width σ controls how strong the distribution is smeared out, similarly to the neighborhood function $h(\cdot)$, but using a separate bell size.

This form of local linear map proved to be very successful in many applications, e.g. like the kinematic mapping for an industrial robot (Ritter, Martinetz, and Schulten 1989; Walter and Schulten 1993). In time-series prediction it was introduced in conjunction with the SOM (Walter, Ritter, and Schulten 1990) and later with the Neural-Gas network (Walter 1991; Martinetz et al. 1993). Wan (1993) won the Santa-Fee time-series contest (series X part) with a network built of *finite impulse response* ("FIR") elements, which have strong similarities to LLMs.

Considering the local mapping as an "expert" for a particular task subdomain, the LLM-extended SOM can be regarded as the precursor to the architectural idea of the *"mixture-of-experts"* networks (Jordan and Jacobs 1994). In this idea, the competitive SOM network performs the *gating* of the parallel operating, local experts. We will return to the *mixture-of-experts* architecture in Chap. 9.

Chapter 4

The PSOM Algorithm

Despite the improvement by the LLMs, the discrete nature of the standard SOM can be a limitation when the construction of *smooth, higher-dimensional* map manifolds is desired. Here a "blending" concept is required, which is generally applicable — also to higher dimensions.

Since the number of nodes grows exponentially with the number of map dimensions, manageably sized lattices with, say, more than three dimensions admit only very few nodes along each axis direction. Any discrete map can therefore not be sufficiently smooth for many purposes where continuity is very important, as e.g. in control tasks and in robotics.

In this chapter we discuss the *Parameterized Self-Organizing Map* ("PSOM") algorithm. It was originally introduced as the generalization of the SOM algorithm (Ritter 1993). The PSOM parameterizes a set of basis functions and constructs a smooth higher-dimensional map manifold. By this means a very small number of training points can be sufficient for learning very rapidly and achieving good generalization capabilities.

4.1 The Continuous Map

Starting from the SOM algorithm, described in the previous section, the PSOM is also based on a lattice of formal neurons, in the followig also called "nodes". Similarly to the SOM, each node carries a reference vector $\mathbf{w_a}$, projecting into the d-dimensional embedding space $X \subseteq \mathbb{R}^d$.

- The first step is to generalize the index space $\mathbf{A}$ in the Kohonen map to a continuous *auxiliary mapping* or *parameter manifold* $S \in \mathbb{R}^m$ in the

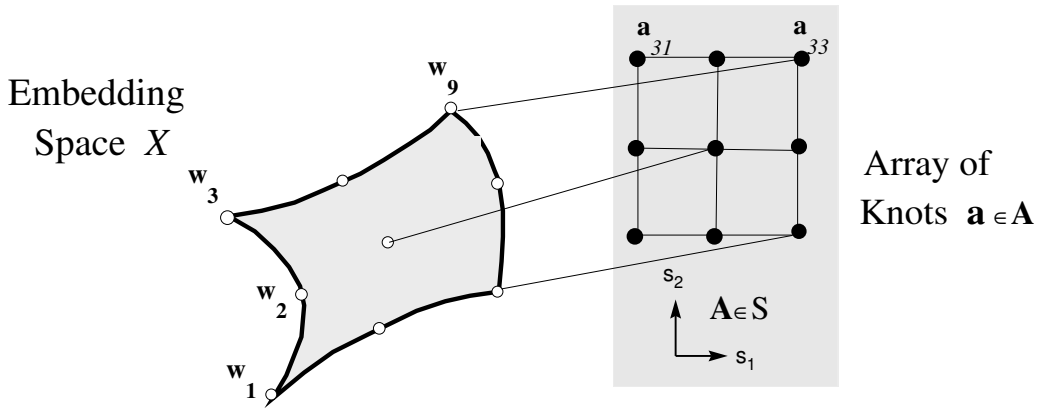


Figure 4.1: The PSOM's starting position is very much the same as for the SOM depicted in Fig. 3.5. The gray shading indicates that the index space $\mathbf{A}$, which is discrete in the SOM, has been generalized to the continuous space S in the PSOM. The space S is referred to as *parameter space* S.

PSOM. This is indicated by the grey shaded area on the right side of Fig. 4.1.

- The second important step is to define a *continuous mapping* $\mathbf{w}(\cdot) : \mathbf{s} \mapsto \mathbf{w}(\mathbf{s}) \in M \subset X$, where $\mathbf{s}$ varies continuously over $S \subseteq \mathbb{R}^m$.

Fig. 4.2 illustrates on the left the m=2 dimensional *"embedded manifold"* M in the d=3 dimensional embedding space X. M is spanned by the nine (dot marked) reference vectors $\mathbf{w}_1, \ldots, \mathbf{w}_9$, which are lying in a tilted plane in this didactic example. The cube is drawn for visual guidance only. The dashed grid is the image under the mapping $\mathbf{w}(\cdot)$ of the (right) rectangular grid in the parameter manifold S.

How can the smooth manifold $\mathbf{w}(\mathbf{s})$ be constructed? We require that the embedded manifold M passes through all supporting reference vectors $\mathbf{w_a}$ and write $\mathbf{w}(\cdot) : S \to M \subset X$:

$$\mathbf{w}(\mathbf{s}) = \sum_{\mathbf{a} \in \mathbf{A}} H(\mathbf{a}, \mathbf{s})\, \mathbf{w_a} \tag{4.1}$$

This means that, we need a **"basis function"** $H(\mathbf{a}, \mathbf{s})$ for each formal node, weighting the contribution of its reference vector (= initial "training point") $\mathbf{w_a}$ depending on the location $\mathbf{s}$ relative to the node position $\mathbf{a}$, and possibly, also *all* other nodes $\mathbf{A}$ (however, we drop in our notation the dependency $H(\mathbf{a}, \mathbf{s}) = H(\mathbf{a}, \mathbf{s};\, \mathbf{A})$ on the latter).

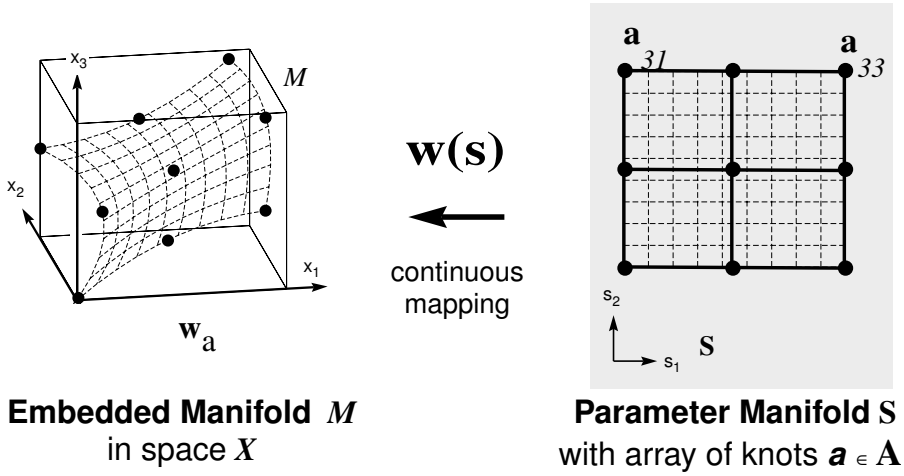


Figure 4.2: The mapping $\mathbf{w}(\cdot) : S \rightarrow M \subset X$ builds a continuous image of the *right* side S in the embedding space X at the *left* side.

Specifying for each training vector a node location $\mathbf{a} \in \mathbf{A}$ introduces a **topological order** between the training points $\mathbf{w_a}$: training vectors assigned to nodes $\mathbf{a}$ and $\mathbf{a}'$, that are neighbors in the lattice $\mathbf{A}$, are perceived to have this specific neighborhood relation. This has an important effect: it allows the PSOM to *draw extra curvature information* from the training set. Such information is not available within other techniques, such as the RBF approach (compare Fig. 3.3, and later examples, also in Chap. 8).

The topological organization of the given data points is crucial for a good generalization behavior. For a general data set the topological ordering of its points may be quite irregular and a set of suitable basis functions $H(\mathbf{a}, \mathbf{s})$ difficult to construct.

A suitable set of basis functions can be constructed in several ways but must meet two conditions:

Orthonormality Condition: (i) The hyper-surface M shall pass through all desired support points. At those points, only the local node contributes (with weight one):

$$H(\mathbf{a}_i, \mathbf{a}_j) = \delta_{ij} \; ; \quad \forall \; \mathbf{a}_i, \mathbf{a}_j \in \mathbf{A}. \tag{4.2}$$

Partition-of-Unity Condition: Consider the task of mapping a constant function $\mathbf{x} = \mathbf{w_a}$. Obviously, the sum in Eq. 4.1 should be constant

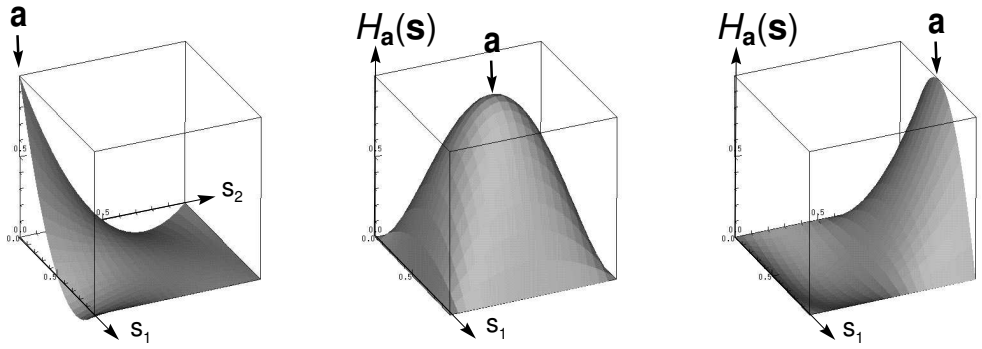


Figure 4.3: Three of the nine basis functions $H(\mathbf{a},\cdot)$ for a 3×3 PSOM with equidistant node spacing $\mathbf{A} = \{0,\frac{1}{2},1\} \otimes \{0,\frac{1}{2},1\}$ *(left:)* $H((0,0),\mathbf{s})$; *(middle:)* $H((\frac{1}{2},\frac{1}{2}),\mathbf{s})$; *(right:)* $H((\frac{1}{2},1),\mathbf{s})$. The remaining six basis functions are obtained by $90°$ rotations around $\mathbf{s} = (\frac{1}{2},\frac{1}{2})$.

as well, which means, the sum of all contribution weights should be one:

$$\sum_{\mathbf{a}\in\mathbf{A}} H(\mathbf{a},\mathbf{s}) = 1,\ \forall\mathbf{s}. \tag{4.3}$$

A simple construction of basis functions $H(\mathbf{a},\mathbf{s})$ becomes possible when the topology of the given points is sufficiently regular. A particularly convenient situation arises for the case of a multidimensional rectangular grid. In this case, the set of functions $H(\mathbf{a},\mathbf{s})$ can be constructed from products of one-dimensional Lagrange interpolation polynomials. Fig. 4.3 depicts three (of nine) basis functions $H(\mathbf{a},\mathbf{s})$ for the $m = 2$ dimensional example with a 3×3 rectangular node grid $\mathbf{A}$ shown in Fig. 4.5. Sec. 4.5 will give the construction details and reports about implementation aspects for fast and efficient computation of $H(\mathbf{a},\mathbf{s})$ etc.

4.2 The Continuous Associative Completion

When M has been specified, the PSOM is used in an analogous fashion like the SOM: given an input vector $\mathbf{x}$, (i) first find the best-match position $\mathbf{s}^*$ on the mapping manifold S by minimizing a distance function $dist(\cdot)$:

$$\mathbf{s}^* = \underset{\forall\mathbf{s}\in S}{\text{argmin}}\ dist\left(\mathbf{w}(\mathbf{s}),\mathbf{x}\right). \tag{4.4}$$

Then (ii) use the surface point $\mathbf{w}(\mathbf{s}^*)$ as the output of the PSOM in response to the input $\mathbf{x}$.

To build an input-output mapping, the standard SOM is often extended by attaching a second vector $\mathbf{w}^{out}$ to each formal neuron. Here, we generalize this and view the embedding space X as the Cartesian product of the input subspace X^{in} and the output subspace X^{out}

$$X = X^{in} \otimes X^{out} \quad \subset \mathbb{R}^d. \tag{4.5}$$

Then, $\mathbf{w}(\mathbf{s}^*)$ can be viewed as an *associative completion* of the input space component of $\mathbf{x}$ if the distance function $dist(\cdot)$ (in Eq. 4.4) is chosen as the Euclidean norm applied only to the input components of $\mathbf{x}$ (belonging to X^{in}). Thus, the function $dist(\cdot)$ actually selects the input subspace X^{in}, since for the determination of $\mathbf{s}^*$ (Eq. 4.4) and, as a consequence, of $\mathbf{w}(\mathbf{s}^*)$, only those components of $\mathbf{x}$ matter, that are regarded in the distance metric $dist(\cdot)$. The mathematical formulation is the definition of a diagonal projection matrix

$$\mathbf{P} = \text{diag}(p_1, p_2, \ldots, p_d) \tag{4.6}$$

with diagonal element $p_k > 0,\ \forall\, k \in I$, and all other elements zero. The set I is the subset of components of X ($\{1, 2, \ldots, d\}$) belonging to the desired X^{in}. Then, the distance function can be written as

$$dist(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^T \, \mathbf{P} \, (\mathbf{x} - \mathbf{x}') = \sum_{k \in I} p_k (x_k - x'_k)^2 = \sum_{k=1}^{d} p_k \, (x_k - x'_k)^2. \tag{4.7}$$

For example, consider a $d = 5$ dimensional embedding space X, where the components $I = \{1, 3, 4\}$ belong to the input space. Only those must be specified as inputs to the PSOM:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ * \\ x_3 \\ x_4 \\ * \end{pmatrix} \begin{matrix} \\ \leftarrow \\ \\ \\ \leftarrow \end{matrix} \begin{matrix} \\ \text{missing} \\ \text{components} \\ = \text{desired} \\ \text{output} \end{matrix} \tag{4.8}$$

The next step is the costly part in the PSOM operation: the iterative "best-match" search for the parameter space location $\mathbf{s}^*$, Eq. 4.4 (see next section.) In our example Eq. 4.8, the distance metric Eq. 4.7 is specified

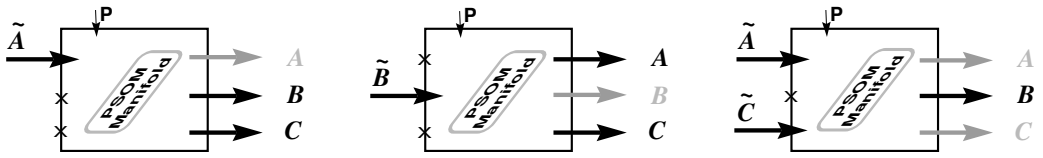


Figure 4.4: "Continuous associative memory" supports multiple mapping directions. The specified $\mathbf{P}$ matrices select different subspaces (here symbolized by $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$) of the embedding space as inputs. Values of variables in the selected input subspaces are considered as "clamped" (indicated by a tilde) and determine the values found by the iterative least square minimization (Eq. 4.7). for the "best-match" vector $\mathbf{w}(\mathbf{s}^*)$. This provides an associative memory for the flexible representation of continuous relations.

as the Euclidean norm applied to the components 1,3 and 4, which is equivalent to writing $\mathbf{P}$=diag(1,0,1,1,0).

The associative completion $\mathbf{x}^*$ is then the extension of the vector $\mathbf{x}$ found by the components in the embedding manifold M:

$$\mathbf{w}(\mathbf{s}^*) = \begin{pmatrix} w_1(\mathbf{s}^*) \\ w_2(\mathbf{s}^*) \\ w_3(\mathbf{s}^*) \\ w_4(\mathbf{s}^*) \\ w_5(\mathbf{s}^*) \end{pmatrix} \begin{matrix} \\ \longrightarrow \\ \\ \\ \longrightarrow \end{matrix} \begin{pmatrix} x_1 \\ w_2(\mathbf{s}^*) \\ x_3 \\ x_4 \\ w_5(\mathbf{s}^*) \end{pmatrix} =: \mathbf{x}^* \tag{4.9}$$

Fig. 4.4 illustrates the procedure graphically.

For the previous $d = 3$ PSOM example, Fig. 4.5 illustrates visually the associative completion $I = \{1, 3\}$ for a set of input vectors. Fig. 4.5 shows the result of the "best-match projection" $\mathbf{x} \mapsto \mathbf{s}^*(\mathbf{x})$ into the manifold M, when $\mathbf{x}$ varies over a regular 10×10 grid in the plane $x_2 = 0$. Fig. 4.5c displays a rendering of the associated "completions" $\mathbf{w}(\mathbf{s}^*(\mathbf{x}))$, which form a grid in X.

As an important feature, the distance function $dist(\cdot)$ can be changed on demand, which allows to freely (re-) partition the embedding space X in input subspace and output subspace. One can, for example, reverse the mapping direction or switch to other input coordinate systems, using the same PSOM.

Staying with the previous simple example, Figures 4.6 illustrate the alternative use of the previous PSOM in example Fig. 4.5. To complete this

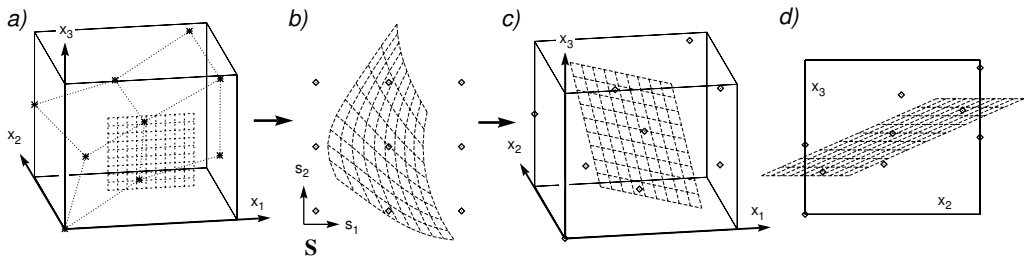


Figure 4.5: a–d: PSOM associative completion or recall procedure ($I = \{1, 3\}$, **P**=diag(1,0,1)) for a rectangular spaced set of 10×10 (x_1, x_3) tuples to (x_2, x_3), together with the original training set of Fig. 4.1,4.5. *(a)* the input space in the $x_2 = 0$ plane, *(b)* the resulting (Eq. 4.4) mapping coordinates $\mathbf{s}^* \in S$, *(c)* the completed data set in X, *(d)* the desired output space projection (looking down x_1).

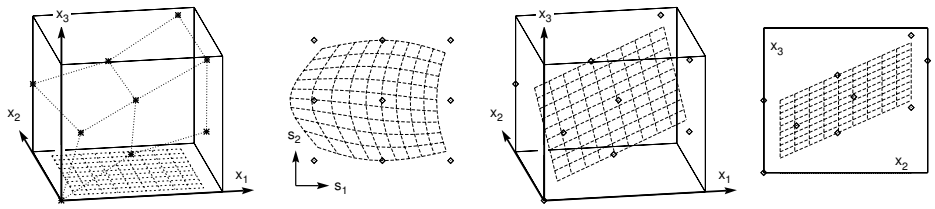


Figure 4.6: a–d: PSOM associative completion procedure, but in contrast to Fig. 4.5 here mapping from the input subspace $x_3 = 0$ with the input components $I = \{1, 2\}$ (**P**=diag(1,1,0)).

alternative set of input vectors an alternative input subspace $I = \{1, 2\}$ is specified.

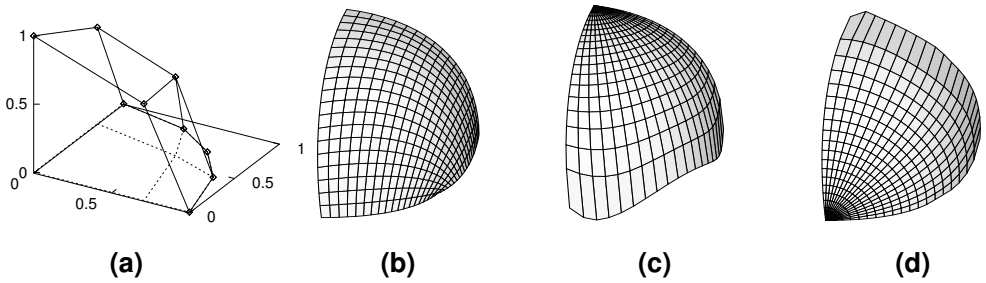


Figure 4.7: *(a:)* Reference vectors of a 3×3 SOM, shared as training vectors by a 3×3 PSOM, representing one octant of the unit sphere surface ($x_1, x_2, x_3 > 0$, see also the projection on the $x_3 = 0$ base plane). *(b:)* Surface plot of the mapping manifold M as image of a rectangular test grid in S. *(c:)* A mapping $x_3 = x_3(x_1, x_2)$ obtained from the PSOM with $\mathbf{P}$ =diag$(1, 1, 0)$, *(d:)* same PSOM, but used for mapping $x_1 = x_1(x_2, x_3)$ by choosing $\mathbf{P}$ =diag$(0, 1, 1)$.

As another simple example, consider a 2-dimensional data manifold in $\mathbb{R}^3$ that is given by the portion of the unit sphere in the octant $x_i \geq 0$ ($i = 1, 2, 3$). Fig. 4.7, *left*, shows a SOM, providing a discrete approximation to this manifold with a 3×3-mesh. While the number of nodes could be easily increased to obtain a better approximation for the two-dimensional manifold of this example, this remedy becomes impractical for higher dimensional manifolds. There the coarse approximation that results from having only three nodes along each manifold dimension is typical. However, we can use the nine reference vectors together with the neighborhood information from the 3×3 SOM to construct a PSOM that provides a much better, fully continuous representation of the underlying manifold.

Fig. 4.7 demonstrates the 3×3 PSOM working in two different mapping "directions". This flexibility in associative completion of alternative input spaces X^{in} is useful in many contexts. For instance, in robotics a positioning constraint can be formulated in joint, Cartesian or, more general, in mixed variables (e.g. position and some wrist joint angles), and one may need to know the respective complementary coordinate representation, requiring the direct and the inverse kinematics in the first two cases, and a mixed transform in the third case. If one knows the required cases

beforehand, one can construct a separate mapping for each. However, it is clearly more desirable to work with a single network that can complete different sets of missing variable values from different sets of known values.

4.3 The Best-Match Search

Returning to Eq. 4.4, we see that the discrete best-match search in the standard SOM is now replaced by solving the continuous minimization problem for the cost function

$$E(\mathbf{s}) = \tfrac{1}{2} dist\left(\mathbf{x}, \mathbf{w}(\mathbf{s})\right) = \tfrac{1}{2} \sum_{k=1}^{d} p_k \left[x_k - w_k(\mathbf{s})\right]^2. \tag{4.10}$$

A very straightforward solution is to start with a good guess $\mathbf{s}_{t=0} = \mathbf{a}^*$ taken as the node location of the best-matching reference vector $\mathbf{w}_{\mathbf{a}^*}$, analog to the SOM best-match step in Eq. 3.9. Then an iterative gradient descent is performed

$$\mathbf{s}_{t+1} = \mathbf{s}_t + \gamma \sum_{k=1}^{d} p_k \frac{\partial w_k(\mathbf{s}_t)}{\partial \mathbf{s}} \left[\mathbf{x}_k - w_k(\mathbf{s}_t)\right]. \tag{4.11}$$

with $\gamma > 0$ as a step size parameter.

However the simple first order gradient descent method given is very sensitive to the proper choice of the step size parameter γ. Too small values lead to unnecessarily many iteration steps, too large values can lead to divergence.

We tested various algorithms for automatically adjusting $\gamma = \gamma(t)$ on the basis of observing the sequence in cost reduction and gradient direction. Some can indeed increase robustness and the rate of convergence but bear the disadvantage of introducing other critical parameters to the algorithm. Other techniques were considered, which are often used for improving the back-propagation learning algorithm: Eq. 4.11 is augmented by a "momentum term"

$$\Delta \mathbf{s}'_{t+1} = \Delta \mathbf{s}_{t+1} + \beta \, \mathbf{s}_t. \tag{4.12}$$

with the intention to suppress oscillations. This technique is very helpful if the parameter β is well-chosen. One method which determines β is known

as *conjugate gradient decent*. It involves a line minimization procedure for each gradient direction and optimizes successive search directions. The line minimization usually probes $E(\mathbf{s} + \kappa\Delta\mathbf{s})$ at three points to find the one-dimensional sub-space minimum on the parabola. The idea behind the conjugate gradient method is to accumulate information on successive steps in order to lead to the exact minimum for quadratic cost functions (*Fletcher-Reeves* and *Polak-Ribiere*, as well as other advanced variable metric minimization schemes, e.g. the *Fletcher-Powell* algorithm; Stoer and Bulirsch 1980; Press et al. 1988; Hertz et al. 1991).

The much more direct way is to use a second order method. In contrast to the feed-forward networks, the PSOM networks does allow easy computation of the second derivatives of the basis function $H(\mathbf{a}, \mathbf{s})$ (shown later in this chapter).

The simplest second-order method is the *Newton-Raphson* method. It can be formulated as the Taylor expansion of the gradient $\nabla_{\mathbf{S}} E$ at the estimated best-match location $\mathbf{s}$. Unfortunately, the second derivative, the Hessian matrix $\nabla_{\mathbf{S}}\nabla_{\mathbf{S}} E$ may not have full rank and as a consequence, it might be non-invertible. A very robust recipe was developed by Levenberg and Marquardt. They suggested to add a diagonal matrix term $\lambda\mathbb{1}$ to the Hessian matrix

$$(\nabla_{\mathbf{S}}\nabla_{\mathbf{S}} E(\mathbf{s}) + \lambda\mathbb{1})\,\delta\mathbf{s} = \nabla_{\mathbf{S}} E(\mathbf{s} + \delta\mathbf{s}). \tag{4.13}$$

If the so-called Levenberg-Marquardt parameter λ is small, the algorithm is dominantly a Newton-Raphson method. On the other hand, if λ is large, the inverted matrix is called diagonally dominant, and the resulting behavior approaches that of the steepest gradient method (Marquardt 1963; Press et al. 1988).

Starting with $\mathbf{s}_{t=0} = \mathbf{a}^*$ and (e.g.) $\lambda = 10^{-2}$, λ is adapted dynamically: if the last iteration step resulted in a cost reduction $E(\mathbf{s}_t - \delta\mathbf{s})$ the step is accepted, $\mathbf{s}_{t+1} = \mathbf{s}_t - \delta\mathbf{s}$, and λ is decreased by a significant factor of, e.g. 10. Otherwise, if the step leads to an increase of cost, then the step will be repeated with $\delta\mathbf{s}$ based on an increased λ. The iteration is stopped when the step size drops below a desired threshold.

The Levenberg-Marquardt works very satisfactorily and finds the minimum of $E(\mathbf{s})$ efficiently within a couple of iterations. Our experience shows a more than one order of magnitude higher speed than the simple steepest gradient algorithm. The general problem of finding a global

or one out of several local minima as best-match is discussed in section 6.1.

4.4 Learning Phases

As pointed out in the introduction, in the biological world we find many types of learning. The PSOM approach offers several forms by itself.

One PSOM's core idea is the construction of the continuous mapping manifold using a topologically ordered set of reference vectors $\mathbf{w_s}$. Therefore the first question of learning can be formulated: how to obtain the topological order of the reference vectors $\mathbf{w_a}$? The second question is, how to adapt and improve the mapping accuracy in an on-line fashion, allowing to cope with drifting and changing tasks and target mappings?

The PSOM algorithm offers two principal ways for this initial learning phase:

PSOM Unsupervised Self-Organization Mode: One way is to employ the Kohonen adaptation rule Eq. 3.10 described in Sect. 3.7. The advantage is that no structural information is required. On the other hand this process is iterative and can be slow. It might require much more data to recover the correct structural order (see also stimulus-sampling theory in the introduction). Numerous examples on the self-organizing process are given in the literature, e.g. (Kohonen 1990; Ritter, Martinetz, and Schulten 1992; Kohonen 1995))

PSOM Supervised Construction (PAM: Parameterized Associative Map) In some cases, the information on the topological order of the training data set can be *otherwise inferred* — for example *generated* — by actively sampling the training data. In the robotics domain frequently this can be done by structuring the data gathering process — often without any extra effort.

Then the learning of the PSOM is *sped up* tremendously: the iterative self-organization step can be replaced by an *immediate construction*. The prototypically sampled data vectors $\mathbf{w_a}$ are simply *assigned* to the node locations $\mathbf{a}$ in the set $\mathbf{A}$. In the following chapter, several examples shall illustrate this procedure.

Irrespective of the way in which the initial structure of the PSOM mapping manifold was established, the PSOM can be continuously **fine-tuned**

following an **error-minimization** process. This is particularly useful to improve the a PSOM that was constructed with a noisy sampled training set or to adapt it to a changing or drifting system.

In the following we propose a supervised learning rule for the reference vectors $\mathbf{w_a}$, minimizing the obtained output error. Required is the target, the embedded input–output vector, here also denoted $\mathbf{x}$. The best-match $\mathbf{w}(\mathbf{s}^*) \in M$ for the current input $\mathbf{x}$ is found according to the current input sub-space specification (set of components I; distance metric $\mathbf{P}$). Each reference vector is then adjusted, weighted by its contribution factor $H(\cdot)$, minimizing the deviation to the desired joint input-output vector $\mathbf{x}$

$$\Delta\mathbf{w_a} = \epsilon\, H(\mathbf{a}, \mathbf{s}^*).\ (\mathbf{x} - \mathbf{w}(\mathbf{s}^*)) \tag{4.14}$$

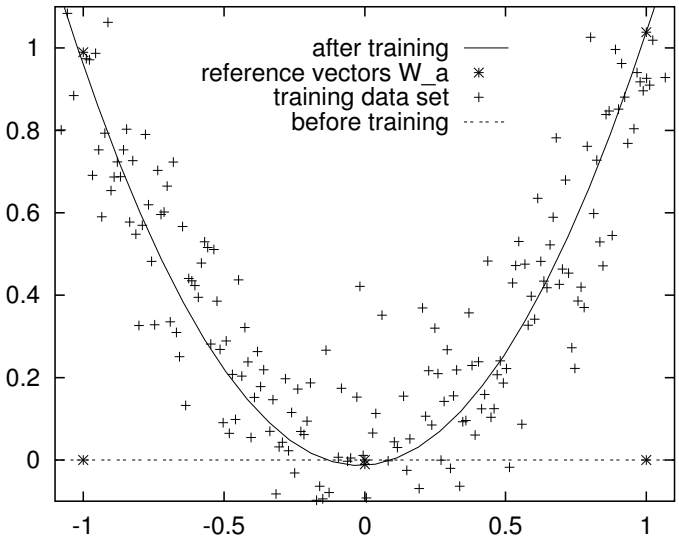


Figure 4.8: The mapping result of a three node PSOM before and after learning by the means of Eq. 3.10 with the "+" marked learning examples (generated by $x_2 = x_1^2 + \eta(0, 0.15)$; normal distributed noise η with mean 0 and standard deviation 0.15). The positions of the reference vectors $\mathbf{w_a}$ are indicated by the asterisks.

The same adaptation mechanism can be employed for learning the output components of the reference vectors from scratch, i.e. even in the absence of initial corresponding output data. As a prerequisite, the input sub-space X^{in} of the mapping manifold must be spanned in a topological order to facilitate the best-match dynamic Eq. 4.4 to work properly.

Assuming that a proper subset of input components I has been specified (usually when the number of input components is equal to the mapping dimension, $|I| = m$), an input vector $\mathbf{Px}$ can be completed with a zero distance ($E(\mathbf{s}^*)$=0) best-match location $\mathbf{Pw}(\mathbf{s}^*) = \mathbf{Px}$ in the mapping manifold M. Then, only the output components are effectively adapted by Eq. 4.14 (equivalent to the selection matrix $\mathbb{1} - \mathbf{P}$). If the measurement of the training vector components is differently reliable or noisy, the learning parameter ϵ might be replaced by a suitable diagonal matrix controlling the adaptation strength per component (we return to this issue in Sect. 8.2.)

Fig. 4.8 illustrates this procedure for an $m = 1$ dimensional three-node PSOM. We consider the case of a $d = 2$ dimensional embedding space (x_1, x_2) and $I = \{1\}$, i.e., a mapping $x_1 \rightarrow x_2 = x_1^2$. The embedded manifold M is drawn before and after 300 adaptation steps (using Eq. 4.14 with a learning steps parameter ϵ linearly decreased from 0.8 to 0.1) together with the reference vectors and the noisy training data set.

Note, that Eq. 4.14 is taking the entire PSOM output $\mathbf{w}(\mathbf{s}^*)$ into account, in contrast to $\mathbf{w_a}$ in the "locally acting" Kohonen step of Eq. 3.10. But the alternatively proposed learning rule (Ritter 1993)

$$\Delta \mathbf{w_a} = \epsilon \; H(\mathbf{a}, \mathbf{s}^*) \; (\mathbf{x} - \mathbf{w_a}) \tag{4.15}$$

turns out to be unstable. A correctly trained PSOM is even corrupted by a sequence of further training examples.

We demonstrate this fact by an example. For the same learning task, Fig. 4.9 depicts the initially correct parabola shaped mapping manifold M and the asterisk marked locations of the three reference vectors $\mathbf{w_a}$ in the front curve. The other curves, shifted to the back, show M after several epochs of 25 adaptation steps, each using Eq. 4.15 (uniformly random sampled training vectors, $\epsilon = 0.15$). The adaptation leads to a displacement of the reference vectors in such a way, that the symmetric parabola becomes distorted. This shows that Eq. 4.15 is not able to maintain the correct mapping. The reference vectors are shaken around too-much – in a non-localized manner, and in both the output and input space. Note, that the mapping manifold M in Fig. 4.9 does not display one simple parabola but in fact the parameterized curve $(w_1(s), w_2(s))$ (the joint ordinate projection of two parabolas $w_1(s)$, $w_2(s)$ with the invisible abscissa s in the parameter manifold S; Initially $w_1(s)$ is constructed here as a linear function (degenerated parabola) in s, similar as Fig. 4.8).

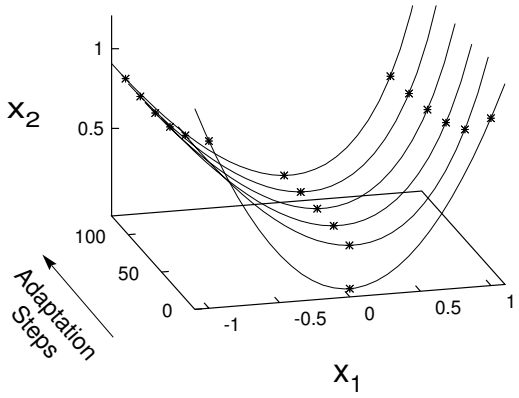


Figure 4.9: In contrast to the adaptation rule Eq. 4.14, the modified rule Eq. 4.15 is here instable, see text.

4.5 Implementation Aspects and the Choice of Basis Functions

As discussed previously in this chapter, the PSOM utilizes a set of basis functions $H(\mathbf{a}, \mathbf{s})$, which must satisfy the orthonormality and the division-of-unity condition. Here we explain one particular class of basis functions. It is devoted to the reader also interested in the numerical and implementation aspects. (This section is not a prerequisite for the understanding of the following chapters.)

A favorable choice for $H(\mathbf{a}, \mathbf{s})$ is the multidimensional extension of the well known Lagrange polynomial. In one dimension ($m = d = 1$) a is from a set $A = \{a_i \mid i = 1, 2, \ldots n\}$ of discrete values and Eq. 4.1 can be written as the Lagrange polynomial interpolating through n support points (a_i, w_i):

$$w(s) = l_1(s, A)w_1 + l_2(s, A)w_2 + \cdots + l_n(s, A)w_n = \sum_{k=1}^{n} l_k(s, A)\ w_k \quad (4.16)$$

where the Lagrange factors $l_i(s, A)$ are defined as

$$l_i(s, A) = \prod_{j=1,\ j\neq i}^{n} \frac{s - a_j}{a_i - a_j}. \quad (4.17)$$

If $m > 1$, Eq. 4.16 and 4.17 can be generalized in a straightforward fashion, provided the set of node points form a m-dimensional hyper-

lattice in S, i.e. the Cartesian product of m one-dimensional node point sets $\mathbf{A} = A_1 \times A_2 \times \cdots \times A_m$, with $A_\nu = \{{}^\nu a_1, {}^\nu a_2, \ldots, {}^\nu a_{n_\nu}\}$. Here, ${}^\nu a_i$ denotes the i-th value that can be taken by the ν-th component ($\nu \in \{1, \ldots, m\}$) of $\mathbf{a}$ along the ν-th dimension of $\mathbf{s} = ({}^1s, {}^2s, \ldots, {}^ms)^T \in S$. Fig. 4.10 illustrates an example with $n_1 = 3$, $n_2 = 4$, and $n_3 = 2$.

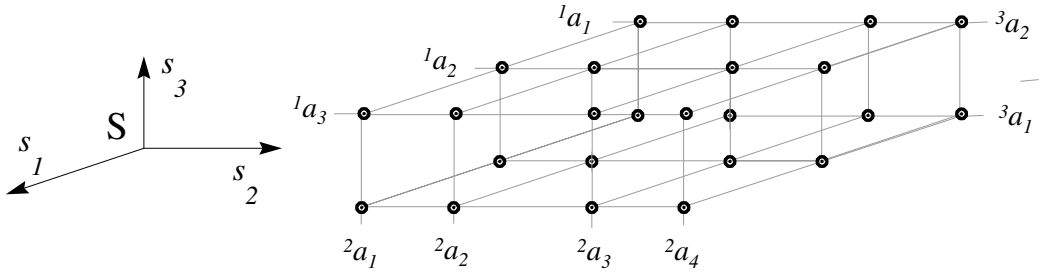


Figure 4.10: Example of a $m = 3$ dimensional mapping manifold S with a $3\times 4\times \times 2$ node set $\mathbf{A} \subset S$ in orthonormal projection. Note the rectangular structure and the non-equidistant spacing of the ${}^2a_i$.

With this notation, we can write $H(\mathbf{a}, \mathbf{s}) = H(\mathbf{a}, \mathbf{s}; \mathbf{A})$ as

$$H(\mathbf{a}, \mathbf{s}) = \prod_{\nu=1}^{m} l_{i_\nu}({}^\nu s, A_\nu) \tag{4.18}$$

using the one-dimensional Lagrange factors and the notation

$$\mathbf{a} = \mathbf{a_i} = ({}^1a_{i_1}, {}^2a_{i_2}, \ldots, {}^ma_{i_m})^T \in \mathbf{A}.$$

In Fig. 4.3 (p. 46) some basis functions of a $m = 2$ dimensional PSOM with equidistantly chosen node spacing, are rendered. Note, that the PSOM algorithm is invariant to any offset for each S axis, and together with the iterative best-match finding procedure, it becomes also invariant to rescaling.

Comparative results using (for $n > 3$) a node spacing derived from the Chebyshev polynomial are reported in Sec. 6.4.

The first derivative of (4.1) turns out to be surprisingly simple, if we write the product rule in the form

$$\frac{\partial}{\partial x} f_1(x) f_2(x) \cdots f_n(x) = f_1(x) f_2(x) \cdots f_n(x) \sum_{k=1}^{n} \frac{\frac{\partial f_k}{\partial x}(x)}{f_k(x)}, \qquad \forall f_k(x) \neq 0. \tag{4.19}$$

With the intermediate result

$$\frac{\partial}{\partial\, {}^{\mu}s} l_{i_\mathcal{V}}({}^{\nu}s, A_\nu) = \delta_{\mu\nu} l_{i_\mathcal{V}}({}^{\nu}s, A_\nu) \sum_{j=1,\ j\neq i_\mathcal{V}}^{n_\mathcal{V}} \frac{1}{{}^{\nu}s - {}^{\nu}a_j} \tag{4.20}$$

and

$$\frac{\partial}{\partial({}^{\mu}s)} H(\mathbf{a},\mathbf{s}) = H(\mathbf{a},\mathbf{s}) \sum_{\mathcal{V}=1}^{m} \frac{\frac{\partial}{\partial({}^{\mu}s)} l_{i_\mathcal{V}}({}^{\nu}s, A_\nu)}{l_{i_\mathcal{V}}({}^{\nu}s, A_\nu)} = H(\mathbf{a},\mathbf{s}) \sum_{j=1,\ j\neq i_\mu}^{n_\mu} \frac{1}{{}^{\mu}s - {}^{\mu}a_j} \tag{4.21}$$

the row vector of the required Jacobian matrix is

$$\frac{\partial \mathbf{w}(\mathbf{s})}{\partial({}^{\mu}s)} = \sum_{\mathbf{a}} \mathbf{w_a}\, H(\mathbf{a},\mathbf{s}) \sum_{j=1,\ j\neq i_\mu}^{n_\mu} \frac{1}{{}^{\mu}s - {}^{\mu}a_j}. \tag{4.22}$$

This is correct, if $f_k(l_i) \neq 0$. Or, in other words, for all s staying away from the dashed grid structure in Fig. 4.10; there, one or more of the product functions f_k (l_i) become zero. Although the vanishing denominator is canceled by a corresponding zero in the pre-factor (the derivative of a polynomial is well-behaved everywhere), the numerical implementation of the algorithm requires special attention in the vicinity of a diminishing denominator in Eq. 4.20.

One approach is, to treat in each S -axis the smallest term $\|{}^{\nu}s - {}^{\nu}a_j\|$ direction ν *always* separately. This is shown below.

For an implementation of the algorithm, an important point is the efficient evaluation of the Lagrange factors and their derivatives. Below we give some hints how to improve their computation requiring $O(n)$ operations, instead of $O(n^2)$ operations for the "naive" approach. For the sake of readability we omit from here on the (upper left) S component index ν ($\in \{1, 2, \ldots, m\}$) and reduce the notation of running indexes $1 \leq {}^{\nu}j \leq {}^{\nu}n$ to the short form "j"; extra exclusions $j \neq i$ are written as "$j, -i$".

We denote the index of the closest support point (per axis) with an asterisk

$$i^* := \operatorname*{argmin}_j \|s - a_j\| \tag{4.23}$$

and rewrite Eq. 4.17 and (4.21):

$$
\begin{aligned}
l_i(s,A) &= \begin{cases} Q_{i^*} & \text{if } i = i^* \\ Q_i d_{i^*} & \text{else} \end{cases} \\
\frac{\partial l_i}{\partial s} &= \begin{cases} Q_{i^*} S_{1,i^*} & \text{if } i = i^* \\ Q_i (S_{1,i} d_{i^*} + 1) & \text{else} \end{cases} \\
\frac{\partial^2 l_i}{\partial s^2} &= \begin{cases} Q_{i^*} (S_{1,i^*}^2 - S_{2,i^*}) & \text{if } i = i^* \\ Q_i \left[\left(S_{1,i}^2 - S_{2,i} \right) d_{i^*} + 2 S_{1,i} \right] & \text{else} \end{cases}
\end{aligned}
$$

using: $d_i := s - a_i$

$$
\begin{aligned}
C_i &:= \prod_{j,-i} (a_i - a_j) = const & Q_i &:= \frac{1}{C_i} \prod_{j,-i,-i^*} d_j \\
S_{1,i} &:= \sum_{j,-i,-i^*} \frac{1}{d_j} & S_{2,i} &:= \sum_{j,-i,-i^*} \left(\frac{1}{d_j} \right)^2
\end{aligned}
\tag{4.24}
$$

The interim quotients Q_i and sums $S_{1,i}$ and $S_{2,i}$ are efficiently generated by defining the master product Q_{i^*} (and sums $S_{\cdot,i^*}$ respectively) and working out the specific terms via "synthetic division" and "synthetic subtraction" for all $i \neq i^*$

$$
Q_i = \frac{Q_{i^*}}{d_i}, \quad S_{1,i} = S_{1,i^*} - \frac{1}{d_i}, \text{ and } \quad S_{2,i} = S_{2,i^*} - \left(\frac{1}{d_i} \right)^2. \tag{4.25}
$$

Computing $H(\mathbf{a}, \mathbf{s})$ and its derivatives is now a matter of collecting the proper pre-computed terms.

$$
\begin{aligned}
H(\mathbf{a},\mathbf{s}) &= \textstyle\prod_{\nu=1}^{m} \; l_{i_\nu}({}^{\nu}s, A_\nu) \\
\frac{\partial H(\mathbf{a},\mathbf{s})}{\partial ({}^{\mu}s)} &= \textstyle\prod_{\nu=1}^{m} \begin{cases} \frac{\partial}{\partial ({}^{\mu}s)} l_{i_\nu}({}^{\nu}s, A_\nu) & \text{if } \nu = \mu \\ l_{i_\nu}({}^{\nu}s, A_\nu) & \text{else} \end{cases} \\
\frac{\partial^2 H(\mathbf{a},\mathbf{s})}{\partial ({}^{\mu}s) \partial ({}^{\kappa}s)} &= \textstyle\prod_{\nu=1}^{m} \begin{cases} \frac{\partial^2}{\partial ({}^{\mu}s)^2} l_{i_\nu}({}^{\nu}s, A_\nu) & \text{if } \nu = \mu = \kappa \\ l_{i_\nu}({}^{\nu}s, A_\nu) & \text{if } \nu \neq \mu \text{ and } \nu \neq \kappa \\ \frac{\partial}{\partial ({}^{\mu}s)} l_{i_\nu}({}^{\nu}s, A_\nu) & \text{else} \end{cases}
\end{aligned}
\tag{4.26}
$$

From here the coefficients of the linear system Eq. 4.13 can be assembled

$$\frac{\partial}{\partial\,({}^{\mu}s)}E = \sum_{k=1}^{d} p_k\,[x_k - w_k(\mathbf{s})]\,\frac{\partial w_k(\mathbf{s})}{\partial\,({}^{\mu}s)} \tag{4.27}$$

and

$$\frac{\partial^2}{\partial\,({}^{\mu}s)\partial\,({}^{\kappa}s)}E = \sum_{k=1}^{d} p_k \left\{ \frac{\partial w_k(\mathbf{s})}{\partial\,({}^{\mu}s)}\frac{\partial w_k(\mathbf{s})}{\partial\,({}^{\kappa}s)} - [x_k - w_k(\mathbf{s})]\,\frac{\partial^2 w_k(\mathbf{s})}{\partial\,({}^{\mu}s)\partial\,({}^{\kappa}s)} \right\}. \tag{4.28}$$

Care should be taken to efficiently skip any computing of non-input components k with $p_k = 0$. Later, we present several examples where the PSOM ability to augment the embedding space is extensively used. In Chap. 9 the PSOM will find a hierarchical structure where it is important, that a few hundred output components (d) can be completed without significant operational load.

Other numerical algorithms exist, which optimize the evaluation of special polynomials, for example the recursive *Neville's algorithm* (Press et al. 1988). Those are advantageous for the situation $m = d = 1$. But the requirement to deal with multiple dimensions m and even very large embedding dimensions d makes the presented algorithm superior with respect to computational efficiency.

4.6 Summary

The key points of the "Parameterized Self-Organizing Map" algorithm are:

- The PSOM is the continuous analog of the standard discrete "Self-Organizing Map" and inherits the SOM's unsupervised learning capabilities (Kohonen 1995). It shows excellent generalization capabilities based on continuous *attractor manifolds* instead of just attractor points.

- Based on a strong bias, introduced by structuring the training data in a *topological order*, the PSOM can generalize from very few examples — if this assumed topological model is a good approximation to the system.

- For training data sets with the *known topology* of a multi-dimensional Cartesian grid the map manifold can be constructed *directly*. The resulting PSOM is immediately usable — without any need for time consuming adaptation sequences. This feature is extremely advantageous in all cases where the training data can be sampled actively. As shown later in Capt. 8, in robotics many sensorimotor transformations can be sampled in a structured manner, without any extra cost.

- Independently of how the the initial structure was found, further on-line *fine-tuning* is facilitated by an on-line error minimization procedure. This adaption is required for example, in the case of coarsely sampled or noise-corrupted data, as well as in cases, where the learning system has to adapt to drifting, or suddenly changing mapping tasks.

- The *multi-way mapping* ability of the PSOM is facilitated by the *non-linear auto-associative completion* of partial inputs. The components of the embedding space are selected as inputs by the diagonal elements p_k of the projection matrix $\mathbf{P}$. Furthermore, the coefficients p_k weight the components relative to each other. This choice can be changed on demand.

- The PSOM completion process performs an iterative search for the best-matching parameter location $\mathbf{s}^*$ in the mapping manifold (Eq. 4.7). This is the price for rapid learning. Fortunately, it can be kept small by applying an adaptive second order minimization procedure (Sect. 4.5). In conjunction with an algorithmic formulation optimized for efficient computation and also for high-dimensional problems, the completion procedure converges already in a couple of iterations.

The following chapter will demonstrate the PSOM properties in a number of examples. Later, several application examples will be presented.

Chapter 5

Characteristic Properties by Examples

As explained in the previous chapter, the PSOM builds a parameterized associative map. Employing the described class of generalized multi-dimensional Lagrange polynomials as basis functions facilitates the construction of very versatile PSOM mapping manifolds. Resulting unusual characteristics and properties are exposed in this chapter.

Several aspects find description: topological order introduces *model bias* to the PSOM and strongly influences the interpretation of the training data; the influence of non-regularities in the process of sampling the training data is explored; "topological defects" can occur, e.g. if the correspondence of input and output data is mistaken; The use of basis polynomials affects the PSOM extrapolation properties. Furthermore, in some cases the given mapping task faces the best-match procedure with the problem of non-continuities or multiple solutions.

Since the visualization of multi-dimensional mappings embedded in even higher dimensional spaces is difficult, the next section illustrates stepwise several PSOM examples. They start with small numbers of nodes.

5.1 Illustrated Mappings – Constructed From a Small Number of Points

The first mapping example is a two-dimensional ($m = 2$) PSOM mapping manifold with 3×3 reference vectors in a four-dimensional ($d = 4$) dimen-

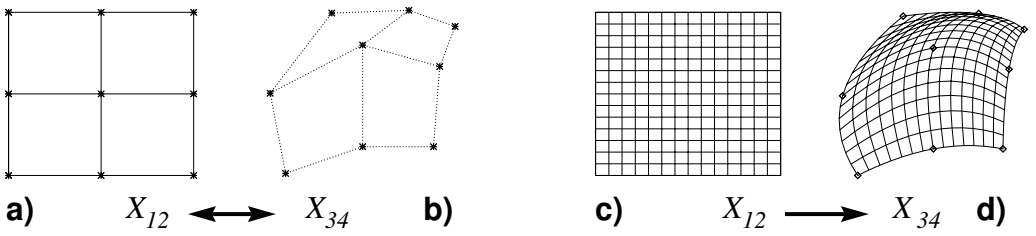


Figure 5.1: A $m = 2$ dimensional PSOM with 3×3 training vectors in a $d = 4$ dimensional embedding space X. *a–b)* Initial training data, or reference vectors $\mathbf{w_a}$, connected by lines and projected in X_{12} and X_{34}. *c–d)* Projections of a 15×15 test grid for visualizing the embedding manifold $M \in \mathbb{R}^4$.

sional embedding space X. The first two components are the input space $X^{\text{in}} = X_{12}$ the last two are the output-space $X^{\text{out}} = X_{34}$. The subscript indicates here the indented projection of the embedding space.

Fig. 5.1a shows the reference vectors drawn in the input sub-space X_{12} and Fig. 5.1b in the output sub-space X_{34} as two separate projections. The invisible node spacing $\mathbf{a} \in \mathbf{A}$ is here, and in the following examples, equidistantly chosen. For the following graphs, the embedding space contains a m dimensional sub-space X_s where the training vector components are set equal to the internal node locations $\mathbf{a}$ in the rectangular set $\mathbf{A}$. The PSOM completion in X_s is then equivalent to the internal mapping manifold location $\mathbf{s}$, here $X_s = X_{12}$ in Fig. 5.1a+c. Since the PSOM approach makes it easy to augment the embedding space X, this technique is generally applicable to visualize the embedding manifold M: A rectangular test-grid in X_s is given as input to the PSOM and the resulting completed d-dimensional grid $\{\mathbf{w}(\mathbf{s})\}$ is drawn in the desired projection.

Fig. 5.1c displays the 15×15 rectangular grid and on the right Fig. 5.1d its image, the output space projection X_{34}. The graph shows, how the embedding manifold nicely picks the **curvature information** found in the connected training set of Fig. 5.1. The edges between the training data points $\mathbf{w_a}$ are the visualization of the essential topological information drawn from the assignment of $\mathbf{w_a}$ to the grid locations $\mathbf{a}$.

Fig. 5.2 shows, what a 2×2 PSOM can do with only four training vectors $\mathbf{w_a}$. The embedding manifold M now non-linearly interpolates between the given support points (which are the same corner points as in the previous figure). Please note, that the mapping is already non-linear, since

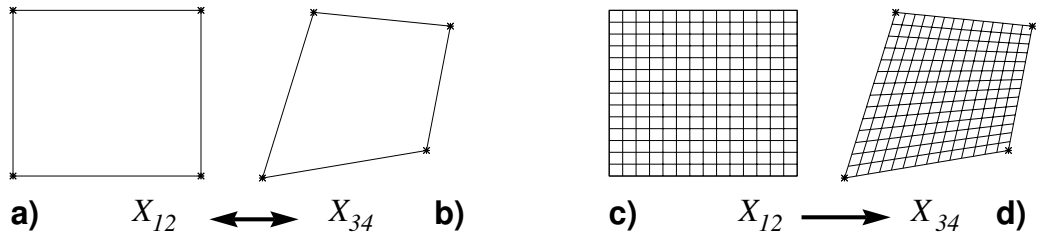


Figure 5.2: A $m = 2$ dimensional PSOM with 2×2 training vectors in a $d = 4$ dimensional embedding space X with the same corner training points as in the previous 3×3 figure.

parallel lines are not kept parallel. Any bending of M has disappeared. To capture the proper "curvature" of the transformation, one should have at least one further point, between the two endpoints of a mapping interval.

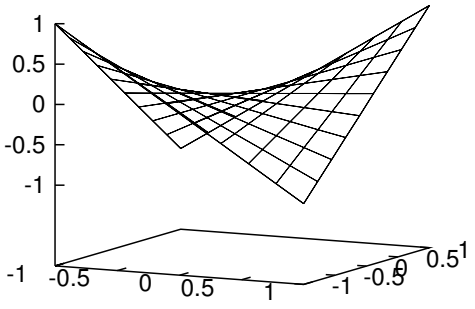


Figure 5.3: Isometric projection of the $d = 3$, $m = 2$ dimensional manifolds M. The 2×2 PSOM manifold spans like a soap film over the four cornering reference vectors $\mathbf{w_a}$.

Fig. 5.3 visualizes the mapping capabilities of a $m = 2$ dimensional mapping manifold spanned by 2×2 training vectors in the 3 D embedding space X.. The resulting mapping belongs to the "harmonic functions" with a zero second derivative ($\nabla_{\mathbf{s}}\nabla_{\mathbf{s}}\mathbf{w}(\mathbf{s}) = 0$ source free) and has the characteristics of a "soap film membrane" spanned over the grid.

Fig. 5.4 illustrates the same mapping capabilities for the $m = 3$ dimensional mapping manifold spanned by the 2×2×2 training vectors. The nodes (corner vertices) span the m-dimensional *hypercube* configuration with 2^m nodes.

Summarizing, even with very small numbers of nodes, the PSOM manifold has very rich mapping properties. Two and three-dimensional PSOM were shown, with two and three nodes per axes. Using multi-dimensional

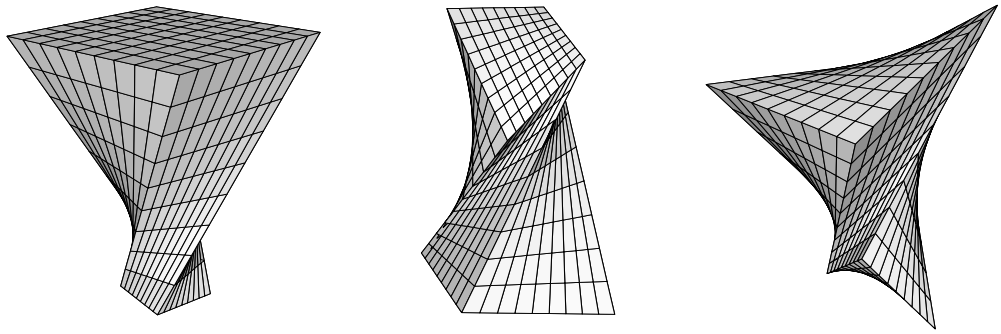


Figure 5.4: Three examples of $m = 3$ dimensional manifolds M of a PSOM with 2×2×2 training vector points. They are shown as perspective surface plots of a grid spanned by the eight corner reference vectors $\mathbf{w_a}$.

test grids the high-dimensional embedded manifolds can be visualized. Selecting a good underlying node model is the first - and a very important step. If a small training set is desired, the presented results suggest to start with a PSOM with three nodes per axis. If one expects a linear dependence in one degree of sampling, two nodes are sufficient.

5.2 Map Learning with Unregularly Sampled Training Points

Here, we want to explore the PSOM mapping behavior in case of the training samples not being drawn from an exact regular grid, but instead from a "roughly regular" grid, sampled with some "jitter". We consider the example of a "barrel shaped" mapping $\mathbb{R}^3 \rightarrow \mathbb{R}^3$, given by

$$\begin{aligned} x_4 &= \kappa x_1 - x_3/4 \\ x_5 &= \kappa x_2 \\ x_6 &= x_3 \\ \kappa &= \sqrt{1 - \frac{x_3^2}{4} - \frac{1}{3}\sqrt{x_1^2 + x_2^2}} \end{aligned} \tag{5.1}$$

Eq. 5.1 maps the unit cube [-1,1]^3 into a barrel shaped region, shown in Fig. 5.5. The first four plots in the upper row illustrate the mapping if the

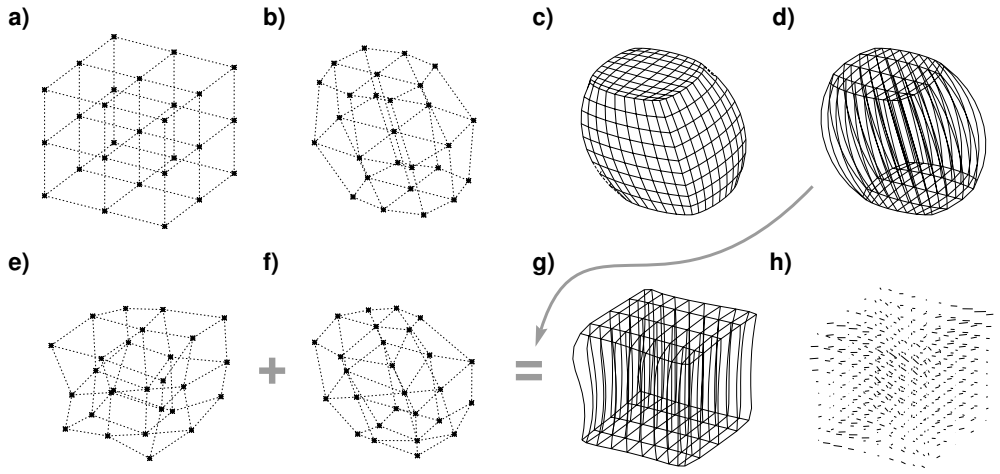


Figure 5.5: The jittery barrel mapping (Eq. 5.1, $X_{123} : (a, e, g, h),\ X_{456} : (b, c, d, f)$ projections). The training data set is asterisk marked shown as the X_{123} *(a)* and X_{456} *(b)* projection and the mapping manifold M (m=3) as surface grid plot in *(c)*. To reveal the internal structure of the mapping inside the barrel, a " filament" picture is drawn by the vertical lines and the horizontal lines connecting only the points of the 10$\times$ 5$\times$ 10 grid in the top and bottom layer *(d)*.
(e)(f) If the samples are not taken on a regular grid in X_{123} but with a certain jitter, the PSOM is still able to perform a good approximation of the target mapping: (g) shows the image of the data set (d) taken as input. The plot (h) draws the difference between the result of the PSOM completion and the target value as lines.

PSOM training samples are taken from the rectangular grid of the asterisk markers depicted in Fig. 5.5ab.

The succeeding plots in the lower row present the situation, that the PSOM only learned the randomly shifted sampling positions Fig. 5.5ef. The mapping result is shown in the rightmost two plots: The $3\times3\times3$ PSOM can reconstruct the goal mapping fairly well, illustrating that there is no necessity of sampling PSOM training points on any precise grid structure. Here, the correspondence between X and S is weakened by the sampling, but the topological order is still preserved.

5.3 Topological Order Introduces Model Bias

In the previous sections we showed the mapping manifolds for various topologies which were already given. This stage of obtaining the topological correspondence includes some important aspects:

1. Choosing a topology is the first step in interpreting a given data set.

2. It introduces a strong *model bias* and reduces therefore the variance. This leads – in case of the correct choice – to an improved generalization.

3. The topological information is mediated by the basis functions. All examples shown here, build on the high-dimensional extension to approximation polynomials. Therefore, the examples are special in the sense, that the basis functions are varying only within their class as described in Sec. 4.5. Other topologies can require other types of basis functions.

To illustrate this, let us consider a 2 D example with six training points. If only such a small data set is available, one may find several topological assignments. In Fig. 5.6 the six data points $\mathbf{w_a}$ are drawn, and two plausible, but different assignments to a $3\times$ 2 node PSOM are displayed.

In the vicinity of the training data points the mapping is equivalent, but the regions interpolating and extrapolating differ, as seen for the cross-marked example query point. Obviously, it needs further information to resolve this ambiguity in topological ordering. Active sampling could resolve this ambiguity.

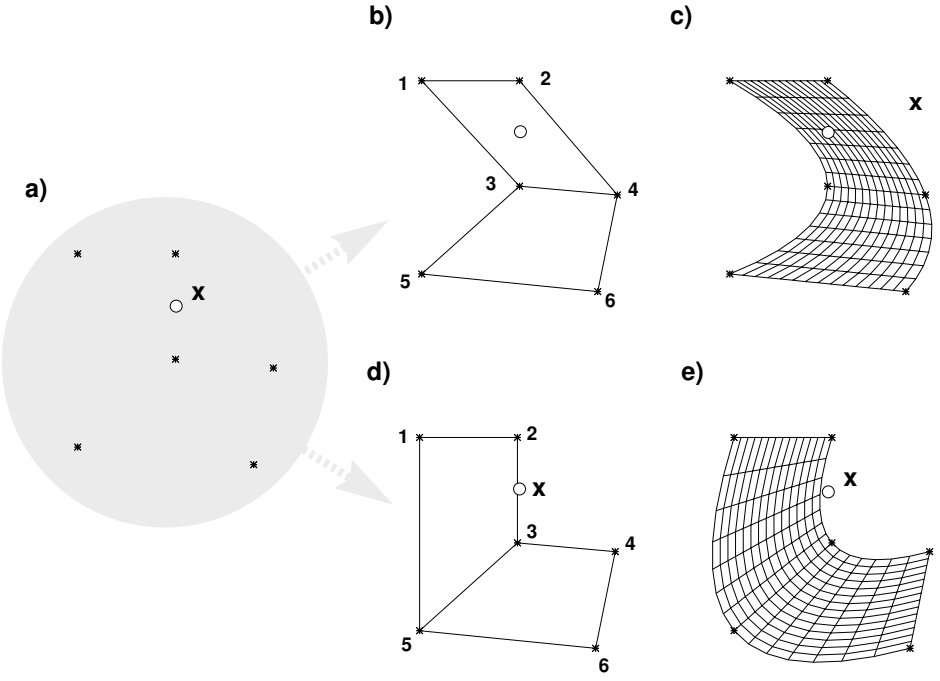


Figure 5.6: The influence of the topological ordering. In pathological situations, one data set can lead to ambiguous topologies. The given six data points *(a)* in X^{in} can be assigned to more than one unique topology: obviously, both $3\times$ 2 grids, *(b)* and *(c)* are compatible. Without extra knowledge, both are equivalently suitable. As seen in *(d)(e)*, the choice of the topology can partition the input space in rather different regions of inter- and extrapolation. For example, the shown query point x lie central between the four points 1,2,3,4 in *(b)*, and for the topology in *(d)*, points 2,3 are much closer to x than points 1 and 4. This leads to significantly different interpretation of the data.

If we have insufficient information about the correct node assignment and are forced to make a some specification, we may introduce "topological defects".

5.4 "Topological Defects"

What happens if the training vectors $\mathbf{w_a}$ are not properly assigned to the node locations $\mathbf{a}$? What happens, if the topological order is mistaken and the neighborhood relations do not correspond the closeness in input space? Let us consider here the case of exchanging two node correspondences.

Fig. 5.7a-b and Fig. 5.7c-d depict two previous examples, where two reference vectors got swapped. One the left side, the 2×2 PSOM exhibits a complete twist, pinching all vertical lines. The right pictures show, how the embedding manifold of the 3×3 PSOM in Fig. 5.1 becomes distorted in the lower right part. The PSOM manifold follows nicely all "topological defects" given and resembles an "elastic net" or cover, pinned at the supporting training vectors.

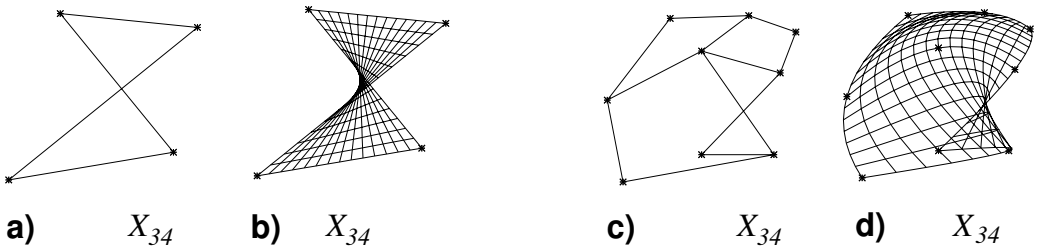


Figure 5.7: "Topological defects" by swapping two training vectors: *a–b* the 2×2 PSOM of Fig. 5.2 and *c–d* the 3×3 PSOM of Fig. 5.1

Note, that the node points are still correctly mapped, as one can expect from Eq. 4.2, but in the inter-node areas the PSOM does not generalize well. Furthermore, if the opposite mapping direction is chosen, the PSOM has in certain areas more than one unique best-match solution $\mathbf{s}^*$. The result, found by Eq. 4.4, will depend on the initial start point $\mathbf{s}_{t=0}$.

Can we algorithmically test for topological defects? Yes, to a certain extent. Bauer and Pawelzik (1991) introduced a method to compare the

"faithfulness" of the mapping from the embedding input space to the parameter space. The *topological*, or *"wavering" product* gives an indication on the presence of topological defects, as well as too small or too large mapping manifold dimensionality.

As already pointed out, the PSOM draws on the curvature information drawn from the topological order of the training data set. This information is visualized by the connecting lines between the reference vectors $\mathbf{w_a}$ of neighboring nodes. How important this relative order is, is emphasized by the shown effect if the proper order is missing, as seen Fig. 5.7.

5.5 Extrapolation Aspects

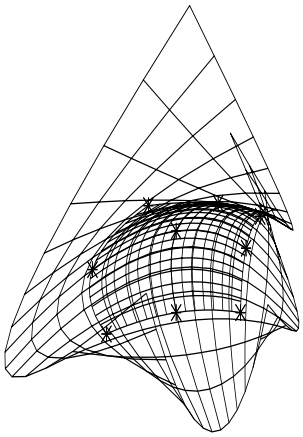


Figure 5.8: The PSOM of Fig. 5.1d in X_{34} projection and in superposition a second grid showing the extrapolation beyond the training set (190 %).

Now we consider the extrapolation areas, beyond the mapping region of the convex hull of the reference vectors. Fig. 5.8 shows a superposition of the original 15×15 test grid image presented in Fig. 5.1d and a second one enlarged by the factor 1.9. Here the polynomial nature of the employed basis functions exhibits an increasingly curved embedding manifold M with growing "remoteness" to the trained mapping area. This property limits the extrapolation abilities of the PSOM, depending on the particular distribution of training data. The beginning in-folding of the map, e.g. seen at the lower left corner in Fig. 5.8 demonstrates further that M shows multiple solutions (Eq. 4.4) for finding a best-match in X_{34}. In general, polynomials ($s \mapsto x$) of even order (uneven node number per axes) will show multiple solutions. Uniqueness of a best-match solution

($\mathbf{s}^*$) is not guaranteed. However, for well-behaved mappings the corresponding $\mathbf{s}^*$ values are "far away", which leads to the advise: *Be suspicious, if the best-match* $\mathbf{s}^*$ *is found far outside the given node-set* $\mathbf{A}$.

Depending on the particular shape of the embedding manifold M, an unfortunate gradient situation may occur in the vicinity of the border training vectors. In some bad cases the local gradient may point to another, far outside local minimum, producing a misleading completion result. Here the following *heuristic* proved useful:

In case the initial best-match node $\mathbf{a}^*$ (Sect. 4.3) has a marginal surface position in $\mathbf{A}$, the minimization procedure Eq. 4.4 should be started at a shifted position

$$\mathbf{s}_{t=0} = \mathbf{a}^* + \Delta\mathbf{a}_\perp. \tag{5.2}$$

The start-point correction $\Delta\mathbf{a}_\perp$ is chosen to move the start location inside the node-set hyper-box, perpendicular to the surface. If $\mathbf{a}^*$ is an edge or corner node, each surface normal contributes to $\Delta\mathbf{a}_\perp$. The shift length is uncritical: one third of the node-set interval, but maximal one inter-node distance, is reasonable. This start-point correction is computationally negligible and helps to avoid critical border gradient situations, which could otherwise lead to another, undesired remote minimum of Eq. 4.4.

5.6 Continuity Aspects

The PSOM establishes a smooth and continuous embedding manifold M : $\mathbf{s} \rightarrow \mathbf{w}(\mathbf{s})$. However, the procedure of associative completion bears several cases of non-continuous responses of the PSOM.

They depend on the particular mapping and on the selection of the input sub-space X^{in}, respectively $\mathbf{P}$. The previous section already exhibited the extrapolation case, where multiple solutions occured. What are important cases, where discontinuous PSOM responses are possible?

Over-specified Input: Consider the case, where the specified input sub-space X^{in} over-determines the best-match point in the parameter manifold S. This happens if the dimensionality of the input space is higher than the parameter manifold S: $\dim(X^{in}) = |I| > m$.

Fig. 5.9 illustrates this situation with a ($m = 1$) one-dimensional PSOM and displays the two input space dimensions X^{in} together

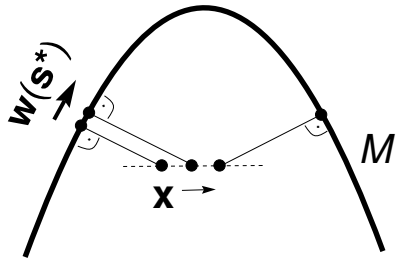


Figure 5.9: The PSOM responses $\mathbf{w}(\mathbf{s}^*)$ for a sequence of inputs $\mathbf{x}$ (dotted line) lead to a "jump" in the resulting best-match $\mathbf{s}^*$ at the corresponding completion $\mathbf{w}(\mathbf{s}^*)$.

with the projection of the embedding manifold $\mathbf{P}M$. Assume that the sequence of presented input vectors $\mathbf{x}$ (2 D!) varies on the indicated dotted line from left to right. The best-match location $\mathbf{P}\mathbf{w}(\mathbf{s}^*)$, determined as the closest point to $\mathbf{x}$, is moving up the arch-shaped embedding manifold M. At a certain point, it will jump to the other branch, obviously exhibiting a discontinuity in $\mathbf{s}^*$ and the desired association $\mathbf{w}(\mathbf{s}^*)$.

Multiple Solutions: The next example Fig. 5.10 depicts the situation $|I| = m = 1$. A one-dimensional four-node PSOM is employed for the approximation of the mapping $x_1 \mapsto x_2$. The embedding manifold $M \subset X = \mathbb{R}^2$ is drawn, together with the reference vectors $\mathbf{w_a}$.

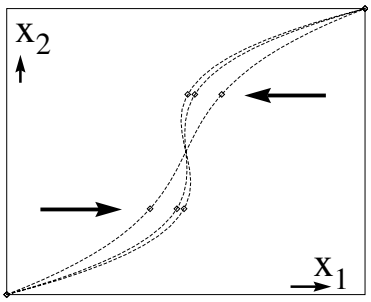


Figure 5.10: The transition from a continuous to a non-continuous response. A four node, one-dimensional PSOM in the two-dimensional embedding space X. The two middle reference vector positions $\mathbf{w}_a$ are increasingly shifted, see text.

The middle two reference vectors are increasingly shifted in opposite horizontal directions, such, that M becomes more and more a *S*-shaped curve. If the curve gets a vertical tangent, a "phase transition" will be encountered. Beyond that point, there are obviously

three compatible solutions $\mathbf{s}^*$ fulfilling Eq. 4.4, which is a *bifurcation* with respect to the shift operation and a discontinuity with respect to the mapping $x_1 \mapsto x_2$.

In view of the pure x_1 projection, the final stage could be interpreted as "topological defect" (see Sec. 5.4). Obviously, this consideration is relative and depends very much on further circumstances, e.g. information embedded in further X components.

5.7 Summary

The construction of the parameterized associative map using approximation polynomials shows interesting and unusual mapping properties. The high-dimensional multi-directional mapping can be visualized by the help of test-grids, shown in several construction examples.

The structure of the prototypical training examples is encoded in the topological order, i.e. the correspondence to the location (a) in the mapping manifold S. This is the source of curvature information utilized by the PSOM to embed a smooth continuous manifold in X. However, in certain cases input-output mappings are non-continuous. The particular manifold shape in conjunction with the associative completion and its optional partial distance metric allows to select sub-spaces, which exhibit multiple solutions. As described, the approximation polynomials (Sec. 4.5) as choice of the PSOM basis function class bears the particular advantage of multi-dimensional generalization. However, it limits the PSOM approach in its extrapolation capabilities. In the case of a low-dimensional input sub-space, further solutions may occur, which are compatible to the given input. Fortunately, they can be easily discriminated by their their remote $\mathbf{s}^*$ location.

Chapter 6

Extensions to the Standard PSOM Algorithm

From the previous examples, we clearly see that in general we have to address the problem of multiple minima, which we combine with a solution to the problem of local minima. This is the subject of the next section.

In the following, section 6.2 describes a way of employing the multi-way mapping capabilities of the PSOM algorithm for additional purposes, e.g. in order to simultaneously comply to auxiliary constraints given to resolve redundancies.

If an increase in mapping accuracy is desired, one usually increases the number of free parameters, which translates in the PSOM method to more training points per parameter axis. Here we encounter two shortcomings with the original approach:

- The choice of polynomials as basis functions of increasing order leads to unsatisfactory convergence properties. Mappings of sharply peaked functions can force a high degree interpolation polynomial to strong oscillations, spreading across the entire manifold.

- The computational effort per mapping manifold dimension grows as $O(\prod_{\nu=1}^{m} n_\nu^2)$ for the number of reference points n_ν along each axis ν. Even with a moderate number of sampling points along each parameter axis, the inclusion of all nodes in Eq. 4.1 may still require too much computational effort if the dimensionality of the mapping manifold m is high ("curse of dimensionality").

Both aspects motivate two extensions to the standard PSOM approach: the "Local-PSOMs" and the "Chebyshev-spaced PSOM", which are the focus of the Sec. 6.3 and 6.4.

6.1 The "Multi-Start Technique"

The *multi-start technique* was developed to overcome the multiple minima limitations of the simpler best-match start procedure adopted so far (see Sec. 4.3).

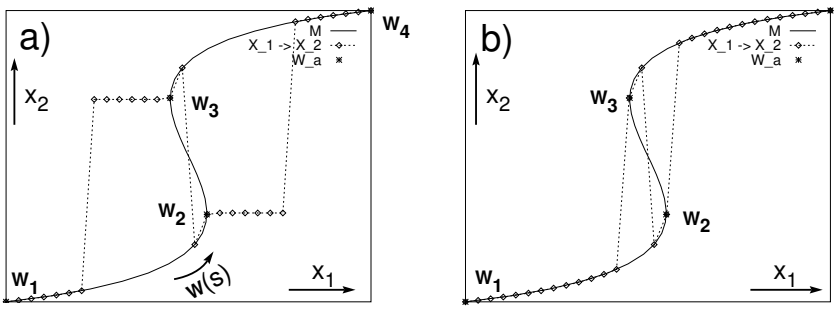


Figure 6.1: The problem of local and multiple minima can be solved by the **multi-start technique**. The solid curve shows the embedded one-dimensional ($m = 1$) PSOM manifold, spanned by the four asterisks marked reference vectors $\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4\}$ in $\mathbb{R}^2$. The dashed line connects a set of diamont-marked PSOM mappings $x_1 \rightarrow x_2$.
(a) A pathological situation for the standard approach: depending on the starting location $\mathbf{s}_{t=0}$, the best-match search can be trapped in a local minimum.
(b) The multi-start technique solves the task correctly and can be employed to find multiple solutions.

To understand the rationale behind this technique let us consider the four-node PSOM with the S-shaped manifold introduced before in Fig. 5.10. On the left Fig. 6.1a the diamonds on the dotted line show a set of PSOM mappings $x_1 \mapsto x_2$ ($\mathbf{P}$=diag(1,0)). Starting at the small x_1 values, the best-match procedure finds the first node $\mathbf{w}_1$ as start point. When (after the 7th trial) the third reference vector $\mathbf{w}_3$ gets closer than $\mathbf{w}_1$, the gradient descent iteration starts at $\mathbf{w}_3$ and becomes "trapped" in a local minimum, giving rise to a totally misleading value for x_2. On the other trials this problem

reoccurs except for the middle region and the right side, close to the last reference point.

First, we want to note that the problem can be detected here by monitoring the residual distance $dist(\cdot)$ (respectively the cost function $E(\mathbf{s})$) which is here the horizontal distance of the found completion (close to $\mathbf{w}_3$) and the input $\mathbf{x}$.

Second, this problem can be solved by re-starting the search: suitable restart points are the distance-ranked list of node locations a found in the first place (e.g. the 10th points probes the start locations at node 3,2,1,4). The procedure stops, if a satisfying solution (low residual cost function) or a maximum number of trials is reached. Fig. 6.1b demonstrates this multi-start procedure and depicts the correctly found solutions.

In case the task is known to involve a consecutive sequence of query points, it is perfectly reasonable to place the previous best-match location $\mathbf{s}^*$ at the head position of the list of start locations.

Furthermore, the multi-start technique is also applicable to find multiple best-match solutions. However, extra effort is required to find the complete list of compatible solutions. E.g. in the middle region of the depicted example Fig. 6.1, at least two of the three solutions will be found.

6.2 Optimization Constraints by Modulating the Cost Function

The best-match minimization procedure Eq. 4.4 can be employed to control the optimization mechanism in various directions. The multi-start technique introduced before approaches suitable solutions from different starting locations $\mathbf{s}_0^*$, but using a constant cost function Eq. 4.10

$$E(\mathbf{s}) = \tfrac{1}{2} dist\left(\mathbf{x}, \mathbf{w}(\mathbf{s})\right) = \tfrac{1}{2} \sum_{k=1}^{d} p_k \left[x_k - w_k(\mathbf{s})\right]^2 .$$

An interesting possibility is to modulate this cost function during the best-match iteration process. This means that the weight factors p_k, which define the distance metric, are varied in order to influence the iteration path in a desired way. This approach can be considered, e.g. to avoid local minima with already known locations.

Furthermore, this option can be used to resolve redundancies. Such an example is presented later in the context of the 6 D robot kinematics in Sec. 8.4.

There the situation arises that the target is under-specified in such a way that a *continuous solution space* exists which satisfies the goal specification. In this case, the PSOM will find a solution depending on the initial starting condition $\mathbf{s}_0^*$ (usually depending on the node with the closest reference vector $\mathbf{a}^*$) which might appear somehow arbitrary.

In many situations, it is of particular interest to intentionally utilize these kinds of redundancies, in order to serve auxiliary goals. These are goals of second priority, which might contradict a primary goal. Here the PSOM offers an elegant and intrinsic solutions mechanism.

Auxiliary goals can be formulated in additional cost function terms and can be activated whenever desired. The cost function terms can be freely constructed with various functional forms and are supplied during the learning phase of the PSOM. Remember, that the PSOM input subspace selection mechanism (p_k) facilitates easy augmentation of the embedding space X with extraneous components, which do not impair the normal operation.

For example, for positioning a robot manipulator at a particular position in the 3 D workspace, the 6 degress-of-freedom (DOF) of the manipulator are under-constrained. There are infinite many solutions – but, how to encode some sort of "reasonableness" within the current situation? In Sec. 8.4, different secondary goals are discussed: e.g. finding the shortest path, or finding an arm configuration being advantageous for further actions. They can be elegantly implemented using the same PSOM.

6.3 The Local-PSOM

In section 3.2 the distinction between local and global models was introduced. As it turns out, the PSOM approach can be viewed as an intermediate position. It is a global model, but in the vicinity of reference points it behaves locally. Still, to overcome the difficulties inherited from high-degree polynomial basis functions, we can look for additional control over the locality of the PSOM.

The concept of Local-PSOMs precisely addresses this issue and allows

to combine the favorable properties of low-degree polynomial PSOMs with the possibility of exploiting the information if many grid points are given. The basic idea is to restrict the number of nodes included in the computation. We suggest to *dynamically* construct the PSOM *only on a sub-grid* of the full training data set. This sub-grid is (in the simplest case) always centered at the reference vector $\mathbf{w}_{\mathbf{a}^*}$ that is closest to the current input $\mathbf{x}$ (primary start node location). The use of the sub-grid leads to lower-degree polynomials for the basis functions and involves a considerably smaller number of points in the sum in Eq. 4.1. Thus, the resulting *Local-PSOMs* ("L-PSOMs") provide an attractive scheme that overcomes both of the shortcomings pointed out in the introduction of this chapter.

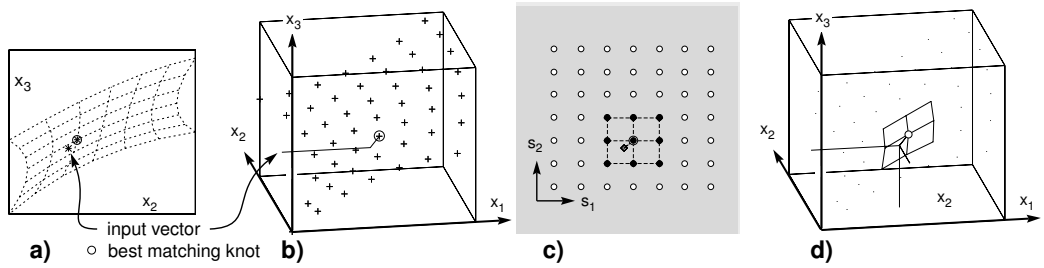


Figure 6.2: *a–d:* The Local-PSOM procedure. The example task of Fig. 4.1, but this time using a 3×3 local PSOM of a 7×7 training set. *(a–b)* The input vector (x_2, x_3) selects the closest node $\mathbf{a}^*$ (in the now specified input space). The associated 3×3 node sub-grid is indicated in *(c)*. The minimization procedure starts at its center $\mathbf{s} = \mathbf{a}^*$ and uses only the PSOM constructed from the 3×3 sub-grid. *(d)* displays the mapping result $\mathbf{w}(\mathbf{s}^*)$ in X, together with the selected sub-grid of nodes in orthonormal projection. The light dots indicate the full set of training nodes. (For the displayed mapping task, a 2×2 PSOM would be appropriate; the 7×7 grid is for illustrative purpose only.)

Fig. 6.2a–d explains the procedure in the the context of the previous simple cube scenario introduced in Sec. 4.1. One single input vector is given in the (x_2, x_3) plane (left cube side), shown also as a projection on the left Fig. 6.2a. The reference vector $\mathbf{w}_\mathbf{a}$ that is closest to the current input $\mathbf{x}$ will serve as the center of the sub-grid. The indicated 3×3 node grid is now used as the basis for the completion by the Local-PSOM.

Continuity is a general problem for local methods and will be discussed after presenting an example in the next section.

6.3.1 Approximation Example: The Gaussian Bell

As a first example to illustrate the Local-PSOMs we consider the Gaussian bell function

$$x_3 = \exp\left(-\frac{x_1^2 + x_2^2}{\lambda^2}\right), \tag{6.1}$$

with $\lambda = 0.3$ chosen to obtain a "sharply" curved function in the square region $[-1, 1]^2$. Fig. 6.3 shows the situation for a 5×5 training data set, Fig. 6.3b, equidistantly sampled on the test function surface plotted in Fig. 6.3a.

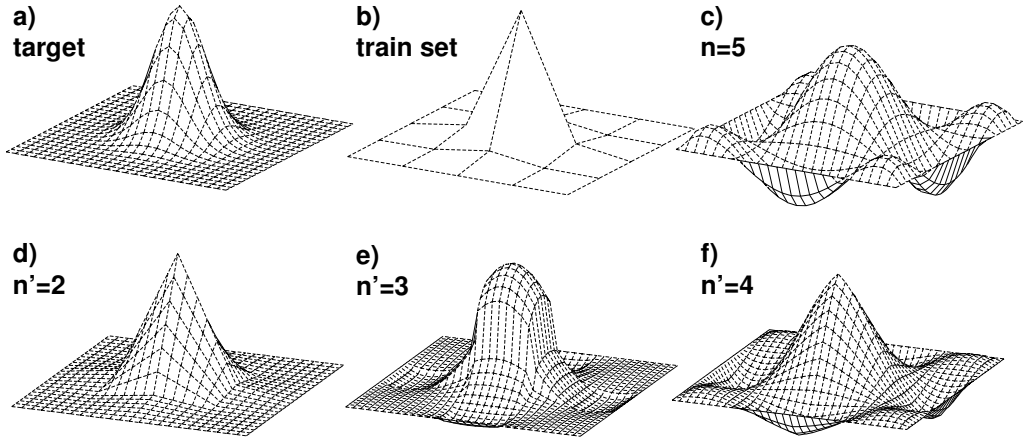


Figure 6.3: The Local-PSOM approach with various sub-grid sizes. Completing the 5×5 sample set (b) Gaussian bell function (a) with the local PSOM approach using sub-grid sizes $n'\times n'$, with $n' = 2, 3,$ and 4; see text.

Fig. 6.3c shows how the full set PSOM completes the embedding manifold M. The marginal oscillations in between the reference vectors $\mathbf{w_a}$ are a product of the polynomial nature of the basis functions. Fig. 6.3d-e is the image of the $2\times$ 2, $3\times$ 3, and the $4\times$ 4 local PSOM.

6.3.2 Continuity Aspects: Odd Sub-Grid Sizes n' Give Options

The choice of the sub-grid needs some more consideration. For example, in the case $n' = 2$ the best-match $\mathbf{s}^*$ should be inside the interval swanned

by the two selected nodes. This requires to shift the selected node window, if $\mathbf{s}^*$ is outside the interval. This happens e.g. when starting at the best-match node $\mathbf{a}^*$, the "wrong" next neighboring node is considered first (left instead of right).

Fig. 6.3d illustrates that the resulting mapping is continuous – also along edge connecting reference vectors. Because of the factorization of the basis functions the polynomials are continuous at the edges, but the derivatives perpendicular to the edges are not, as seen by the sharp edges. An analogous scheme is also applicable for all higher *even* numbers of nodes n'.

What happens for *odd* sub-grid sizes? Here, a central node exists and can be fixated at the search starting location $\mathbf{a}^*$. The price is that an input, continuously moving from one reference vector to the next will experience halfway that the selected sub-grid set changes. In general this results in a *discontinuous* associative completion, which can be seen in Fig. 6.3e which coincides with Fig. 6.4a).

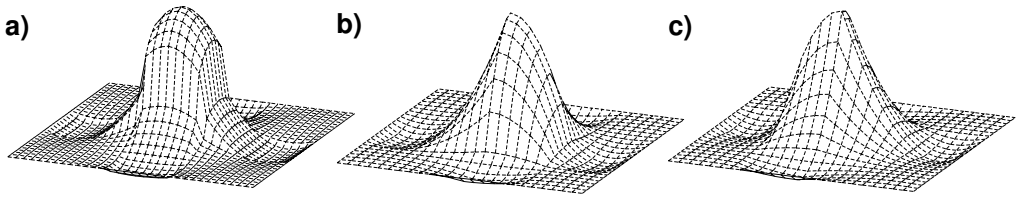


Figure 6.4: Three variants to select a 3×3 sub-grid (in the previous Problem Fig. 6.4) lead to different approximations: *(a)* the standard fixed sub-grid selection and *(b)(c)* the *continuous* but asymmetric mappings, see text.

However, the selection procedure can be modified to make the mapping continuous: the essence is to move the locations where the sub-grid selection changes to the node edge line (in S). Fig. 6.4bc shows the two alternatives: odd node numbers have two symmetric center intervals, of which one is selected to capture the best-match $\mathbf{s}^*$.

Despite the symmetric problem stated, case Fig. 6.4b and Fig. 6.4c give *asymmetric* (here mirrored) mapping results. This can be understood if looking at the different 3×3 selections of training points in Fig. 6.3b. The round shoulder (Fig. 6.4b) is generated by the peak-shaped data sub-set, which is symmetric to the center. On the other hand, the "steep slope part" bears no information on the continuation at the other side of the peak.

Here again, the similarities to Kohonen's SOM algorithm become clearly visible: the closeness of the input $\mathbf{x}$ to the reference vectors $\mathbf{w_a}$ is the important, determining factor. Here, this mechanism also tessellates the input space into Voronoi cells of continuous, here parameterized, curved maps. Similarly, the previously described smoothing schemes for discrete cells could be applied to smooth the remaining discontinuities as well.

6.3.3 Comparison to Splines

The concept of Local-PSOM has similarities with the spline technique (Stoer and Bulirsch 1980; Friedman 1991; Hämmerlin and Hoffmann 1991). Both share the internal piecewise construction of a polynomial on a set of neighboring nodes.

For the sub-grid size $n' = 2$ the local PSOM becomes actually identical to the multidimensional extension of the linear splines, see Fig. 6.3d. Each patch resembles the m-dimensional "soap-film" illustrated in Fig. 5.3 and 5.4.

An alternative was suggested by Farmer and Sidorowich (1988). The idea is to perform a linear simplex interpolation in the $|I|$–dimensional input space X^{in} by using the $|I|$+1 (with respect to the input vector $\mathbf{x}$) closest support points found, which span the simplex containing $\mathbf{Px}$. This lifts the need of topological order of the training points, and the related restriction on the number of usable points.

For $n' = 3$ and $n' = 4$ the local PSOM concept resembles the quadratic and the cubical spline, respectively. In contrast to the spline concept that uses piecewise assembled polynomials, we employ one single, dynamically constructed interpolation polynomial with the benefit of usability for multiple dimensions m.

For $m = 1$ and low d the spline concept compares favorably to the polynomial interpolation ansatz, since the above discussed problem of asymmetric mapping does not occur: at each point 3 (or 4, respectively) polynomials will contribute, compared with one single interpolation polynomial in a selected node sub-grid, as described.

For $m = 2$ the bi-cubic, so-called *tensor-product spline* is usually computed by row-wise spline interpolation and a column spline over the row interpolation results (Hämmerlin and Hoffmann 1991). The procedure is computationally very expensive and has to be independently repeated for

each component d.

For $m > 2$ the tensor splines are simple to extend in theory, but in practice they are not very useful (Friedman 1991). Here, the PSOM approach can be easily extended as shown for the case $m = 6$ later in Sec. 8.2.

6.4 Chebyshev Spaced PSOMs

An alternative way to deal with the shortcomings addressed in the introduction of this chapter, is the use of an improved scheme for constructing the internal basis functions $H(\mathbf{a}, \mathbf{s})$. As described (in Sec. 4.5), they are constructed by products of Lagrange interpolation polynomials on a rectangular grid of nodes in the parameter manifold. So far, a regular equidistant division of an interval (e.g. [0,1] in Fig. 4.3) was suggested for placing the nodes on each parameter manifold axis.

As is well-known from approximation theory, an equidistant grid point spacing is usually not the optimal choice for minimizing the approximation error (Davis 1975; Press et al. 1988). For polynomial approximation on an interval [-1,1], one should choose the support points a_j along each axis of $\mathbf{A}$ at locations given by the zeros a_j of the Chebyshev polynomial $T_n(a)$

$$T_n(a) = \cos\left(n \arccos\ (a)\right). \tag{6.2}$$

The (fixed location of the) n zeros of T_n are given by

$$a_j = \cos\left(\frac{j - \frac{1}{2}}{n}\pi\right). \tag{6.3}$$

The Chebyshev polynomials are characterized by having all minima and maxima of the same amplitude, which finally leads to a smaller bound of the error polynomial describing the residual deviation. Fig. 6.5a displays for example the Chebyshev polynomial T_{10}. A characteristic feature is the increasing density of zeros towards both ends of the interval.

It turns out that this choice of support points can be adopted for the PSOM approach without any increase in computational costs. As we will demonstrate shortly, the resulting *Chebyshev spaced* PSOM ("C-PSOM") tends to achieve considerably higher approximation accuracy as compared to *equidistant spaced* PSOMs for the same number of node points. As a result, the use of Chebyshev PSOMs allows a desired accuracy to be achieved

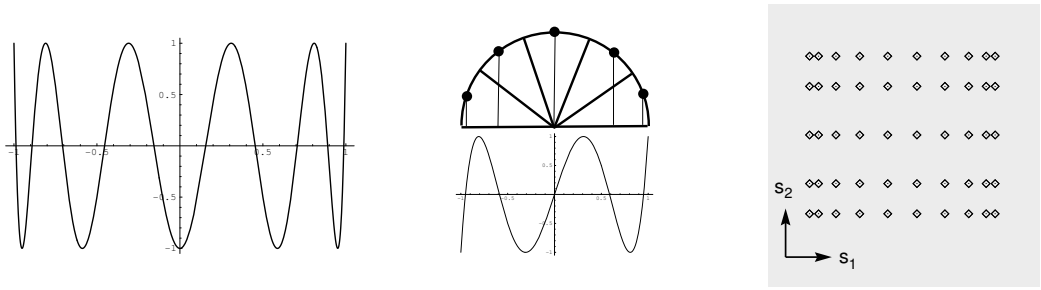


Figure 6.5: *(left:)* The Chebyshev polynomial T_{10} (Eq. 6.2). Note the increased density of zeros towards the approximation interval bounds .
(center:) T_5 below a half circle, which is cut into five equal pieces. The base projection of each arc mid point falls at a zero of T_5 (Eq. 6.3.)
(right:) Placement of 5$\times$ 10 nodes $\mathbf{A} \in S$, placed according to the zeros a_j of the Chebyshev polynomial T_5 and T_{10}.

with a smaller number of nodes, leading to the use of lower-degree polynomials together with a reduced computational effort.

In Sec. 6.5.1 and Sec. 8.1 the Chebyshev spacing is used for pre-specifying the location of the nodes $\mathbf{a} \in \mathbf{A}$ as compared to an equidistant, rectangular node-spacing in the m-dimensional parameter space (e.g. Fig. 4.10).

A further improvement is studied in Sect. 8.2. Additionally to the improvement of the internal polynomial manifold construction, the Chebyshev spacing is also successfully employed for sampling of the training vectors in the embedding space X. Then, the n_ν (constant) zeros of T_{n_ν} in [-1,1] are scaled to the desired working interval and accordingly the samples are drawn.

6.5 Comparison Examples: The Gaussian Bell

In this section the Gaussian bell curve serves again as test function. Here we want to compare the mapping characteristics between the PSOM versus the Local-Linear-Map (LLM) approach, introduced in chapter 3.

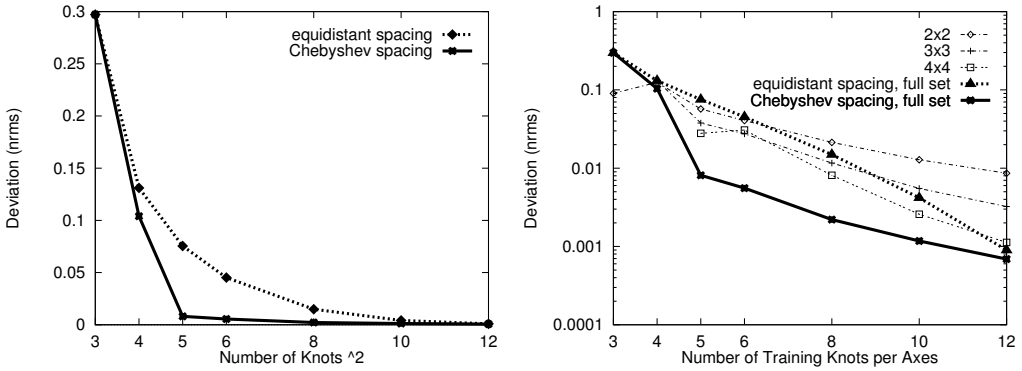


Figure 6.6: a–b; Mapping accuracy of the Gaussian bell function for the presented PSOM variants – in a linear *(left)* and a logarithmic plot *(right)* – versus n the number of training points per axes.

6.5.1 Various PSOM Architectures

To compare the local and Chebyshev spaced PSOMs we return to the Gaussian bell function of Eq. 6.1 with $\lambda = 0.5$ chosen to obtain a "medium sharp" curved function. Using the same $n \times n$ (in x_1, x_2) equidistantly sampled training points we compute the root mean square deviation (RMS) between the goal mapping and the mapping of *(i)* a PSOM with equidistantly spaced nodes, *(ii)* local PSOMs with sub-grid sizes $n' = 2$, 3, 4 (subgrids use equidistant node spacing here as well), and *(iii)* PSOMs with Chebyshev spaced nodes.

Fig. 6.6 compares the numerical results (obtained with a randomly chosen test set of 500 points) versus n. All curves show an increasing mapping accuracy with increasing number of training points. However, for $n > 3$ the Chebyshev spaced PSOM *(iii)* shows a significant improvement over the equidistant PSOM. For $n = 3$, the PSOM and the C-PSOM coincide, since the two node spacings are effectively the same (the Chebyshev polynomials are always symmetric to zero and here equidistant as well.)

In Fig. 6.6 the graphs show at $n = 5$ the largest differences. Fig. 6.7 displays four surface grid plots in order to distinct the mapping performances. It illustrates the 5×5 training node set, a standard PSOM, a C-PSOM, and a (2-of-5)^2 L-PSOM.

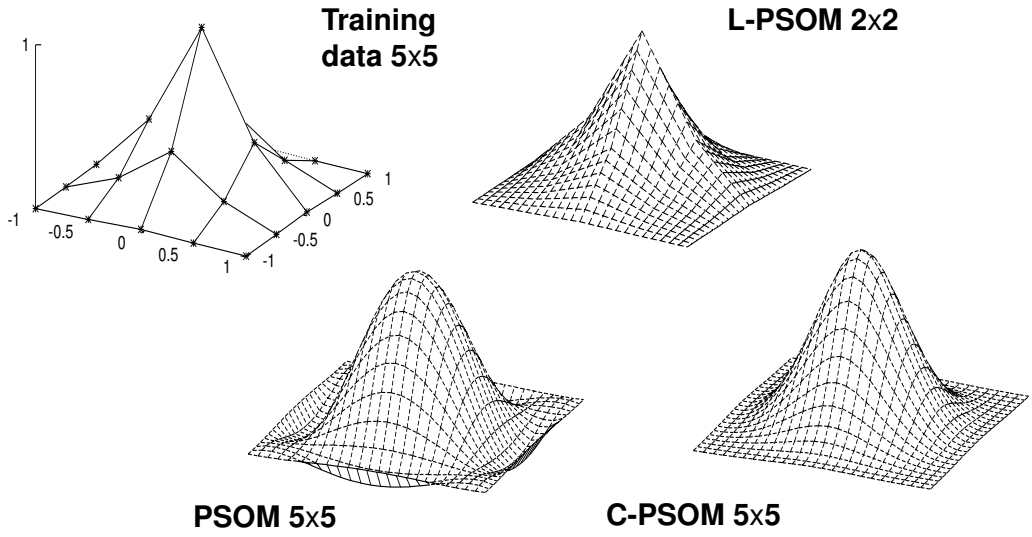


Figure 6.7: *a–d;* PSOM manifolds with a $5\times$ 5 training set. *(a)* 5×5 training points are equidistantly sampled for all PSOMs; *(b)* shows the resulting mapping of the local PSOM with sub-grid size 2×2. *(c)* There are little overshoots in the marginal mapping areas of the equidistant spaced PSOM *(i)* compared to *(d)* the mapping of the Chebyshev-spaced PSOM *(ii)* which is for $n = 5$ already visually identical to the goal map.

6.5.2 LLM Based Networks

The following illustrations compare the approximation performance of discrete Local Linear Maps (LLMs) for the same bell-shaped test function (note the different scaling).

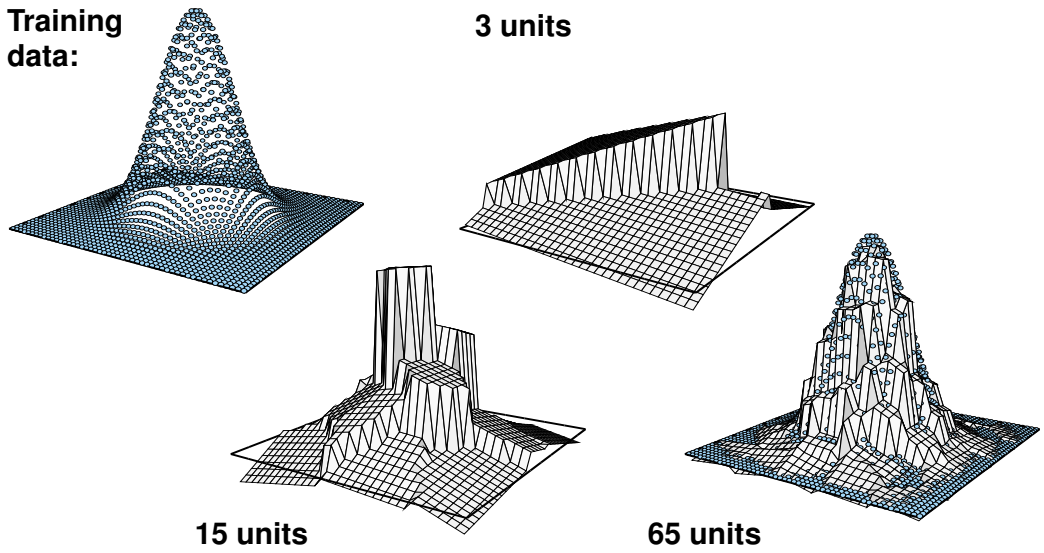


Figure 6.8: Gaussian Bell approximated by LLM units (from Fritzke 1995). A factor of 100 more training points were used - compared to the PSOM in the previous Fig. 6.7.

Fig. 6.8 shows the approximation result if using 3, 15, and 65 units. Since each unit is equipped with a parameterized local linear mapping (hyper-plane), comprising 2+3 (input+output) adaptable variables, the approximations effectively involves 15, 75, and 325 parameters respectively. For comparison, each PSOM node displayed in Fig. 6.7 uses only 2+1 non-constant parameters times 25 nodes (total 75), which makes it comparable to the LLM network with "15 units".

The pictures (Fig. 6.8) were generated by and reprinted with permission from Fritzke (1995). The units were allocated and distributed in the input space using a additive "Growing Neural Gas" algorithm (see also Sec. 3.6). Advantageously, this algorithm does not impose any constraints on the topology of the training data.

Since this algorithm does not need any topological information, it can-

not use any topological information. The differences are significant:

- the required training data set for the LLM network is much larger, here 2500 points versus 25 data for the PSOM network (factor 100); the PSOM can employ the self-organizing adaption rule Eq. 3.10 on a much smaller data set (25), or it can be instantly constructed if the sampling structure is known apriori (*rapid-learning*);
- the obtained *degree of continuity* and accuracy compares very favorably (Fig. 6.8 and Fig. 6.7). The PSOM approach does not need any extra heuristics to overcome the discrete nature of the local expert structure, which is visible in the discontinuities at the cell borders in Fig. 6.8.

6.6 RLC-Circuit Example

In this section a four-dimensional mapping example is presented. The purpose of this exercise is to see *(i)* to what extent the various $m = 4$ dimensional PSOM networks approximate the underlying functions; *(ii)* how this compares to the work of Friedman (1991) investigating the same parameter regime; *(iii)* to yield insight into the variable relationships.

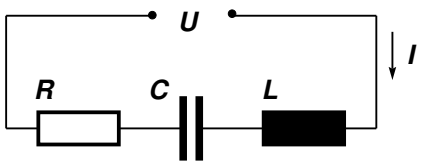


Figure 6.9: Schematic diagram of the alternating current series RLC - circuit.

Fig. 6.9 displays a schematic diagram of a simple series circuit involving a resistor R, inductor L and capacitor C. The circuit is driven by the sinusoidal generator placing an alternating voltage U

$$U = U_0 \sin 2\pi f t.$$

The resulting current I that flows through the circuit is also a sinusoidal with the same frequency,

$$I = \frac{U_0}{Z} \sin(2\pi f t - \Phi).$$

Its amplitude is governed by the impedance Z of the circuit and there is a phase shift Φ. Both depend on the frequency f and the components in the circuit:

$$Z = Z(R, L, C, f) = \sqrt{R^2 + \left(2\pi f L - \frac{1}{2\pi f C}\right)^2}. \quad (6.4)$$

$$\Phi = \Phi(R, L, C, f) = \text{atan}\frac{2\pi f L - \frac{1}{2\pi f C}}{R}. \quad (6.5)$$

Following Friedman (1991), we varied the variables in the range:

$$\begin{aligned} 0 \leq & \quad R \quad \leq 100\ [\Omega], \\ 0 \leq & \quad L \quad \leq 1\ [\text{H}], \\ 1 \leq & \quad C \quad \leq 11\ [\mu\text{F}], \\ 20 \leq & \quad f \quad \leq 280[\ \text{Hz}], \end{aligned}$$

which results in a impedance range $Z \in [50\Omega, 8000\Omega]$ and the phase lag Φ between $\pm 90^\circ$.

The PSOM training sets are generated by active sampling (here computing) the impedance Z and Φ for combination of one out of n resistors values R, one (of n) capacitor values C, one (of n) inductor values L, at n different frequencies f. As the embedding space we used the $d = 6$ dimensional space X spanned by the variables $\mathbf{x} = (R, L, C, f, Z, \Phi)$.

Type	n	Z-RMS	Φ-RMS	Z-NRMS	Φ-NRMS
C-PSOM	3	243Ω	31°	0.51	0.43
C-PSOM	4	130Ω	25°	0.24	0.34
L-PSOM	3 of 5	61Ω	25°	0.11	0.34
L-PSOM	3 of 7	25Ω	19°	0.046	0.25

Table 6.1: Results of various PSOM architectures for the $m = 4$ dimensional prediction task for the RLC circuit impedance.

Table Tab. 6.1 shows the root mean square (RMS and NRMS) results of several PSOM experiments when using Chebyshev spaced sampling and nodes placement. Within the same parameter regime Friedman (1991) reported results achieved with the MARS algorithm. He performs a hyper-rectangular partitioning of the task variable space and the fit of uni- and

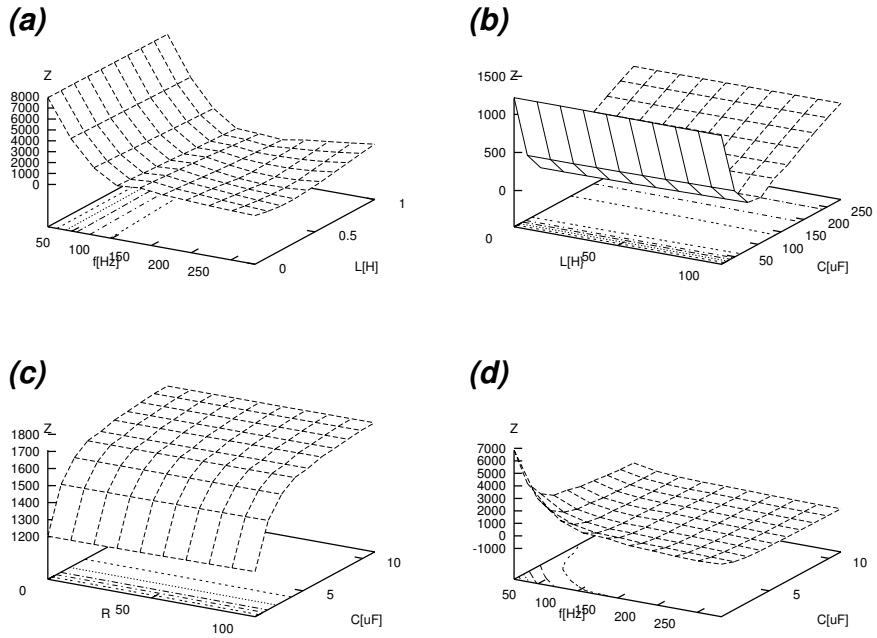


Figure 6.10: Isometric surface plots of 2 D cuts of the RLC-circuit mapping task of one single PSOM (contour lines projected on the base). All visualize the same 4 D continuous PSOM mapping manifold, here in the $d = 6$ embedding space X.

bi-variate splines within these regions. His best results for 400 data points were reported as a NRMS of 0.069 for the impedance value Z.

In contrast the PSOM approach constructs (here) a 4-dimensional parameterized manifold throughout. Fig. 6.10 visualizes this underlying mapping manifold in 3 D isometric views of various 2 D slices. Shown are $Z_{0\,\Omega,\ 1\,\mu F}(f, L)$, $Z_{0.5\,H,\ 6\,\mu F}(L, C)$, $Z_{1\,H,\ 280\,Hz}(R, C)$, and $Z_{0\,\Omega,\ 0\,H}(f, C)$. All views are obtained from the same PSOM, here a $n = 5$, $n' = 3$ L-PSOM, in the following also denoted (3-of-5)^4 L-PSOM.

The mapping accuracy depends crucially on the faithfulness of the parametric model – and the fit to the given mapping task. Fig. 6.11 depicts one situation in different views: Drawn are the change in impedance and phase lag with L and C, given R and f, and three PSOM networks (here left: $Z_{60\,\Omega,\ 186\,Hz}(L, C)$, right: $\Phi_{60\,\Omega,\ 186\,Hz}(L, C)$. Note, the reference vectors do not lie within the visualized surfaces, instead they lay "below" and "above" in the 4-dimensional mapping manifold, embedded in the 6 D space X).

The first row depicts the target mapping situation at the RLC-circuit, the others present a 3^4 PSOM, (3-of-5)^4 L-PSOM, and 5^4 PSOM (3^4 is the short form notation of 3× 3× 3× 3 nodes etc.). The impedance Z plot is marked by a curved valley, which is insufficiently picked up by the 3^4 case, but well approximated by the latter two PSOM types. The phase diagram in the right column shows a plateau with a steep slope at the left and front side. Again, the 3^6 type approximates a too smooth curve, because the network does not represent more detailed information. The PSOM with five nodes per axis exhibits at the right side little "overshots". Here, the contrast to the Local-PSOM is again apparent: the L-PSOM represents the flat plateau without long-range interferences of the "step".

6.7 Summary

We proposed the *multi-start* algorithm to face the problem of obtaining several solutions to the given mapping task. The concept of cost function modulation was introduced. It allows to dynamically add optimization goals which are of lower priority and possibly conflict with the primary goal.

Furthermore, we presented two extensions to the PSOM algorithm which

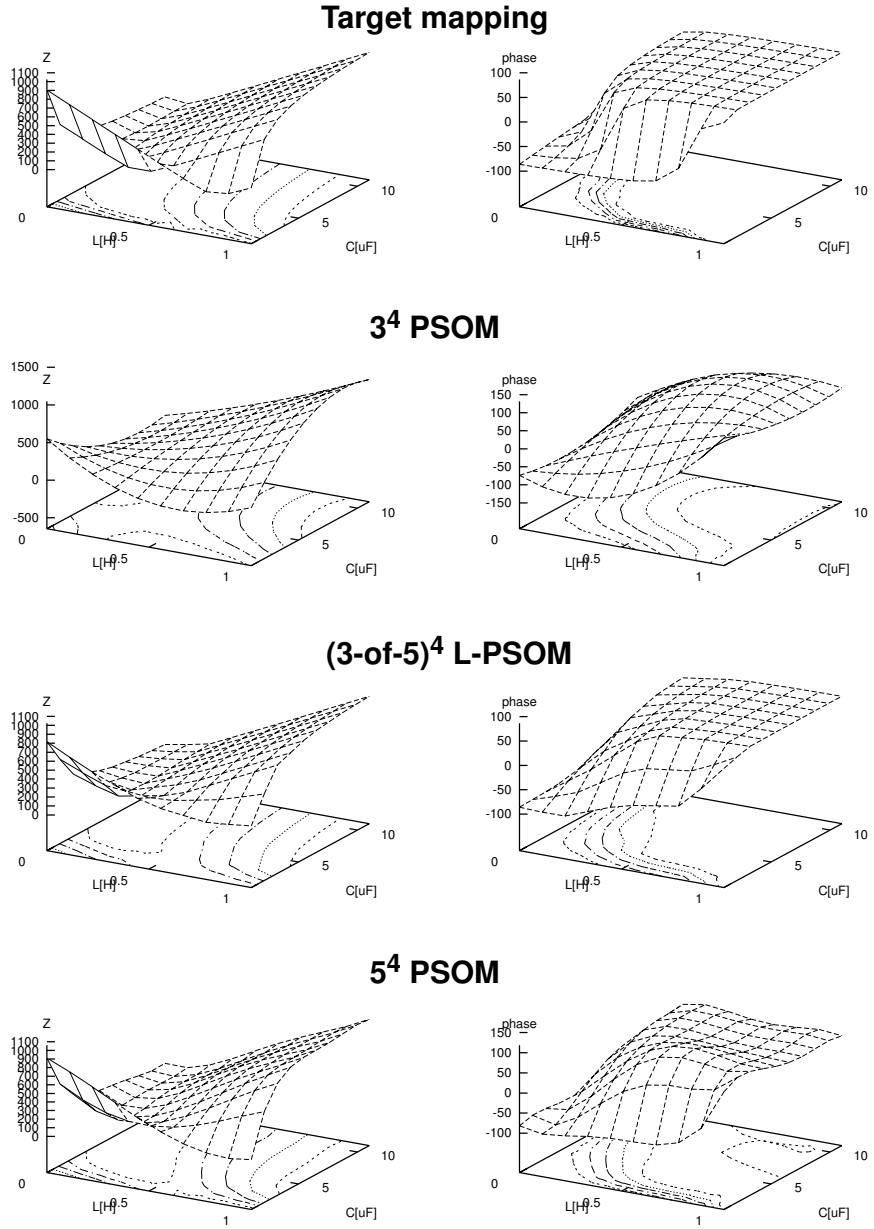


Figure 6.11: Comparison of three PSOM networks and the target mapping for one particularly interesting 2 D cut at a given R and f, drawn as (left column) $Z(L, C)$ and (right column) $\Phi(L, C)$ surface grids with contour plot projections on the base. See text.

aim at improving the mapping accuracy and the computational efficiency with larger training sets.

The proposed "Local-PSOM" algorithm constructs the constant sized PSOM on a dynamically determined sub-grid and keeps the computational effort constant when the number of training points increases. Our results suggest an excellent cost–benefit relation in cases with more than four nodes per axes.

An alternative to improve the mapping accuracy is the use of the "Chebyshev spaced PSOM" exploiting the superior approximation capabilities of the Chebyshev polynomials for the design of the internal basis functions. This imposes no extra effort but offers a significant precision advantage when using four or more nodes per axes.

Chapter 7

Application Examples in the Vision Domain

The PSOM algorithm has been explained in the previous chapters. In this chapter a number of examples are presented which expose its applicability in the vision domain. Vision is a sensory information source and plays an increasingly important role as perception mechanism for robotics.

The parameterized associative map and its particular completion mechanism serves here for a number of interesting application possibilities. The first example is concerned with the completion of an image feature set found here in a 2 D image, invariant to translation and rotation of the image. This idea can be generalized to a set of "virtual sensors". A redundant set of sensory information can be fused in order to improve recognition confidence and measurement precision. Here the PSOM offers a natural way of performing the fusion process in a flexible manner. As shown, this can be useful for further tasks, e.g. for inter-sensor cooperation, and identifying the the object's 3 D spatial rotations and position. Furthermore, we present also a more low-level vision application. By employing specialized feature filters, the PSOM can serve for identification of landmarks in gray-scale images, here shown for fingertips.

7.1 2 D Image Completion

First we want to consider here a planar example. The task is to complete a partial set of image feature locations, where the image can be translated

and rotated freely. The goal is to determine the proper shift and twist angle parameters when at least two image points are seen. Furthermore we desire to predict the locations of the hidden – maybe occluded or concealed – features. For example, this can be helpful to activate and direct specialized (possibly expensive) perception schema upon the predicted region.

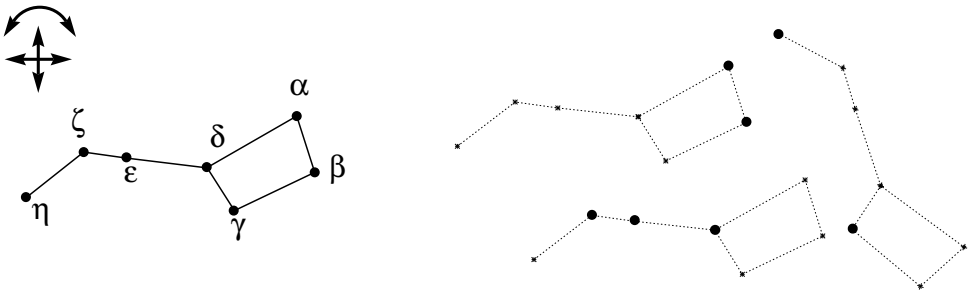


Figure 7.1: The star constellation the "Big Dipper" with its seven prominent stars $\alpha, \beta, \gamma, \delta, \epsilon, \zeta,$ and η. *(Left):* The training example consists of the image position of a seven stars $\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta$ in a particular viewing situation. *(Right):* Three examples of completed constellations after seeing only two stars in translated and rotated position. The PSOM output are the image location of the missing stars and desired set of viewing parameters (shift and rotation angle.)

Fig. 7.1 depicts the example. It shows the positions of the seven prominent stars in the constellation *Ursa Major* to form the asterisk called the "Big Dipper". The positions (x, y) in the image of these seven stars $(\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta)$ are taken from e.g. a large photograph. Together with the center-of-gravity position x_c, y_c and the rotation angle ϕ of the major axis (β-ζ) they span the embedding space X with the variables $\mathbf{x} = \{x_c, y_c, \phi,\ x_\alpha, y_\alpha,\ x_\beta, y_\beta, \ldots, x_\eta, y_\eta\}$.

As soon as two stars are visible, the PSOM network can predict the location of the missing stars and outputs the current viewing parameters – here shift and rotation (x_c, y_c, ϕ). Additionally other correlated parameters of interest can be trained, e.g. azimuth, elevation angle, or time values.

While two points are principally enough to fully recover the solution in this problem any realistic reconstruction task inevitably is faced with noise. Here the fusion of more image features is therefore an important problem, which can be elegantly solved by the associative completion mechanism of the PSOM.

7.2 Sensor Fusion and 3 D Object Pose Identification

Sensor fusion overcomes the limitation of individual sensor values for a particular task. When one kind of sensor cannot provide all the necessary information, a complementary observation from another sensory subsystem may fill the gap. Multiple sensors can be combined to improve measurement accuracy or confidence in recognition (see e.g. Baader 1995; Murphy 1995). The concept of a sensor system can be generalized to a "virtual sensor" – an abstract sensor module responsible for extracting certain feature informations from one or several real sensors.

In this section we suggest the application of a PSOM to naturally solve the sensor fusion problem. For the demonstration the previous planar (2 D) problem shall be extended.

Assume a 3 D object has a set of salient features which are observed by one or several sensory systems. Each relevant feature is detected by a "virtual sensor". Depending on the object pose, relative to the observing system, the sensory values change, and only a certain sub-set of features may be successfully detected.

When employing a PSOM, its associative completion capabilities can solve a number of tasks:

- knowing the object pose, predict the sensor response;
- knowing a sub-set of sensor values, reconstruct the object pose, and
- complete further information of interest (e.g. in the context of a manipulation task pose related grasp preshape and approach path informations);
- generate hypotheses for further perception "schemata", i.e. predict not-yet-concluded sensor values for "guidance" of further virtual sensors.

7.2.1 Reconstruct the Object Orientation and Depth

Here we want to extent the previous planar object reconstruction example to the three-dimensional world, which gives in total to 6 degrees of

freedom (3+3 DOF translation + rotation). We split the problem in two sub-tasks: *(i)* in locating and tracking[1] the object center (which is a 2 DOF problem), and *(ii)* finding the orientation together with the depth of the object seen. This factorization is advantageous in order to reduce the number of training examples needed.

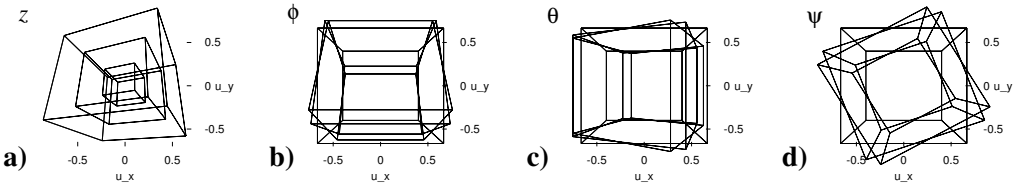


Figure 7.2: The ϕ, θ, ψ, z system. *(a)* The cubical test object seen by the camera when rotated and shifted in several depths z (ϕ=10° , θ=20° , ψ=30° , z=2...6L, cube size L.) *(b–d)* 0° , 20° , and 30° rotations in the roll ϕ, pitch θ, and yaw ψ system. (The transformations are applied from right to left ψ, θ, ϕ, z.)

Here we demonstrate a solution of the second, the part involving four independent variables (DOFs). Eight *virtual sensors* detect the corners of a test cube, seen in a perspective camera view of this object. Fig. 7.2 illustrates the parameterization of the object pose in the depths z and the three unit rotations in the roll ϕ, pitch θ, and yaw ψ angle system (see e.g. Fu et al. 1987). The embedding space X is spanned by the variables $\mathbf{x} = (\phi, \theta, \psi, z, \ \vec{u}_{P1}, \vec{u}_{P2}, \cdots, \vec{u}_{P8}, \ \cdots)$ where $\vec{u}_{Pi}$ is the signal of sensor i, here the image coordinate pair of point P_i.

Tab. 7.1 presents the achieved accuracy for recovering the object pose for a variety of experimental set-up parameters. Various ranges and training vector numbers demonstrate how the precision to which identification is achieved depends on the range of the involved free variables and the number of data vectors which are invested for training.

With the same trained PSOM, the system can predict locations of occluded object parts when a sufficient number of points has already been found. These hypotheses could be fed back to the perceptual processing stages, e.g. to "look closer" on the predicted sub-regions of the image

[1] Walter and Ritter (1996e) describes the experiment of Cartesian tracking of a 3 D free movable sphere. The ball is presented by a human and real-time tracked in 3 D by the Puma robot using the end-effector based vision system (camera-in-hand configuration).

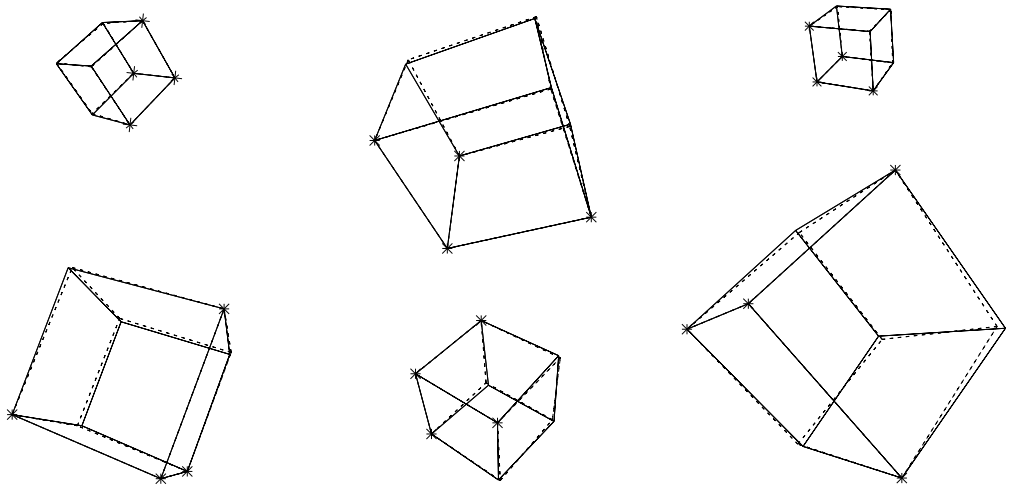


Figure 7.3: Six Reconstruction Examples. *Dotted* lines indicate the test cube as seen by a camera. *Asterisks* mark the positions of the four corner points used as inputs for reconstruction of the object pose by a PSOM. The *full* lines indicate the reconstructed and completed object.

(*inter-sensor coordination*). The lower part of the table shows the results when only four points are found and the missing locations are predicted. Only the appropriate p_k in the projection matrix $\mathbf{P}$ (Eq. 4.7) are set to one, in order to find the best-matching solution in the attractor manifold. For several example situations, Fig. 7.3 depicts the completed cubical object on the basis of the found four points (asterisk marked = input to the PSOM), and for comparative reasons the true target cube with dashed lines (case $3{\times}3{\times}3{\times}3$PSOM with ranges 150° ,$2L$). In Sec. 9.3.1 we will return to this problem.

7.2.2 Noise Rejection by Sensor Fusion

The PSOM best-match search mechanism (Eq. 4.4) performs an automatic minimization in the least-square sense. Therefore, the PSOM offers a very natural way of *fusing redundant sensory information* in order to improve the reconstruction accuracy in case of input noise.

In order to investigate this capability we added Gaussian noise to the virtual sensor values and determined the resulting average orientation de-

PSOM	ϕ,θ,ψ range	z/L	$\langle\Delta\phi\rangle$	$\langle\Delta\theta\rangle$	$\langle\Delta\psi\rangle$	$\langle\Delta z\rangle$	$\langle\lvert\Delta\vec{n}\rvert\rangle$	$\langle\lvert\Delta\vec{o}\rvert\rangle$	$\langle\vec{n}\cdot\vec{o}\rangle$	$\langle\lvert\Delta\vec{u}_{P5}\rvert\rangle$ $[L]$	$\langle\lvert\Delta\vec{u}_{P6}\rvert\rangle$ $[L]$
4 and 8 points are input											
3×3×3×2	150°	2	2.6°	3.1°	2.9°	0.11	0.039	0.046	0.0084	given	given
3×3×3×2	150°	2	2.7°	3.2°	2.8°	0.12	0.043	0.048	0.0084	0.010	0.0081
Learn only rotational part											
3×3×3	150°		2.6°	3.0°	2.5°		0.046	0.048	0.0074	0.018	0.012
4×4×4	150°		0.63°	1.2°	0.93°		0.021	0.019	0.0027	0.013	0.0063
5×5×5	150°		0.12°	0.12°	0.094°		0.0034	0.0027	0.00042	0.0017	0.00089
Various rotational ranges											
3×3×3×2	90°	1	0.64°	0.56°	0.53°	0.034	0.0085	0.0082	0.00082	0.0036	0.0021
3×3×3×2	120°	1	1.5°	1.5°	1.4°	0.037	0.021	0.021	0.0032	0.0079	0.0049
3×3×3×2	150°	1	2.7°	3.2°	2.8°	0.077	0.044	0.048	0.0084	0.013	0.010
3×3×3×2	180°	1	6.5°	5.4°	7.0°	0.19	0.079	0.098	0.014	0.019	0.016
Various training set sizes											
3×3×3×2	150°	2	2.7°	3.2°	2.8°	0.12	0.043	0.048	0.0084	0.010	0.0081
3×3×3×3	150°	2	2.6°	3.2°	2.8°	0.11	0.043	0.048	0.0084	0.0097	0.0077
4×4×4×2	150°	2	0.49°	0.97°	0.73°	0.12	0.018	0.016	0.0030	0.0089	0.0059
4×4×4×3	150°	2	0.52°	0.98°	0.71°	0.035	0.017	0.014	0.0026	0.0082	0.0053
5×5×5×3	150°	2	0.14°	0.13°	0.14°	0.024	0.0033	0.0030	0.00043	0.0018	0.0011
Shift depth range z											
3×3×3×3	150°	1..3	3.8°	3.4°	3.7°	0.12	0.061	0.064	0.0083	0.049	0.025
3×3×3×3	150°	2..4	2.6°	3.2°	2.8°	0.11	0.043	0.048	0.0084	0.0097	0.0077
3×3×3×3	150°	3..5	2.6°	3.2°	2.9°	0.15	0.042	0.047	0.0084	0.0050	0.0045
Various distance ranges											
3×3×3×3	150°	2	2.6°	3.2°	2.8°	0.11	0.043	0.048	0.0084	0.0097	0.0077
3×3×3×3	150°	4	2.6°	3.2°	2.8°	0.20	0.042	0.047	0.0084	0.0068	0.0059
3×3×3×3	150°	6	2.6°	3.2°	2.9°	0.36	0.043	0.048	0.0084	0.0057	0.0052
5×5×5×3	150°	6	0.65°	0.73°	0.93°	0.39	0.016	0.013	0.00047	0.0070	0.0051
4×4×4×4	150°	6	0.44°	0.43°	0.60°	0.14	0.0097	0.0083	0.00042	0.0043	0.0029

Table 7.1: Mean Euclidean deviation of the reconstructed pitch, roll, yaw angles ϕ,θ,ψ, the depth z, the column vectors $\vec{n},\vec{o}$ of the rotation matrix T, the scalar product of the vectors $\vec{n}\cdot\vec{o}$ (orthogonality check), and the predicted image position of the object locations $P5, P6$. The results are obtained for various experimental parameters in order to give some insight into their impact on the achievable reconstruction accuracy. The PSOM training set size is indicated in the first column, the ϕ,θ,ψ intervals are centered around $0°$, and depth z ranges from $z_{min} = 2L$, where L denotes the cube length (focal length of the lens is also taken as $\lambda = L$.) In the first row all corner locations are inputs. All remaining results are obtained using only four (non-coplanar) points as inputs.

viation (norm in ψ, θ, ϕ) as function of the noise level and the number of sensors contributing to the desired output.

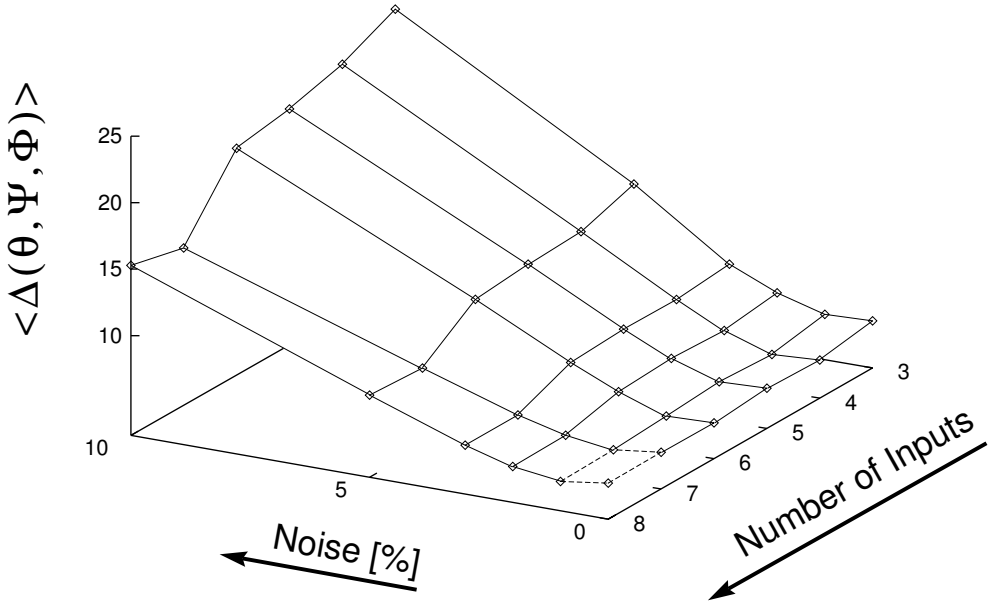


Figure 7.4: The reconstruction deviation versus the number of fused sensory inputs and the percentage of Gaussian noise added. By increasing the number of fused sensory inputs the performance of the reconstruction can be improved. The significance of this feature grows with the given noise level.

Fig. 7.4 exposes the results. Drawn is the mean norm of the orientation angle deviation for varying added noise level σ from 0 to 10 % of the average image size, and for 3,4,... and 8 fused sensory inputs, which were taken into account. We clearly find with higher noise levels there is a growing benefit from an increasing increased number of contributing sensors.

And as one expects from a sensor fusion process, the overall precision of the entire system is improved in the presence of noise. Remarkable is how naturally the PSOM associative completion mechanism allows to include available sensory information. Different feature sensors can also be relatively weighted according to their overall accuracy as well as their estimated confidence in the particular perceptual setting.

7.3 Low Level Vision Domain: a Finger Tip Location Finder

So far, we have been investigating PSOMs for learning tasks in the context of well pre-processed data representing clearly defined values and quantities. In the vision domain, those values are results of low level processing stages where one deals with extremely high-dimensional data. In many cases, it is doubtful to what extent smoothness assumptions are valid at all.

Still, there are many situations in which one would like to compute from an image some low-dimensional parameter vector, such as a set of parameters describing location, orientation or shape of an object, or properties of the ambient illumination etc. If the image conditions are suitably restricted, the input images may be samples that are represented as vectors in a very high dimensional vector space, but that are concentrated on a much lower dimensional sub-manifold, the dimensionality of which is given by the independently varying parameters of the image ensemble.

A frequently occurring task of this kind is to identify and mark a particular part of an object in an image, as we already met in the previous example for determination of the cube corners. For further example, in face recognition it is important to identify the locations of salient facial features, such as eyes or the tip of the nose. Another interesting task is to identify the location of the limb joints of humans for analysis of body gestures. In the following, we want to report from a third application domain, the identification of finger tip locations in images of human hands (Walter and Ritter 1996d). This would constitute a useful preprocessing step for inferring 3 D-hand postures from images, and could help to enhance the accuracy and robustness of other, more direct approaches to this task that are based on LLM-networks (Meyering and Ritter 1992).

For the results reported here, we used a restricted ensemble of hand postures. The main degree of freedom of a hand is its degree of "closure". Therefore, for the initial experiments we worked with an image set comprising grips in which all fingers are flexed by about the same amount, varying from fully flexed to fully extended. In addition, we consider rotation of the hand about its arm axis. These two basic degrees of freedom yield a two-dimensional image ensemble (i.e., for the dimension m of the map manifold we have $m = 2$). The objective is to construct a PSOM that

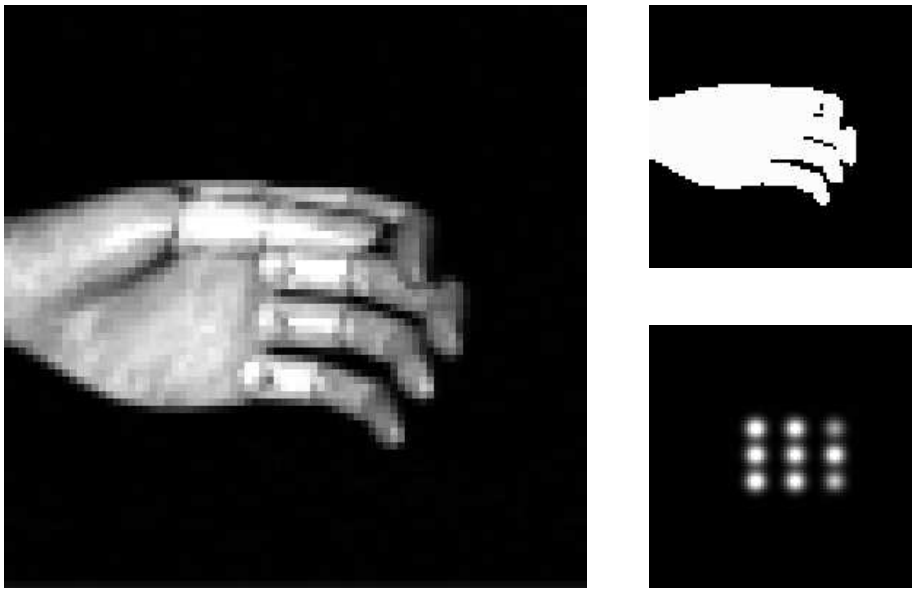


Figure 7.5: *Left,(a):* Typical input image. *Upper Right,(b):* after thresholding and binarization. *Lower Right,(c):* position of 3×3 array of Gaussian masks (the displayed width is the actual width reduced by a factor of four in order to better depict the position arrangement)

maps a monocular image from this ensemble to the 2 D-position of the index finger tip in the image.

In order to have reproducible conditions, the images were generated with the aid of an adjustable wooden hand replica in front of a black background (for the required segmentation to achieve such condition for more realistic backgrounds, see e.g. Kummert et al. 1993a; Kummert et al. 1993b). A typical image (80×80 pixel resolution) is shown in Fig. 7.5a. From the monochrome pixel image, we generated a 9-dimensional feature vector first by thresholding and binarizing the pixel values (threshold = 20, 8-bit intensity values), and then by computing as image features the scalar product of the resulting binarized images (shown in Fig. 7.5b) with a grid of 9 Gaussians at the vertices of a 3×3 lattice centered on the hand (Fig. 7.5c). The choice of this preprocessing method is partly heuristically motivated (the binarization makes the feature vector more insensitive to variations of the illumination), and partly based on good results achieved with a similar method in the context of the recognition of hand postures

(Kummert et al. 1993b).

To apply the PSOM-approach to this task requires a set of labeled training data (i.e., images with known 2 D-index finger tip coordinates) that result from sampling the parameter space of the continuous image ensemble on a 2 D-lattice. In the present case, we chose the subset of images obtained when viewing each of four discrete hand postures (fully closed, fully opened and two intermediate postures) from one of seven view directions (corresponding to rotations in $30°$-steps about the arm axis) spanning the full 180^0-range. This yields the very manageable number of 28 images in total, for which the location of the index finger tip was identified and marked by a human observer.

Ideally, the dependency of the x- and y-coordinates of the finger tip should be smooth functions of the resulting 9 image features. For real images, various sources of noise (surface inhomogeneities, small specular reflections, noise in the imaging system, limited accuracy in the labeling process) lead to considerable deviations from this expectation and make the corresponding interpolation task for the network much harder than it would be if the expectation of smoothness were fulfilled. Although the thresholding and the subsequent binarization help to reduce the influence of these effects, compared to computing the feature vector directly from the raw images, the resulting mapping still turns out to be very noisy. To give an impression of the degree of noise, Fig. 7.7 shows the dependence of horizontal (x-) finger tip location (plotted vertically) on two elements of the 9 D-feature vector (plotted in the horizontal $xy-$plane). The resulting mesh surface is a projection of the full 2 D-map-manifold that is embedded in the space X, which here is of dimensionality 11 (nine dimensional input features space X^{in}, and a two dimensional output space $X^{\mathrm{out}} = (x, y)$ for position.) As can be seen, the underlying "surface" does not appear very smooth and is disrupted by considerable "wrinkles".

To construct the PSOM, we used a subset 16 images of the image ensemble by keeping the images seen from the two view directions at the ends ($\pm 90°$) of the full orientation range, plus the eight pictures belonging to view directions of $\pm 30°$. For subsequent testing, we used the 12 images from the remaining three view directions of $0°$ and $\pm 60°$. I.e., both training and testing ensembles consisted of image views that were multiples of $60°$ apart, and the directions of the test images are midway between the directions of the training images.

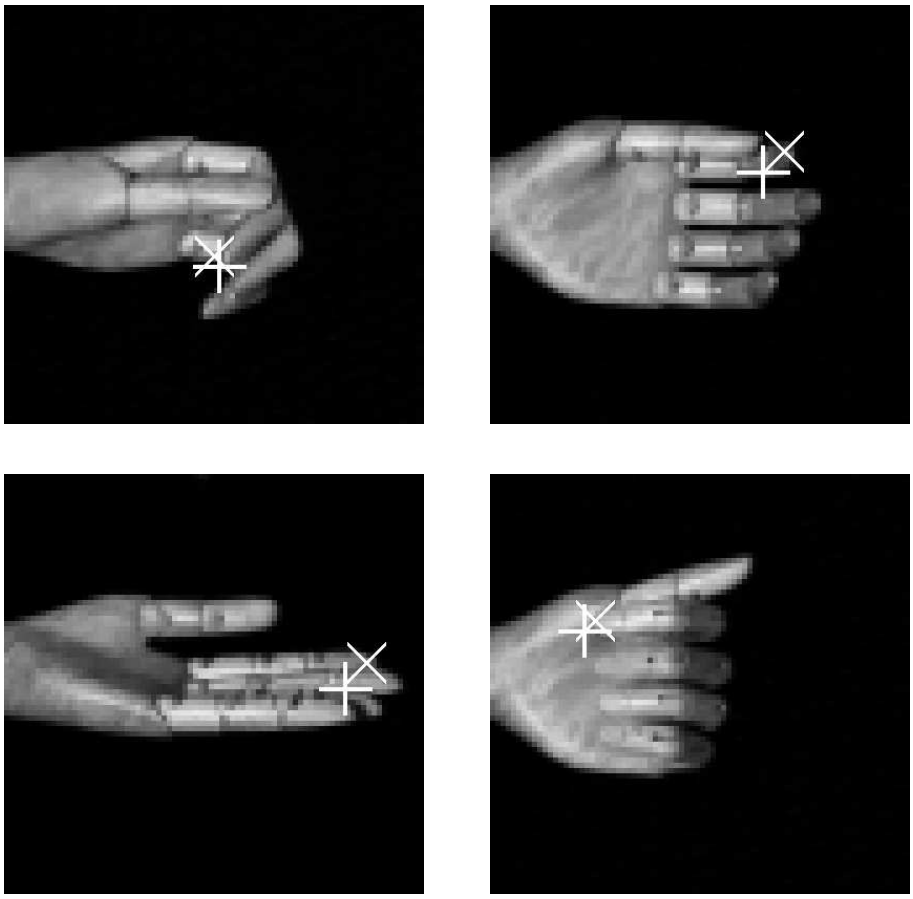


Figure 7.6: Some examples of hand images with correct (cross-mark) and predicted (plus-mark) finger tip positions. Upper left image shows average case, the remaining three pictures show the three worst cases in the test set. The NRMS positioning error for the marker point was 0.11 for horizontal, 0.23 for vertical position coordinate.

Even with the very small training set of only 16 images, the resulting PSOM achieved a NRMS-error of 0.11 for the x-coordinate, and of 0.23 for the y-coordinate of the finger tip position (corresponding to absolute RMS-errors of about 2.0 and 2.4 pixels in the 80×80 image, respectively). To give a visual impression of this accuracy, Fig. 7.6 shows the correct (cross mark) and the predicted (plus mark) finger tip positions for a typical average case (upper left image), together with the three worst cases in the test set (remaining images).

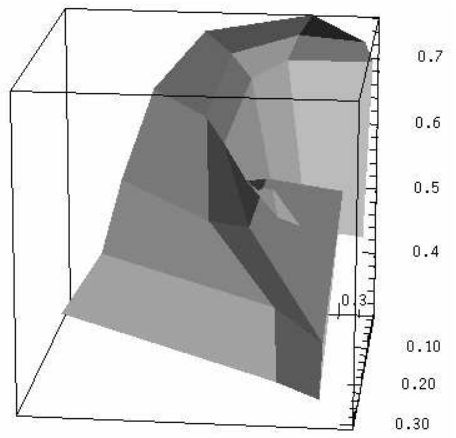


Figure 7.7: Dependence of vertical index finger position on two of the nine input features, illustrating the very limited degree of smoothness of the mapping from feature to position space.

This closes here the list of presented PSOM applications homing purely in the vision domain. In the next two chapters sensorimotor transformation will be presented, where vision will again play a role as sensory part.

Chapter 8

Application Examples in the Robotics Domain

As pointed out before in the introduction, in the robotic domain the availability of sensorimotor transformations are a crucial issue. In particular, the *kinematic relations* are of fundamental character. They usually describe the relationship between joint, and actuator coordinates, and the position in one, or several particular Cartesian reference frames.

Furthermore, the effort spent to obtain and adapt these mappings plays an important role. Several thousand training steps, as required by many former learning schemes, do impair the practical usage of learning methods in the domain of robotics. Here the wear-and-tear, but especially the needed time to acquire the training data must be taken into account.

Here, the PSOM algorithm appears as a very suitable learning approach, which requires only a small number of training data in order to achieve a very high accuracy in continuous, smooth, and high-dimensional mappings.

8.1 Robot Finger Kinematics

In section 2.2 we described the TUM robot hand, which is built of several identical finger modules. To employ this (or a similar dextrous) robot hand for manipulation tasks requires to solve the forward and inverse kinematics problem for the hand finger. The TUM mechanical design allows roughly the mobility of the human index finger. Here, a cardanic base joint

(2 DOF) offers sidewards gyring of $\pm 15^\circ$ and full adduction with two additional coupled joints (one further DOF). Fig. 8.1 illustrates the workspace with a stroboscopic image.

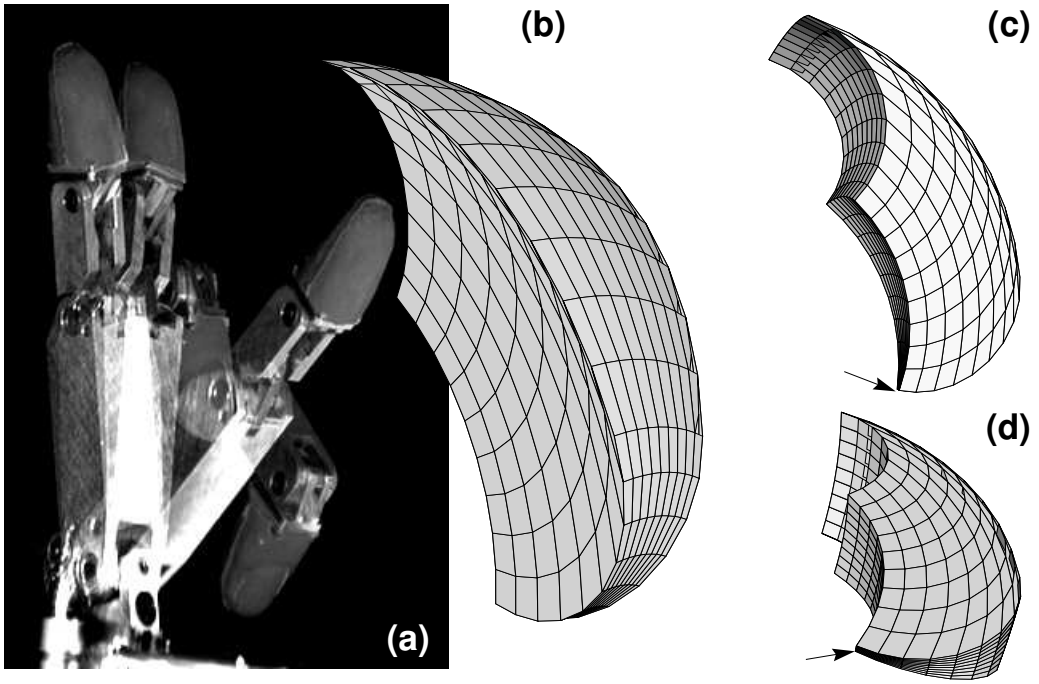


Figure 8.1: a–d: *(a)* stroboscopic image of one finger in a sequence of extreme joint positions.
(b–d) Several perspectives of the workspace envelope $\vec{r}$, tracing out a cubical $10\times10\times10$ grid in the joint space $\vec{\theta}$. The arrow marks the fully adducted position, where one edge contracts to a tiny line.

For the kinematics in the case of our finger, there are several coordinate systems of interest, e.g. the joint angles, the cylinder piston positions, one or more finger tip coordinates, as well as further configuration dependent quantities, such as the Jacobian matrices for force / moment transformations. All of these quantities can be simultaneously treated in one single common PSOM; here we demonstrate only the most difficult part, the classical inverse kinematics. When moving the three joints on a cubical $10\times10\times10$ grid within their maximal configuration space, the fingertip (or more precisely the mount point) will trace out the "banana" shaped grid displayed in Fig. 8.1 (confirm the workspace with your finger!) Obviously,

the underlying transformation is highly non-linear and exhibits a point-singularity in the vicinity of the "banana tip". Since an analytical solution to the inverse kinematic problem was not derived yet, this problem was a particular challenging task for the PSOM approach (Walter and Ritter 1995).

We studied several PSOM architectures with n×n×n nine dimensional data tuples $(\vec{\theta}, \vec{c}, \vec{r})$, where $\vec{\theta}$ denotes the joint angles, $\vec{c}$ the piston displacement and $\vec{r}$ the Cartesian finger point position, all equidistantly sampled in $\vec{\theta}$. Fig. 8.2a–b depicts a $\vec{\theta}$ and an $\vec{r}$ projection of the smallest training set, $n = 3$.

To visualize the inverse kinematics ability, we require the PSOM to back-transform a set of workspace points of known arrangement (by specifying $\vec{\theta}$ as input sub-space). In particular, the workspace filling "banana" set of Fig. 8.1 should yield a rectangular grid of $\vec{\theta}$. Fig. 8.2c–e displays the actual result. The distortions look much more significant in the joint angle space *(a)*, and the piston stoke space *(b)*, than in the corresponding world coordinate result $\vec{r}'$ *(b)* after back-transforming the PSOM angle output. The reason is the peculiar structure; e.g. in areas close to the tip a certain angle error corresponds to a smaller Cartesian deviation than in other areas.

When measuring the mean Cartesian deviation we get an already satisfying result of 1.6 mm or **1.0 %** of the maximum workspace length of 160 mm. In view of the extremely small training set displayed in Fig. 8.2a–b this appears to be a quite remarkable result.

Nevertheless, the result can be further improved by supplying more training points as shown in the asterisk marked curve in Fig. 8.3. The effective inverse kinematic accuracy is plotted versus the number of training nodes per axes, using a set of 500 randomly (in $\vec{\theta}$ uniformly) sampled positions.

For comparison we employed the "plain-vanilla" MLP with one and two hidden layers (units with tanh(·) squashing function) and linear units in the output layer. The encoding was similar to the PSOM case: the plain θ angles as inputs augmented by a constant bias of one (Fig. 3.1). We found that this class of problems appears to be very hard for the standard MLP network, at least without more sophisticated learning rules than the standard back-propagation gradient descent. Even for larger training set sizes, we did not succeed in training them to a performance comparable

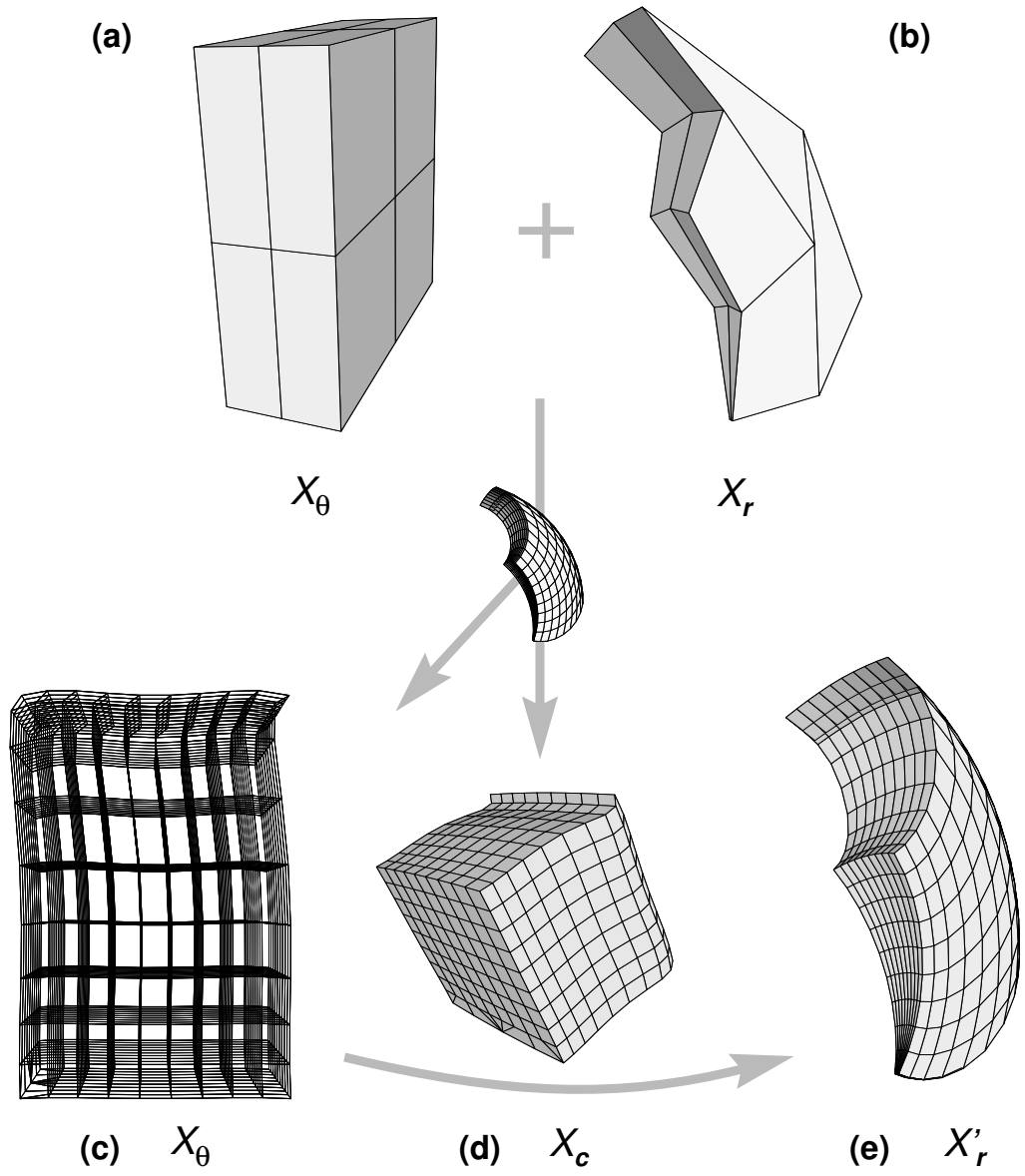


Figure 8.2: a–b and c–e; Training data set of 27 nine-dimensional points in X for the 3×3×3 PSOM, shown as perspective surface projections of the *(a)* joint angle $\vec{\theta}$ and *(b)* the corresponding Cartesian sub space. Following the lines connecting the training samples allows one to verify that the "banana" really possesses a cubical topology. *(c–e)* Inverse kinematic result using the grid test set displayed in Fig. 8.1. *(c)* projection of the joint angle space $\vec{\theta}$ (transparent); *(d)* the stroke position space $\vec{c}$; *(e)* the Cartesian space $\vec{r}\,'$, after back-transformation.

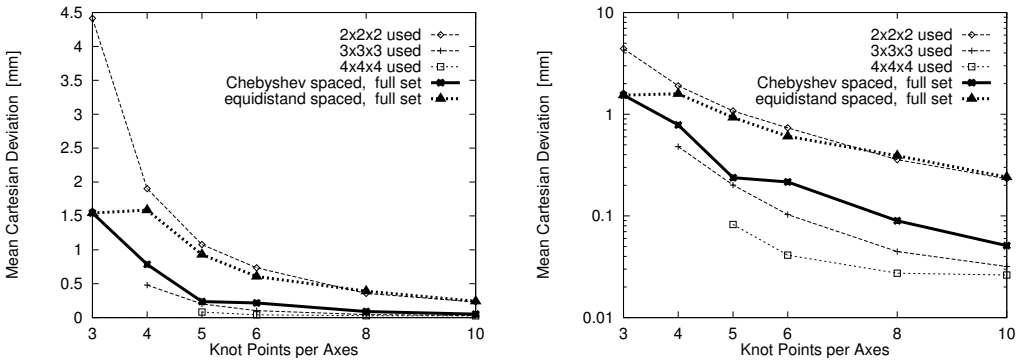


Figure 8.3: a–b: Mean Cartesian inverse kinematics error (in mm) of the presented PSOM types versus number of training nodes per axes (using a test set of 500 randomly chosen positions; *(a)* linear and *(b)* log plot). Note, the result of Fig. 8.2c–e corresponds to the smallest training set $n = 3$. The maximum workspace length is 160 mm.

to the PSOM network. Table 8.1 shows the result of two of the best MLP-networks compared to the PSOM.

Network	ϵ_i	ϵ_f	$n = 3$	$n = 4$	$n = 5$
MLP 3–50–3	0.02	0.004	0.72	0.57	0.54
MLP 3–100–3	0.01	0.002	0.86	0.64	0.51
PSOM			0.062	0.037	0.004

Table 8.1: Normalized root mean square error (NRMS) of the inverse kinematic mapping task $\vec{r} \mapsto \vec{\theta}$ computed as the resulting Cartesian deviation from the goal position. For a training set of n×n×n points, obtained by the two best performing standard MLP networks (out of 12 different architectures, with various (linear decreasing) step size parameter schedules $\epsilon = \epsilon_i \ldots \epsilon_f$) 100000 steepest gradient descent steps were performed for the MLP and one pass through the data set for PSOM network.

Why does the PSOM perform more that an order of magnitude better than the back-propagation algorithm? Fig. 8.4 shows the 27 training data pairs in the Cartesian input space $\vec{r}$. One can recognize some zig-zig clusters, but not much more. If neighboring nodes are connected by lines, it is easy to recognize the coarse "banana" shaped structure which was successfully generalized to the desired workspace grid (Fig. 8.2). The PSOM

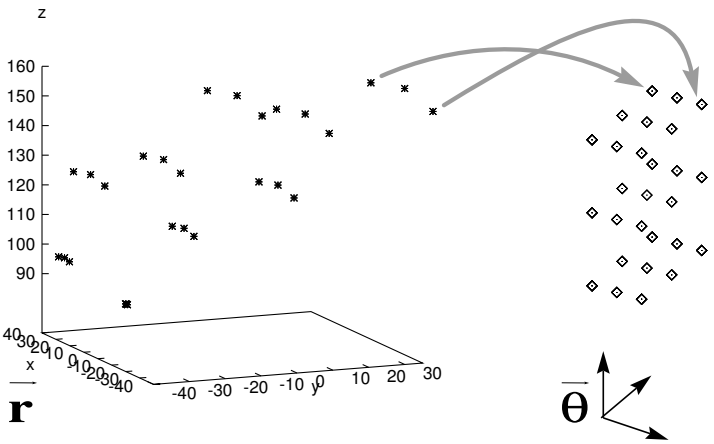


Figure 8.4: The 27 training data vectors for the Back-propagation networks: *(left)* in the input space $\vec{r}$ and *(right)* the corresponding target output values $\vec{\theta}$.

gets the same data-pairs as training vectors — but additionally, it obtains the assignment to the node location $\mathbf{a}$ in the $3\times3\times3$ node grid illustrated in Fig. 8.5.

As explained before in Sec. 5, specifying $\mathbf{a} \in \mathbf{A}$ *introduces topological order* between the training vectors $\mathbf{w_a}$. This allows the PSOM to advantageously draw *extra curvature information* from the data set — information, that is *not available* with other techniques, such as the MLP or the RBF network approach. The visual comparison of the two viewgraphs demonstrates the essential value of the added structural information.

8.2 A Higher Dimensional Mapping: The 6-DOF Inverse Puma Kinematics

To demonstrate the capabilities of the PSOM approach in a higher dimensional mapping domain, we apply the PSOM to construct an approximation to the kinematics of the Puma 560 robot arm with six degrees of freedom. As embedding space X we first use the 15-dimensional space X spanned by the variables

$$\mathbf{x} = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6,\ r_x, r_y, r_z,\ a_x, a_y, a_z,\ n_x, n_y, n_z).$$

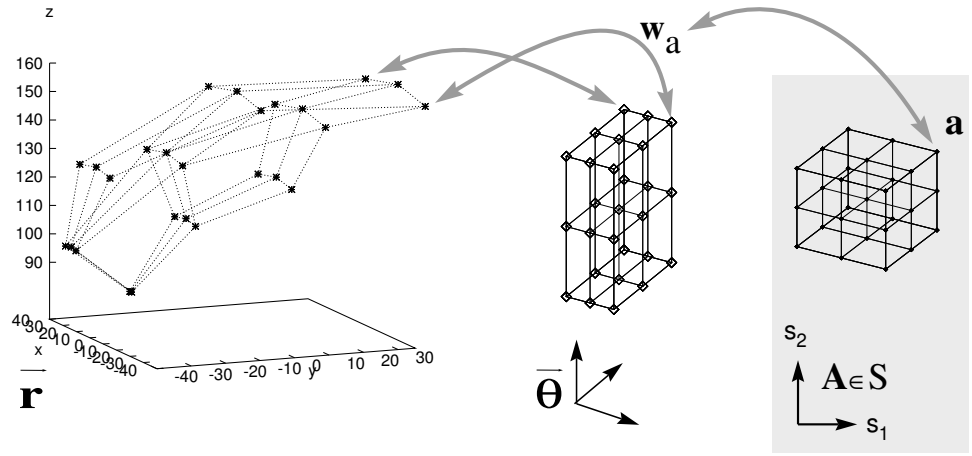


Figure 8.5: The same 27 training data vectors (cmp. Fig. 8.4) for the bi-directional PSOM mapping: *(left)* in the Cartesian space $\vec{r}$, *(middle)* the corresponding joint angle space $\vec{\theta}$. *(Right:)* The corresponding node locations $\mathbf{a} \in \mathbf{A}$ in the parameter manifold S. Neighboring nodes are connected by lines, which reveals now the "banana" structure on the left.

Here, $\theta_1 \ldots \theta$ denote the joint angles, $\vec{r}$ is the Cartesian position of the end effector of length l_z in world coordinates. $\vec{a}$ and $\vec{n}$ denote the normalized approach vector and the vector normal to the hand plane. The last nine components vectors are part of the homogeneous coordinate transformation matrix

$$\mathbf{T} = \begin{bmatrix} n_x & o_x & a_x & r_x \\ n_y & o_y & a_y & r_y \\ n_z & o_z & a_z & r_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{8.1}$$

(The missing second matrix column $\vec{o}$ is the cross product of the normalized orientation vectors $\vec{a}$ and $\vec{n}$ and therefore bears no further information, see Fig. 8.6 and e.g. (Fu et al. 1987; Paul 1981).)

In this space, we must construct the $m = 6$ dimensional embedding manifold M that represents the configuration manifold of the robot. With three nodes per axis direction we require $3^6 = 726$ reference vectors $\mathbf{w_a} \in X$. The distribution of these vectors might have been found with a SOM, however, for the present demonstration we generated the values for the $\mathbf{w_a}$ by augmenting 729 joint angle vectors on a rectangular $3\times$ $3\times$ $3\times$ $3\times$ $3\times$ 3 grid in joint angle space $\vec{\theta}$ with the missing $r_x, r_y, r_z, a_x, a_y, a_z$, -

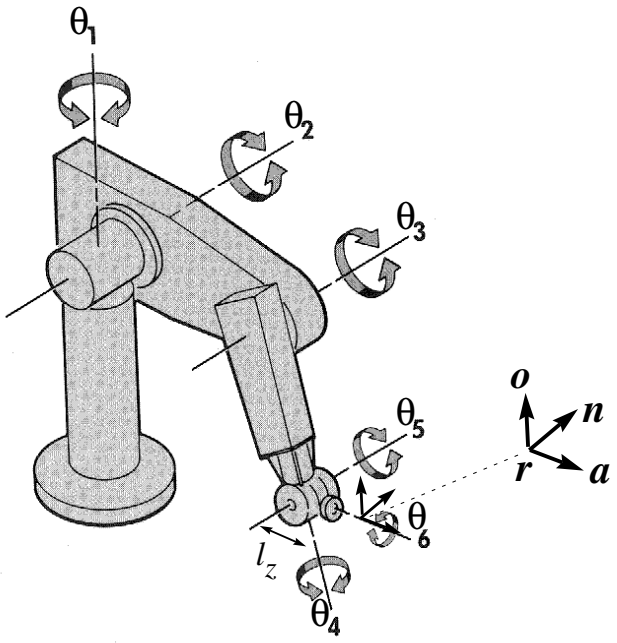


Figure 8.6: The 15 components of the training data vectors for the PSOM networks: The six Puma axes and the position $\vec{r}$ and orientation vectors $\vec{n}$, $\vec{o}$, and $\vec{a}$ of the tool frame.

n_x, n_y, n_z–components, using the forward kinematics transform equations (Paul 1981) ($\theta_1 \in$[-135° ,-45°], $\theta_2 \in$[-180° ,-100°], $\theta_3 \in$[-35° ,55°], $\theta_4 \in$[-45° ,45°], $\theta_5 \in$[-90° ,0°], $\theta_6 \in$[45° ,135°], and tool length l_z={0,200} mm in z direction of the T_6 frame, see Fig. 8.6.

Similar to the previous example, we then test the PSOM based on the 3^6 points in the inverse mapping direction. To this end, we specify Cartesian goal positions $\vec{r}$ and orientation values $\vec{a}, \vec{n}$ at 200 randomly chosen intermediate test points and use the PSOM to obtain the missing joint angles $\vec{\theta}$.

Thus, nine dimensions of the embedding space X are selected as input sub-space. The three components $\{r_x, r_y, r_z\}$ are given in length units ([mm] or [m]) and span intervals of range {1.5, 1.2, 1.6} meters for the given training set, in contrast to the other six dimensionless orientation components, which vary in the interval [-1,+1]. Here the question arises what to do with these incommensurable components of different unit and magnitude? The answer is to account for this in the distance metric $dist(\cdot)$. The best solution is to weight each component k in Eq. 4.7 reciprocally to the measurement variance

$$p_k = [\text{var}\,(w_k)]^{-1}\,. \tag{8.2}$$

If the number of measurements is small, as it is usual for small data sets,

PSOM 3^6		Cartesian position deviation $\Delta\vec{r}$		approach vector deviation $\Delta\vec{a}$		normal vector deviation $\Delta\vec{n}$	
l_z [mm]	Sampling	Mean	NRMS	Mean	NRMS	Mean	NRMS
0	bounded	19 mm	0.055	0.035	0.055	0.034	0.057
200	bounded	23 mm	0.053	0.035	0.055	0.034	0.057
0	Chebyshev	12 mm	0.033	0.022	0.035	0.020	0.035
200	Chebyshev	14 mm	0.034	0.022	0.035	0.021	0.035

Table 8.2: Full 6 DOF inverse kinematics accuracy using a 3×3×3×3×3×3 PSOM for a Puma robot with two different tool lengths l_z. The training set was sampled in a rectangular grid in $\theta_{1\ldots6}$, in each axis centered at the working range midpoint. The bordering samples were taken at the range borders (*bounded*), or according to the zeros of the *Chebyshev* polynomial T_3 (Eq. 6.3).

we may roughly approximate the variance by the following computational shortcut. In Eq. 8.2 the non-zero diagonal elements p_k of the projection matrix $\mathbf{P}$ are set according to the interval spanned by the set of reference vectors $\mathbf{w_a}$:

$$p_k = (w_k^{max} - w_k^{min})^{-2}. \tag{8.3}$$

With

$$w_k^{max} = \max_{\forall \mathbf{a} \in \mathbf{A}} w_{k,\mathbf{a}} \quad \text{and} \quad w_k^{min} = \min_{\forall \mathbf{a} \in \mathbf{A}} w_{k,\mathbf{a}} \tag{8.4}$$

the distance metric becomes invariant to a rescaling of any component of the embedding space X. This method can be generally recommended when input components are of uneven scale, but considered equally significant. As seen in the next section, the differential scaling of the components can by employed to serve further needs.

To measure the accuracy of the inverse kinematics approximation, we determine the deviation between the goal pose and the actually attained position after back-transforming (true map) the resulting angles computed by the PSOM. Two further question are studied in this case:

1. What is the influence of using tools with different length l_z mounted on the last robot segment?

2. What is the influence of standard and Chebyshev-spaced sampling of training points inside their working interval? When the data values (here 3 per axis) are sampled proportional to the Chebyshev zeros in the unit interval (Eq. 6.3), the border samples are moved by a constant fraction (here 16 %) towards the center.

Tab. 8.2 summarizes the resulting mean deviation of the desired Cartesian positions and orientations. While the tool length l_z has only marginal influence on the performance, the Chebyshev-spaced PSOM exhibits a signifcant advantage. As argued in Sect. 6.4, Chebyshev polynomials have arguably better approximation capabilities. However, in the case $n = 3$ both sampling schemes have equidistant node-spacing, but the Chebyshev-spacing approach contracts the marginal sampling points inside the working interval. Since the vicinity of each reference vector is principally approximated with high-accuracy, this advantage is better exploited if the reference training vector is located within the given workspace, instead of located at the border.

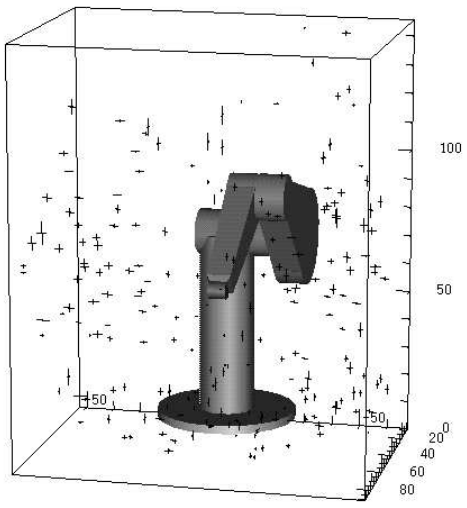


Figure 8.7: Spatial distribution of positioning errors of the PUMA robot arm using the 6 D inverse kinematics transform computed with a 3× 3× 3× 3× 3× 3 C-PSOM. The six-dimensional manifold is embedded in a 15-dimensional $\vec{r}, \vec{a}, \vec{n}, \vec{\theta}$-space.

The spatial distribution of the resulting $\vec{r}$ deviations is displayed in Fig. 8.7 (of the third case in Tab. 8.2). The local deviations are indicated

by little (double sized) cross-marks in the perspective view of the Puma's workspace.

	Cartesian position $\Delta\vec{r}$	
PSOM Type	Average	NRMS
3×3×3 PSOM	17 mm	0.041
3×3×3 C-PSOM	11 mm	0.027
4×4×4 PSOM	2.4 mm	0.0061
4×4×4 C-PSOM	1.7 mm	0.0042
5×5×5 PSOM	0.11 mm	0.00027
5×5×5 C-PSOM	0.091 mm	0.00023
3×3×3 L-PSOM of 4×4×4	6.7 mm	0.041
3×3×3 L-PSOM of 5×5×5	2.4 mm	0.0059
3×3×3 L-PSOM of 7×7×7	1.3 mm	0.018

Table 8.3: 3 DOF inverse Puma robot kinematics accuracy using several PSOM architectures including the equidistantly ("PSOM"), Chebyshev spaced ("C-PSOM"), and the local PSOM ("L-PSOM").

The full 6-dimensional kinematics problem is already a rather demanding task. Most neural network applications in this problem domain have considered lower dimensional transforms, for instance (Kuperstein 1988) ($m = 5$), (Walter, Ritter, and Schulten 1990) ($m = 3$), (Ritter et al. 1992) ($m = 3$ and $m = 5$), and (Yeung and Bekey 1993) ($m = 3$); all of them use several thousand training samples.

To set the present approach into perspective with these results, we investigate the same Puma robot problem, but with the three wrist joints fixed. Then, we may reduce the embedding space X to the essential variables $(\theta_1, \theta_2, \theta_3, p_x, p_y, p_z)$. Again using only three nodes per axis we require only 27 reference vectors $\mathbf{w_a}$ to specify the PSOM. Using the same joint ranges as in the previous case we obtain the results of Tab. 8.3 for several PSOM network architectures and training set sizes.

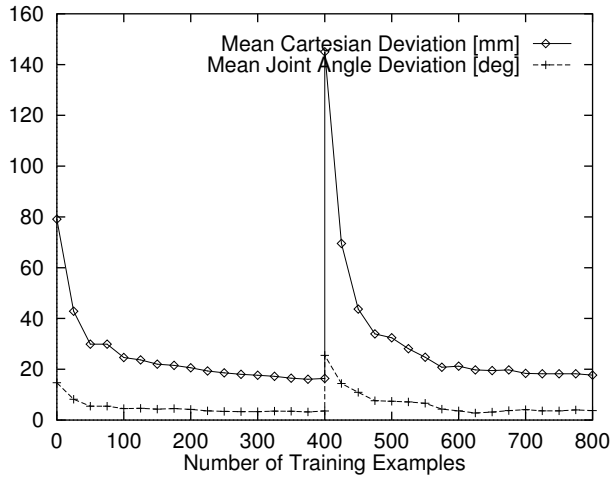


Figure 8.8: The positioning capabilities of the 3×3×3 PSOM network over the course of learning. The graph shows the mean Cartesian $\langle|\Delta\vec{r}|\rangle$ and angular $\langle|\Delta\vec{\theta}|\rangle$ deviation versus the number of already experienced learning examples. After 400 training steps the last arm segment was suddenly elongated by 150 mm (≈10 % of the linear work-space dimensions.)

8.3 Puma Kinematics: Noisy Data and Adaptation to Sudden Changes

The following experiment shows the adaptation capabilities of the PSOM in the 3 D inverse Puma kinematics task. Here, in contrast to the previous case, the initial training data is corrupted by noise. This may happen when only poor measurement instruments or limited time are available to make a quick and dirty initial "mapping guess". Fig. 8.8 presents the mean deviation of the joint angles $\langle|\Delta\vec{\theta}|\rangle$ and the back-transformed Cartesian deviation $\langle|\Delta\vec{r}|\rangle$ from the desired position (tested on a separate test set) versus the number of already experienced fine-adaptation steps. The PSOM was initially trained with a data set with (zero mean) Gaussian noise with a standard deviation of 50 mm $\eta(0,\ 50\,\mathrm{mm})$ added to the Cartesian measurement. (The fine-adaptation of the only coarsely constructed 3×3×3 C-PSOM employed Eq. 4.14 with $\epsilon = 0.7$ decreasing exponentially to 0.3 during the course of learning with two times 400 steps). In the early learning phase the position accuracy increased rapidly within the first 50–100 learning examples and reached the final average positioning error asymp-

totically.

A very important advantage of self-learning algorithms is their ability to adapt to different and also changing environments. To demonstrate the adaptability of the network, we interrupted the learning procedure after 400 training steps and extended the last arm segment by 150 mm ($l'_z = 350\,mm$). The right side of Fig. 8.8 displays how the algorithm responded. After this drastic change of the robot's geometry only about 100 further iterations where necessary to re-adapt the network for regaining the robot's previous positioning accuracy.

8.4 Resolving Redundancy by Extra Constraints for the Kinematics

The control of redundant degrees-of-freedom (DOF) is an important problem for manipulators built for dextrous operations. A particular task has a minimal requirement with respect to the manipulator's ability to move freely. When the task leaves the kinematics problem under-specified, there is not one possible solution, instead there exists a higher-dimensional solution space, which is compatible with the task specification. The practice requires a mechanism which determines exactly one solution. Naturally, it is desirable that these mechanisms offer a high degree of flexibility for commanding the robot task.

In this section the PSOM will be employed to elegantly realize an integrated system. Important is the flexible selection mechanism for the input sub-space components and the concept of modulating the cost function, as it was introduced in Sec. 6.2.

We return to the full 6 DOF Puma kinematics problem (Sec. 8.2) and use the PSOM to solve the following, typical redundancy problem: e.g., specifying only the 3 D target positioning $\vec{r}$ without any special target orientation, will leave three remaining DOFs open. In this under-constrained case the solutions form a continuous 3 D space. It is this redundancy that we want to use to meet additional constraints — in contrast to the discontinuous redundancies by multiple compatible robot configurations. Here we stay with the right-arm, elbow-up, no-wrist-flip configuration seen in Fig. 8.7 (see also Fu et al. 1987).

The PSOM input sub-space selection mechanism (matrix $\mathbf{P}$) facilitates

simple augmentation of the embedding space X with extraneous components (note, they do not affect the normal operation.) Those can be used to formulate additional cost function terms and can be activated whenever desired. The cost function terms can be freely constructed in various functional dependencies and are supplied during the learning phase of the PSOM.

The best-match location s^* is under-constrained, since $|I| = 3 < m = 6$ (in contrast to the cases described in Sec. 5.6.) Certainly, the standard best-match search algorithm will find one possible solution — but it can be any compatible solution and it will depend on the initial start condition $\mathrm{s}_{t=0}$.

Here, the PSOM offers a versatile way of formulating extra goals or constraints, which can be turned on and off, depending on the situation and the user's desire. For example, of particular interest are:

Minimal joint movement: "fast" or "lazy" robot. One practical goal can be: reaching the next target position with minimum motor action. This translates into finding the shortest path from the current joint configuration $\vec{\theta}_{curr}$ to a new $\vec{\theta}$ compatible with the desired Cartesian position $\vec{r}$.

Since the PSOM is constructed on a hyper-lattice in θ, finding the shortest route s in S is equivalent to finding the shortest path in θ. Thus, all we need to do is to start the best-match search at the best-match position s^*_{curr} belonging to the current position, and the steepest gradient descent procedure will solve the problem.

Orientation preference: the "traditional solution". If a certain end effector approach direction, for example a top–down orientation, is preferred, the problem transforms into the standard mixed position / orientation task, as described above.

Maximum mobility reserve: "comfortable configuration". If no further orientation constraints are given, it might be useful to gain a large joint mobility reserve — a reserve for further actions and re-actions to unforeseen events.

Here, the latter case is of particular interest. A high mobility reserve means to stay away from configurations close to any range limits. We

model this goal as a "discomfort" term in the cost function $E(\vec{\theta})$ and demonstrate how to incorporate extra cost terms in the standard PSOM mechanism.

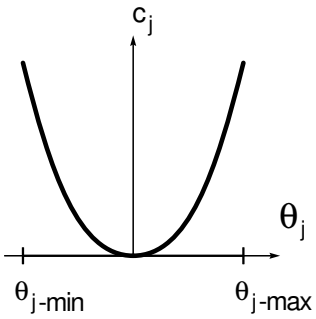


Figure 8.9: "Discomfort" cost function $c_j(\theta_j) = \left(2\frac{\theta_j - \theta_j^{mid}}{\theta_j^{max} - \theta_j^{min}}\right)^2$ for each joint angle i. A target value of zero, will attract the best-match towards the joint range center θ_j^{mid}.

Fig. 8.9 shows a suitable cost function term, which is constructed by a parabola shaped function $c_j(\theta_j)$ for all joint angles $\theta_{1...6}$. $c_j(\theta_j)$ is zero at the interval midpoint θ_j^{mid} and positive at both joint range limits. The 15-dimensional embedding space X is augmented to 21 dimensions such that all training vectors $\mathbf{w}$ become extended by the tuple $c_1, \ldots, c_6$. If the corresponding p_k in the selection matrix $\mathbf{P}$ are chosen as zero, the PSOM provides the same kinematics mapping as in the absence of the extension. However, when we now turn on the new $\mathbf{P}$ elements ($p_{16...21} > 0$), and set the input components to zero ($x_{16...21} = 0$), the iterative best-match procedure of the PSOM tries to simultaneously satisfy the constraints imposed by the kinematics equation together with the constraints $c_j = 0$. The latter

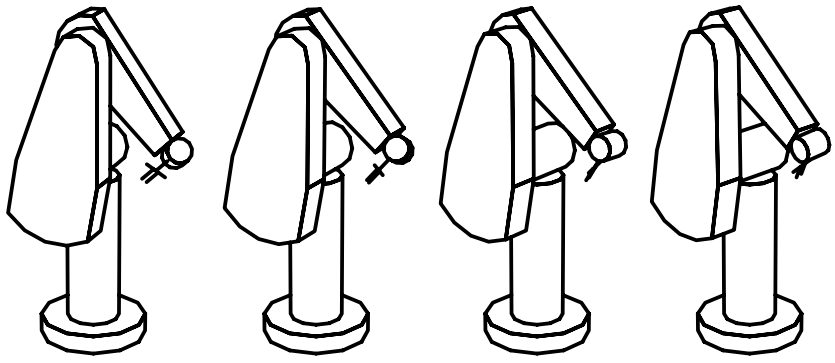


Figure 8.10: Series of intermediate steps for optimizing the remaining joint angle mobility in the same position.

attracts the solution to the particular single configuration with all joints in mid range position. Any further kinematics specification is usually conflicting, and the result therefore a compromise (the least-square optimum; $|I| > 6$). How to solve this conflict?

To avoid this mis-attraction effect, the auxiliary constraint terms $p_k = p_k(t)$

1. should be generally kept small, otherwise the solution would be too strongly attracted to the single mid-point position;

2. should decay during the gradient descent iteration. The final step should be done with all extra terms c_j weighted with factors p_k zero (here $p_{16...21} = 0$). This assures that the final solution will be – without compromise – within the solution space, spanned by the primary goal, here the end-effector position.

To demonstrate the impact of the auxiliary constraints the augmented $m = 6$ PSOM is engaged to re-arrange a suitable robot arm configuration. The initial starting position is already a solution of the desired end-effector positions and Fig. 8.10 and Fig. 8.11 show intermediate steps in approaching the desired result. Here, the extra cost components were weighted in a fixed ratio of 0:0.04:0.06:1:1:0.04 among each other and weighted initially by 0.5 % with respect to the $\vec{r}$ components (see Eq. 8.3). During intermediate best-match search steps all weights gradually decay to zero. The stroboscopic image (Fig. 8.11d) shows how the arm frees itself from an extremal configuration (position close to the limit) to a configuration leaving more space to move freely.

It should be emphasized that several constraint functions can be simultaneously inserted and turned "on" and "off" to suit the current needs. This a good example of the strength of a versatile and flexible input selection mechanism. The implementation should care that any in-active augmentations (with $p_\mu = 0$) of the embedding space X are handled efficiently, i.e. all related component operations are skipped. By this means, the extraneous features do not impair the PSOM's performance, but can be engaged at any time.

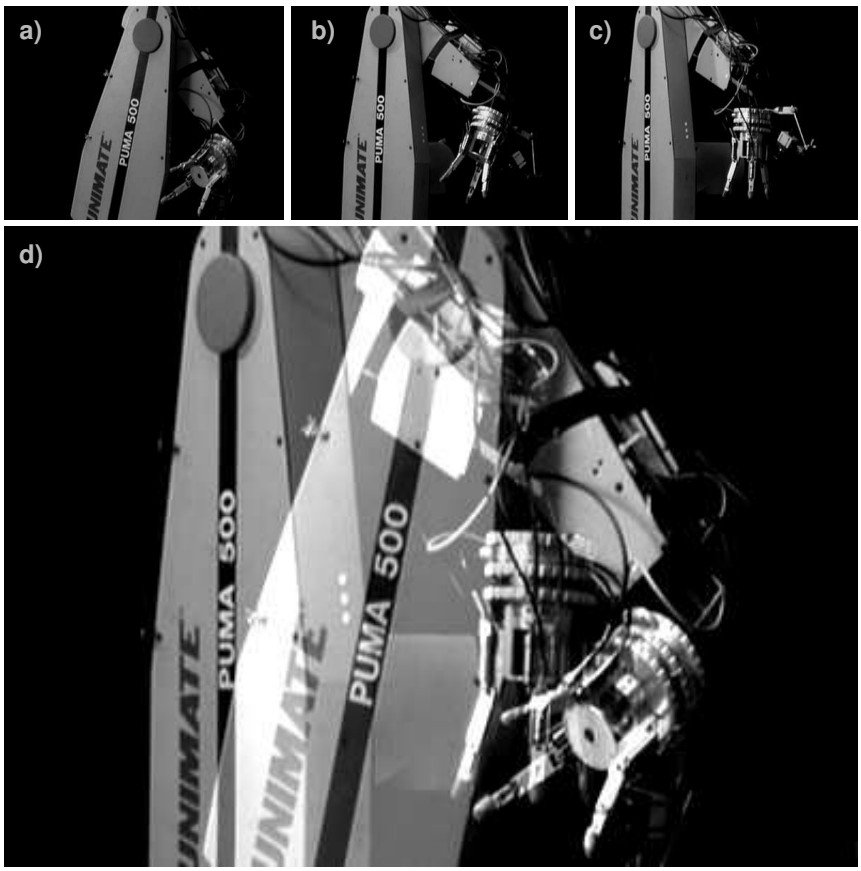


Figure 8.11: The PSOM resolves redundancies by extra constraints in a convenient functional definition. *(a-c)* Sequence of images, showing how the Puma manipulator turns from a joint configuration close to the range limits *(a)* to a configuration with a larger mobility reserve *(c)*. The stroboscopic picture *(d)* demonstrates that the same tool center point is kept.

8.5 Summary

The PSOM learning algorithm shows very good generalization capability for smooth continuous mapping tasks. This property becomes highlighted at the robot finger inverse kinematics problem with 3 inherent degrees-of-freedom (see also 6 D kinematics). Since in many robotics learning tasks the data set can be actively sampled, the PSOM's ability to construct the high-dimensional manifold from a small number of training data turns out to be here a many-sided beneficial mechanism for rapid learning.

Furthermore, the associative mapping concept has several interesting properties. Several coordinate spaces can be maintained and learned simultaneously, as shown for the robot finger example. This multi-way mapping solves, e.g. the forward and inverse kinematics with the very same network. This simplifies learning and avoids any asymmetry of separate learning modules. As pointed out by Kawato (1995), the learning of bi-directional mappings is not only useful for the planning phase (action simulation), but also for bi-directional sensor–motor integrated control.

By the method of dynamic cost function modulation the PSOM's internal best-match search can be employed for partially meeting additional, possibly conflicting target functions. This scheme was demonstrated in the redundancy problem of the 6 DOF inverse robot kinematics.

Chapter 9

Context Dependent Mixture-of-Expertise: Investment Learning

If one wants to learn with extremely few examples, one inevitably faces a dilemma: on the one hand, with few examples one can only determine a rather small number of adaptable parameters and, as a consequence, the learning system must be either very simple, or, and usually this is the relevant alternative, it must have a structure that is already well-matched to the task to be learned. On the other hand, however, having to painstakingly pre-structure a system by hand is precisely what one wants to avoid when using a learning approach.

It is possible to find a workable compromise that can cope with this dilemma, i.e., that somehow allows the structuring of a system without having to put in too much by hand?

9.1 Context dependent "skills"

To be more concrete, we want to consider the learning of a "skill" which is dependent on some *environment* or *system context*. The notion of "skill" is very general and includes a task specific, hand-crafted function mapping mechanism, a control system, as well as a general learning system. As illustrated by Fig. 9.1, we assume:

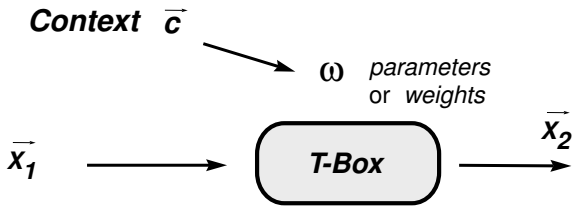


Figure 9.1: The T-BOX maps between different task variable sets within a certain context ($\vec{c}$), describable by a set of parameters ω.

- that the "skill" can be acquired by a "**t**ransformation box" ("T-BOX"), which is a suitable building block with learning capabilities; the T-BOX is responsible for the multi-variate, continuous-valued mapping $T : \vec{\mathbf{x}}_1 \rightarrow \vec{\mathbf{x}}_2$, transforming between the two task-variable sets $\vec{\mathbf{x}}_1$ and $\vec{\mathbf{x}}_2$.

- the mapping "skill" T-BOX is internally modeled and determined by a set of parameters ω (which can be accessed from outside the "black box", which makes the T-BOX rather an open "white box");

- the correct parameterization ω changes smoothly with the *context* of the system;

- the situational context can be observed and is associated with a set of suitable sensor values $\vec{c}$ (some of them are possibly expensive and temporarily unavailable);

- the context changes only from time to time, or on a much larger time scale, than the time scale on which the task mapping T-BOX is employed.

The conventional approach is to consider the joined problem of learning the mapping from all relevant input values, $\vec{x}_1, \vec{c}$ to the desired output $\vec{x}_2$. This leads to large, specialized networks. Their disadvantages are first, the possible catastrophic interference (after-learning in a situated context may effect other contexts in an uncontrolled way, see Sec. 3.2); and second, their low modularity and re-usability.

9.2 "Investment Learning" or "Mixture-of-Expertise" Architecture

Here, we approach a solution in a modular way and suggest to *split learning* structurally and temporally: the **structural** split is implemented at the level of the learning moduls:

- the T-BOX;
- the META-BOX, which has the responsibility for providing the mapping between the context information $\vec{c}$ to the weight or parameter set ω.

The **temporal** split is implemented at the learning itself:

- The first, the ***investment learning*** stage may be slow and has the task to pre-structure the system for
- the ***one-shot adaptation phase***, in which the specialization to a particular solution (within the chosen domain) can be achieved extremely rapidly.

These two stages are described next.

9.2.1 Investment Learning Phase

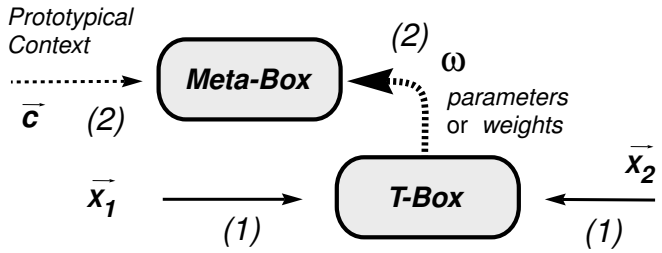


Figure 9.2: The *Investment Learning Phase*.

In the *investment learning* phase a set of *prototypical context situations* is experienced: in each context j the T-BOX is trained and the appropriate set of weights / parameters ω_j determined (see Fig. 9.2, arrows *(1)*). It

serves together with the context information $\vec{c}$ as a high-dimensional training data vector for the META-BOX *(2)*. During the *investment learning* phase the META-BOX mapping is constructed, which can be viewed as the stage for the *collection of expertise* in the suitably chosen prototypical contexts.

9.2.2 One-shot Adaptation Phase

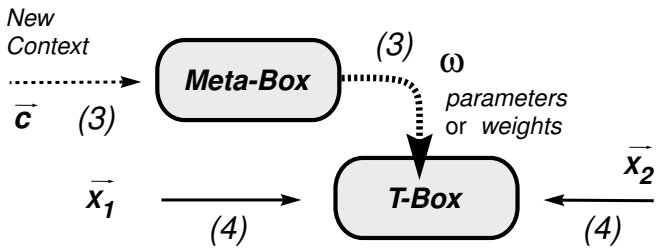


Figure 9.3: The *One-shot Adaptation Phase*.

After the META-BOX has been trained, the task of adapting the "skill" to a new system context is tremendously accelerated. Instead of any time-consuming re-learning of the mapping T this adjustment now takes the form of an *immediate* META-BOX $\rightarrow$ T-BOX mapping or *"one-shot adaptation"*. As illustrated in Fig. 9.3, the META-BOX maps a new (unknown) context observation $\vec{c}_{new}$ *(3)* into the parameter / weight set ω_{new} for the T-BOX. Equipped with ω_{new}, the T-BOX provides the desired mapping T_{new} *(4)*.

9.2.3 "Mixture-of-Expertise" Architecture

It is interesting to compare this approach with a feed-forward architecture which Jordan and Jacobs (1994) coined **"mixture-of-experts"**. As illustrated in Fig. 9.4 a number of "experts" receive the same input task variables together with the context information $\vec{c}$. In parallel, each expert produces an output and contributes – with an individual weight – to the overall system result. All these weights are determined by the **"gating network"**, based on the context information $\vec{c}$ (see also LLM discussion in Sec. 3.8).

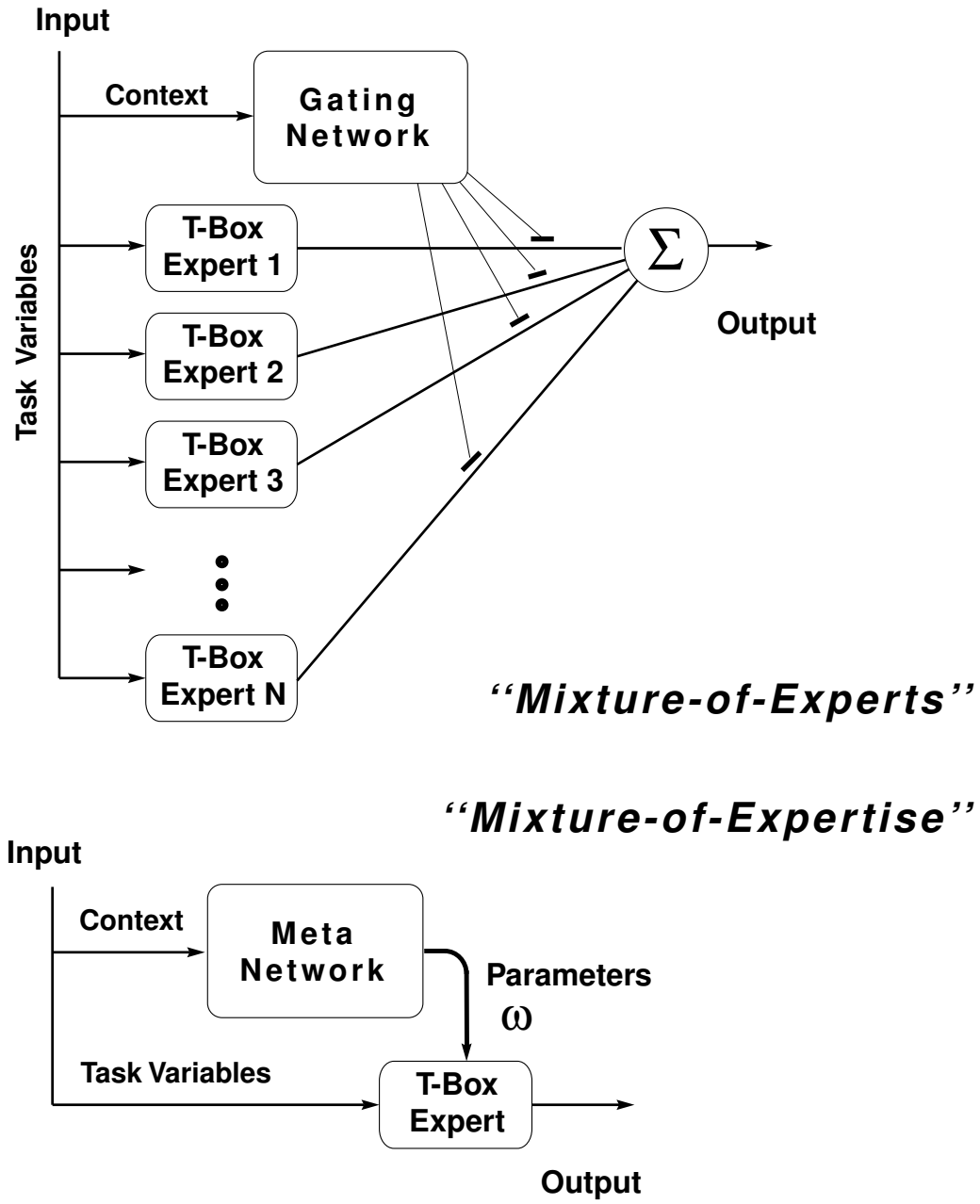


Figure 9.4: The *"Mixture-of-Experts"* architecture versus the *"Mixture-of-Expertise"* architecture.

The lower part of Fig. 9.4 redraws the proposed hierarchical network scheme and suggests to name it **"mixture-of-expertise"**. In contrast to the specialized "experts" in Jordan's picture, here, one single "expert" gathers specialized **"expertise"** in a number of prototypical context situations (see *investment learning phase*, Sec. 9.2.1). The META-BOX is responsible for the non-linear "mixture" of this "expertise".

With respect to networks' requirements for memory and computation, the "mixture-of-expertise" architecture compares favorably: the "expertise" (ω) is gained and implemented in a single "expert" network (T-BOX). Furthermore, the META-BOX needs to be re-engaged only when the context is changed, which is indicated by a deviating sensor observation $\vec{c}$.

However, this scheme requires from the learning implementation of the T-BOX that the parameter (or weight) set ω is represented as a continuous function of the context variables $\vec{c}$. Furthermore, different "degenerate" solutions must be avoided: e.g. a regular multilayer perceptron allows many weight permutations ω to achieve the same mapping. Employing a MLP in the T-BOX would result in grossly inadequate interpolation between prototypical "expertises" ω_j, denoted in different kinds of permutations. Here, a suitable stabilizer would be additionally required.

Please note, that the new "mixture-of-expertise" scheme does not only identify the context and retrieve a suitable parameter set (association). Rather it achieves a high-dimensional generalization of the learned (invested) situations to new, previously unknown contexts.

A "mixture-of-expertise" aggregate can serve as an expert module in a hierarchical structure with more than two levels. Moreover, the two architectures can be certainly combined. This is particularly advantageous when very complex mappings are smooth in certain domains, but non-continuous in others. Then, different types of learning experts, like PSOMs, Meta-PSOMs, LLMs, RBF and others can be chosen. The domain weighting can be controlled by a competitive scheme, e.g. RBF, LVQ, SOM, or a "Neural-Gas" network (see Chap. 3).

9.3 Examples

The concept imposes a strong need for efficient learning algorithms: to keep the number of required training examples manageable, those should

be efficient in particular with respect to the number of required training points.

The PSOM network appears as a very attractive solution, but not the only possible one. Therefore, the first example will compare three ways to apply the mixture-of-expertise architecture to a four DOF problem concerned about coordinate transformation. Two further examples demonstrate a visuo-motor coordination tasks for mono- and binocular camera sight.

9.3.1 Coordinate Transformation with and without Hierarchical PSOMs

This first task is related to the visual object orientation finder example presented before in Sec. 7.2 (see also Walter and Ritter 1996a). Here, an interesting skill for a robot could be the correct coordinate transformation from a camera reference frame (world or tool; yielding coordinate values $\vec{x}_1$) to the object centered frame (yielding coordinate values $\vec{x}_2$). This mapping would have to be represented by the T-BOX. The *"context"* would be the current orientation of the object relative to the camera.

Fig. 9.5 shows three ways how the investment learning scheme can be implemented in that situation. All three share the same PSOM network type as the META-BOX building block. As already pointed out, the "Meta-PSOM" bears the advantage that the architecture can easily cope with situations where various (redundant) sensory values are or are not available (dynamic sensor fusion problem).

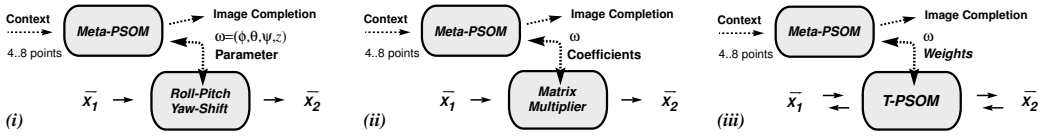


Figure 9.5: Three different ways to solve the context dependent, or investment learning task.

The first solution (i) uses the Meta-PSOM for the reconstruction of object pose in roll-pitch-yaw-depth values from Sec. 7.2. The T-BOX is given by the four successive homogeneous transformations (e.g. Fu et al. 1987) on the basis of the ϕ, θ, ψ, z values obtained from the Meta-PSOM.

The solution (ii) represents the coordinate transformation as the product of the four successive transformations. Thus, in this case the Meta-PSOM controls the coefficients of a matrix multiplication. As in (i), the required parameter values ω are gained by a suitable calibration, or system identification procedure.

When no explicit ansatz for the T-BOX is readily available, we can use method (iii). Here, for each prototypical context, the required T-mapping is learned by a network and becomes encoded in its weight set ω. For this, one can use any trainable network that meets the requirement stated at the end of the previous section. However, PSOMs are a particularly convenient choice, since they can be directly constructed from a small data set and additionally offer the advantage of associative multi-way mappings.

In this example, we chose for the T-BOX a $2\times\ 2\times\ 2$ "T-PSOM" that implements the coordinate transform for *both directions simultaneously*. Its training required eight training vectors arranged at the corners of a cubical grid, e.g. similar to the cube structure depicted in Fig. 7.2.

In order to compare approaches $(i) - (iii)$, the transformation T-BOX accuracy was averaged over a set of 50 contexts (given by 50 randomly chosen object poses), each with 100 object volume points $\vec{x}_2$ to be transformed into camera coordinates $\vec{x}_1$.

T-BOX	x - RMS [L]	y - RMS [L]	z - RMS [L]
(i) (ϕ, θ, ψ, z)	0.025	0.023	0.14
(ii) $\{A_{ij}\}$	0.016	0.015	0.14
(iii) PSOM	0.015	0.014	0.12

Table 9.1: Results for the three variants in Fig. 9.5.

Comparing the RMS results in Tab. 9.1 shows, that the PSOM approach *(iii)* can fully compete with the dedicated hand-crafted, one-way mapping solutions *(i)* and *(ii)*.

9.3.2 Rapid Visuo-motor Coordination Learning

The next example is concerned with a robot sensorimotor transformation. It involves the Puma robot manipulator, which is monitored by a camera, see Fig. 9.6. The robot is positioned behind a table and the entire scene is

displayed on a monitor. With a mouse-click, a user can select on the monitor some target point of the displayed table area. The goal is to move the robot end effector to the indicated position on the table. This requires to compute a transformation $T : \vec{x} \leftrightarrow \vec{u}$ between coordinates on the monitor (or "camera retina" coordinates) and corresponding world coordinates $\vec{x}$ in the frame of reference of the robot. This transformation depends on several factors, among them the relative position between the robot and the camera. The learning task (for the later stage) is to rapidly re-learn this transformation whenever the camera has been repositioned.

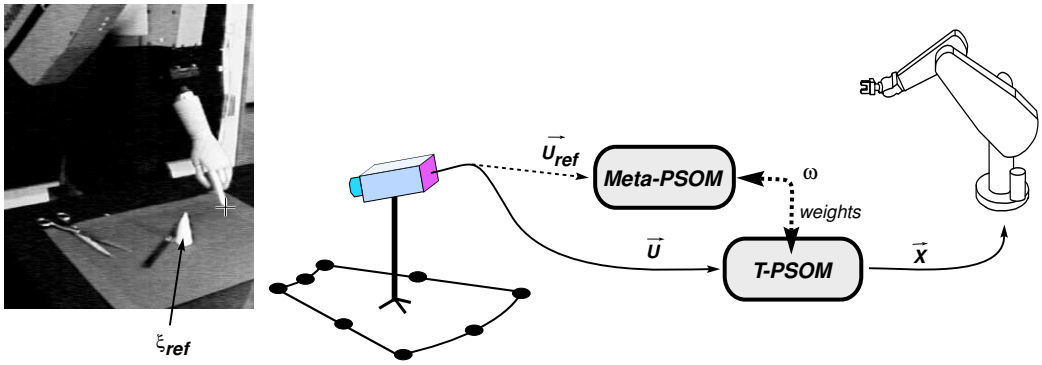


Figure 9.6: Rapid learning of the 2D visuo-motor coordination for a camera in changing locations. The basis T-PSOM is capable of mapping to (and from) the Cartesian robot world coordinates $\vec{x}$, and the location of the end-effector (here the wooden hand replica) in camera coordinates $\vec{u}$ (see cross mark.) In the pre-training phase, nine basis mappings are learned in prototypical camera locations (chosen to lie on the depicted grid.) Each mapping gets encoded in the weight parameters $\vec{\omega}$ of the T-PSOM and serves then, together with the system context observation $\vec{u}_{ref}$ (here, e.g. the cone tip), as a training vector for the Meta-PSOM.

In other words, here, the T-PSOM has to represent the transformation $T : \vec{x} \leftrightarrow \vec{u}$ with the camera position as the additional context. To apply the previous scheme, we must first learn ("investment stage") the mapping T for a set of prototypical contexts, i.e., camera positions.

To keep the number of prototype contexts manageable, we reduce some DOFs of the camera by requiring fixed focal length, camera tripod height, and roll angle. To constrain the elevation and azimuth viewing angle, we choose one fixed land mark, or "fixation point" ξ_{fix} somewhere centered in the region of interest. After repositioning the camera, its viewing angle

must be re-adjusted to keep this fixation point visible in a constant image position, serving at the same time the need of a fully visible region of interest. These practical instructions achieve the reduction of free parameters per camera to its 2D lateral position, which can now be sufficiently determined by a *single* extra observation of a chosen auxiliary world reference point ξ_{ref}. We denote the camera image coordinates of ξ_{ref} by $\vec{u}_{ref}$. By reuse of the cameras as a "context" or "environment sensor", $\vec{u}_{ref}$ now implicitly encodes the camera position.

For the present investigation, we chose from this set 9 different camera positions, arranged in the shape of a 3×3 grid (Fig. 9.6). For each of these nine contexts, the associated mapping $T \equiv T_j$, $(j = 1, 2 \ldots 9)$ is learned by a T-PSOM by visiting a rectangular grid set of end effector positions ξ_i (here we visit a 3×3 grid in $\vec{x}$ of size $30 \times 30\,\mathrm{cm}^2$) jointly with the location in camera retina coordinates (2D) $\vec{u}_i$. This yields the tuples $(\vec{x}_i, \vec{u}_i)$ as the training vectors $\mathbf{w_a}_i$ for the construction of a weight set $\vec{\omega}_j$ (valid for context j) for the T-PSOM in Fig. 9.3.

Each T_j (the T-PSOM in Fig. 9.3, equipped with weight set $\vec{\omega}_j$) solves the mapping task only for the camera position for which T_j was learned. Thus there is not yet any particular advantage to other, more specialized methods for camera calibration (Fu, Gonzalez, and Lee 1987). However, the important point is, that now we can employ the Meta-PSOM to rapidly map a new camera position into the associated transform T by *interpolating* in the space of the previously constructed basis mappings T_j.

The constructed input-output tuples $(\vec{u}_{ref/j}, \vec{\omega}_j)$, $j \in \{1, \ldots, 9\}$, serve as the training vectors for the construction of the Meta-PSOM in Fig. 9.3 such that each $\vec{u}_{ref}$ observation that pertains to an intermediate camera positioning becomes mapped into a weight vector $\vec{\omega}$ that, when used in the base T-PSOM, yields a suitably *interpolated mapping* in the space spanned by the basis mappings T_j.

This enables in the following *one-shot adaptation* for new, unknown camera places. On the basis of *one single* observation $\vec{u}_{ref,new}$, the Meta-PSOM provides the weight pattern $\vec{\omega}_{new}$ that, when used in the T-PSOM in Fig. 9.3, provides the desired transformation T_{new} for the chosen camera position. Moreover (by using different projection matrices $\mathbf{P}$), the T-PSOM can be used for different mapping directions, formally:

$$\vec{x}(\vec{u}) \quad = \quad F^{u \mapsto x}_{T-PSOM}(\vec{u}; \quad \vec{\omega}(\vec{u}_{ref})) \tag{9.1}$$

$$\vec{u}(\vec{x}) = F^{x\mapsto u}_{T-PSOM}(\vec{x};\quad \vec{\omega}(\vec{u}_{ref})) \tag{9.2}$$

$$\vec{\omega}(\vec{u}_{ref}) = F^{u\mapsto\vec{\omega}}_{Meta-PSOM}(\vec{u}_{ref};\quad \vec{\omega}_{Meta}) \tag{9.3}$$

Table 9.2 shows the experimental results averaged over 100 random locations ξ (from within the range of the training set) seen from 10 different camera locations, from within the 3×3 roughly radial grid of the training positions, located at a normal distance of about 65–165 cm (to work space center, about 80 cm above table, total range of about 95–195 cm), covering a 50° sector. For identification of the positions ξ in image coordinates, a tiny light source was installed at the manipulator tip and a simple procedure automatized the finding of $\vec{u}$ with about ±1 pixel accuracy. For the achieved precision it is important that all learned T_j share the same set of robot positions ξ_i, and that the training sets (for the T-PSOM and the Meta-PSOM) are topologically ordered, here as two 3×3 grids. It is not important to have an alignment of this set to any exact rectangular grid in e.g. world coordinates, as demonstrated with the radial grid of camera training positions (see Fig. 9.6 and also Fig. 5.5).

	Directly trained T-PSOM		**T-PSOM with Meta-PSOM**	
pixel $\vec{u}\mapsto\vec{x}_{robot}\ \Rightarrow$ Cart. error $\Delta\vec{x}$	2.2 mm	0.021	3.8 mm	0.036
Cartesian $\vec{x}\mapsto\vec{u}\ \Rightarrow$ pixel error	1.2 pix	0.016	2.2 pix	0.028

Table 9.2: Mean Euclidean deviation (mm or pixel) and normalized root mean square error (NRMS) for 1000 points total in comparison of a directly trained T-PSOM and the described hierarchical PSOM-network, in the rapid learning mode with one observation.

These data demonstrate that the hierarchical learning scheme does not fully achieve the accuracy of a straightforward re-training of the T-PSOM after each camera relocation. This is not surprising, since in the hierarchical scheme there is necessarily some loss of accuracy as a result of the interpolation in the weight space of the T-PSOM. As further data becomes available, the T-PSOM can certainly be fine-tuned to improve the performance to the level of the directly trained T-PSOM. However, the possibility to achieve the already very good accuracy of the hierarchical approach with the first single observation per camera relocation is extremely attractive and may often by far outweigh the still moderate initial decrease that

is visible in Tab. 9.2.

9.3.3 Factorize Learning: The 3 D Stereo Case

The next step is the generalization of the monocular visuo-motor map to the stereo case of two independent movable cameras. Again, the Puma robot is positioned behind the table and the entire scene is displayed on two windows on a computer monitor. By mouse-pointing, the user can, for example, select one point on the monitor and the position on a line appearing in the other window, to indicate a goal position for the robot end effector, see Fig. 9.7. This requires to compute the transformation T between the combined pair of pixel coordinates $\vec{u} = (\vec{u}^L, \vec{u}^R)$ on the monitor images and corresponding 3 D world coordinates $\vec{x}$ in the robot reference frame — or alternatively — the corresponding six robot joint angles $\vec{\theta}$ (6 DOF). Here we demonstrate an integrated solution, offering both solutions with the same network (see also Walter and Ritter 1996b).

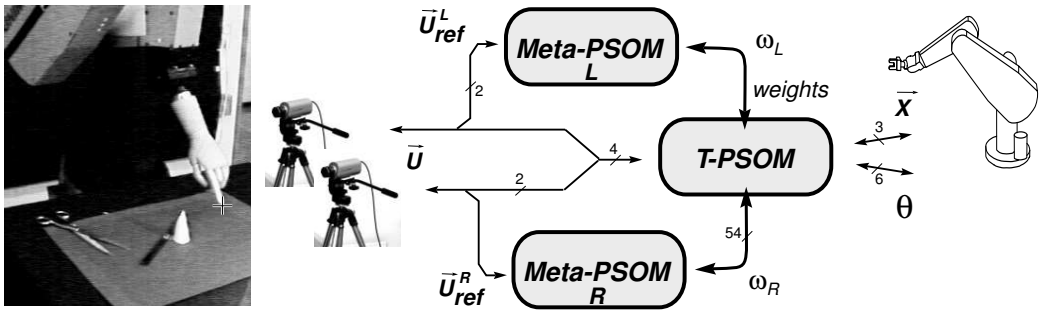


Figure 9.7: Rapid learning of the 3D visuo-motor coordination for two cameras. The basis T-PSOM ($m = 3$) is capable of mapping to and from three coordinate systems: Cartesian robot world coordinates, the robot joint angles (6-DOF), and the location of the end-effector in coordinates of the two camera retinas. Since the left and right camera can be relocated independently, the weight set of T-PSOM is split, and parts ω_L, ω_R are learned in two separate Meta-PSOMs ("L" and "R").

The T-PSOM learns each individual basis mapping T_j by visiting a rectangular grid set of end effector positions ξ_i (here a 3×3×3 grid in $\vec{x}$ of size $40{\times}40{\times}30\,\mathrm{cm}^3$) jointly with the joint angle tuple $\vec{\theta}_j$ and the location in camera retina coordinates (2D in each camera) $\vec{u}^L_j, \vec{u}^R_j$. Thus the training vectors $\mathbf{w_a}_i$ for the construction of the T-PSOM are the tuples $(\vec{x}_i, \vec{\theta}_i, \vec{u}^L_i, \vec{u}^R_i)$.

In the investing pre-training phase, nine mappings T_j are learned by the T-PSOM, each camera visiting a 3×3 grid, sharing the set of visited robot positions ξ_i. As Fig. 9.3 suggests, normally the entire weight set ω serves as part of the training vector to the Meta-PSOM. Here the problem factorizes since the left and right camera change tripod place independently: the weight set of the T-PSOM is split, and the two parts can be learned in separate Meta-PSOMs. Each training vector $\mathbf{w_{a_j}}$ for the left camera Meta-PSOM consists of the context observation $\vec{u}^L_{ref}$ and the T-PSOM weight set part $\omega_L = (\vec{u}^L_1, \ldots, \vec{u}^L_{27})$ (analogously for the right camera Meta-PSOM.)

Also here, only *one single* observation $\vec{u}_{ref}$ is required to obtain the desired transformation T. As visualized in Fig. 9.7, $\vec{u}_{ref}$ serves as the input to the second level Meta-PSOMs. Their outputs are interpolations between previously learned weight sets, and they project directly into the weight set of the basis level T-PSOM.

The resulting T-PSOM can map in various directions. This is achieved by specifying a suitable distance function $dist(\cdot)$ via the projection matrix $\mathbf{P}$, e.g.:

$$\vec{x}(\vec{u}) = F^{u\mapsto x}_{T-PSOM}(\vec{u};\quad \omega_L(\vec{u}^L_{ref}), \omega_R(\vec{u}^R_{ref})) \tag{9.4}$$

$$\vec{\theta}(\vec{u}) = F^{u\mapsto \theta}_{T-PSOM}(\vec{u};\quad \omega_L(\vec{u}^L_{ref}), \omega_R(\vec{u}^R_{ref})) \tag{9.5}$$

$$\vec{u}(\vec{x}) = F^{x\mapsto u}_{T-PSOM}(\vec{x};\quad \omega_L(\vec{u}^L_{ref}), \omega_R(\vec{u}^R_{ref})) \tag{9.6}$$

$$\omega_L(\vec{u}^L_{ref}) = F^{u\mapsto \omega}_{Meta-PSOM,L}(\vec{u}^L_{ref};\quad \Omega_L);\quad \text{analog } \omega_R(\vec{u}^R_{ref}) \tag{9.7}$$

Mapping Direction	**Directly trained T-PSOM**		**T-PSOM with Meta-PSOM**	
pixel $\vec{u} \mapsto \vec{x}_{robot}$ $\Rightarrow$ Cartesian error $\Delta\vec{x}$	1.4 mm	0.008	4.4 mm	0.025
Cartesian $\vec{x} \mapsto \vec{u}$ $\Rightarrow$ pixel error	1.2 pix	0.010	3.3 pix	0.025
pixel $\vec{u} \mapsto \vec{\theta}_{robot}$ $\Rightarrow$ Cartesian error $\Delta\vec{x}$	3.8 mm	0.023	5.4 mm	0.030

Table 9.3: Mean Euclidean deviation (mm or pixel) and normalized root mean square error (NRMS) for 1000 points total in comparison of a directly trained T-PSOM and the described hierarchical Meta-PSOM network, in the rapid learning mode after *one single* observation.

Table 9.3 shows experimental results averaged over 100 random locations ξ (from within the range of the training set) seen in 10 different

camera setups, from within the 3×3 square grid of the training positions, located in a normal distance of about 125 cm (center to work space center, 1 m^2), covering a disparity angle range of $25°$–$150°$.

The achieved accuracy of 4.4 mm after learning by a single observation, compares very well with the total distance range 0.5–2.1 m of traversed positions. As further data becomes available, the T-PSOM can be fine-tuned and the performance improved to the level of the directly trained T-PSOM.

The next chapter will summarize the presented work.

Chapter 10

Summary

The main concern of this work is the development and investigation of new building blocks aiming at rapid and efficient learning. We chose the domain of continuous, high-dimensional, non-linear mapping tasks, as they often play an important role in sensorimotor transformations in the field of robotics.

The design of better re-usable building blocks, not only adaptive neural network modules, but also hardware, as well as software modules can be considered as the desire for efficient learning in a broader sense. The construction of those building blocks is driven by the given experimental situation. Similar to a training exercise, the procedural knowledge of, for example, interacting with a device is usually incorporated in a building block, e.g. a piece of software. The criterion to call this activity *"learning"* is whether this "knowledge" can be later used, more precisely, *re-used* in form of "association" or "generalization" in a new, previously unexpected application situation.

The first part of this work was directed at the robotics infrastructure investment: the building and development of a test and research platform around an industrial robot manipulator Puma 560 and a hydraulic multi-finger hand. We were particularly concerned about the interoperability of the complex hardware by general purpose Unix computers in order to gain the flexibility needed to interface the robots to distributed information processing architectures.

For more intelligent and task-oriented action schemata the availability of fast and robust sensory environment feedback is a limiting factor. Nevertheless, we encountered a significant lack in suitable and commer-

cially available sensor sub-systems. As a consequence, we started to enlarge the robot's sensory equipment in the direction of force, torque, and haptic sensing. We developed a multi-layer tactile sensor for detailed information on the current contact state with respect to forces, locations and dynamic events. In particular, the detection of incipient slip and timely changes of contact forces are important to improve stable fine control on multi-contact grasp and release operations of the articulated robot hand.

Returning to the more narrow sense of *rapid learning*, what is important?

To be practical, learning algorithms must provide solutions that can compete with solutions hand-crafted by a human who has analyzed the system. The criteria for success can vary, but usually the costs of gathering data and of teaching the system are a major factor on the side of the learning system, while the effort to analyze the problem and to design an algorithm is on the side of the hand crafted solution.

Here we suggest the "Parameterized Self-Organizing Map" as a versatile module for the rapid learning of high-dimensional, non-linear, smooth relations. As shown in a row of application examples, the PSOM learning mechanism offers excellent generalization capabilities based on a remarkably small number of training examples.

Internally, the PSOM builds an m-dimensional continuous mapping manifold, which is embedded in a higher d-dimensional task space ($d > m$). This manifold is supported by a set of reference vectors in conjunction with a set of basis functions. One favorable choice of basis functions is the class of (m-fold) products of Lagrange approximation polynomials. Then, the (m-dimensional) grid of reference vectors parameterizes a topologically structured data model.

This topologically ordered model provides curvature information — information which is not available within other learning techniques. If this assumed model is a good approximation, it significantly contributes to achieve the presented generalization accuracy. The difference of information contents — with and without such a topological order — was emphasized in the context of the robot finger kinematics example.

On the one hand, the PSOM is the continuous analog of the standard discrete "Self-Organizing Map" and inherits the well-known SOM's unsupervised learning capabilities (Kohonen 1995). One the other hand, the PSOM offers a most rapid form of "learning", i.e. the form of *immediate*

construction of the desired manifold. This requires to assign the training data set to the set of internal node locations. In other words, for this procedure the training data set must be known, or must be inferred (e.g. with the SOM scheme).

The applicability is demonstrated in a number of examples employing training data sets with the known topology of a multi-dimensional Cartesian grid. The resulting PSOM is immediately usable — without any need for time consuming adaptation sequences. This feature is extremely advantageous in all cases where the training data can be sampled actively. For example, in robotics, many sensorimotor transformations can be sampled in a structured manner, without any additional cost.

Irrespectively of how the data model was initially generated the PSOM can be *fine-tuned* on-line. Using the described error minimization procedure, a PSOM can be refined even in the cases of coarsely sampled data, when the original training data was corrupted by noise, or the underlying task is changing. This is illustrated by the problem of adapting to sudden changes in the robot's geometry and its corresponding kinematics.

The PSOM manifold is also called *parameterized associative map* since it performs *auto-associative completion* of partial inputs. This facilitates *multi-directional* mapping in contrast to only uni-directional feed-forward networks. Which components of the embedding space are selected as inputs, is simply determined by specifying the diagonal elements p_k of the projection matrix $\mathbf{P}$. This mechanism allows to easily augment the embedding space by further sub-spaces. As pointed out, the PSOM algorithm can be implemented, such that inactive components do not affect the normal PSOM operation.

Several examples demonstrate how to profitably utilize the multi-way association capabilities: e.g. feature sets can be completed by a PSOM in such a manner that they are invariant against certain operations (e.g. shifted/rotated object) and provide at the same pass the unknown operation parameter (e.g. translation, angles).

The same mechanism offers a very natural and flexible way of *sensor data fusion*. The incremental availability of more and more results from different sensors can be used to improve the measurement accuracy and confidence of recognition. Furthermore, the PSOM multi-way capability enables an effective way of inter-sensor coordination and sensor system guidance by predictions.

Generally, in robotics the availability of precise mappings from and to different variable spaces, including sensor, actuator, and reference coordinate spaces, plays a crucial role. The applicability of the PSOM is demonstrated in the robot finger application, where it solves the classical forward and inverse kinematics problem in Cartesian, as well as in the actuator piston coordinates — within the same PSOM. Here, a set of only 27 training data points turns out sufficient to approximate the 3 D inverse kinematics relation with a mean positioning deviation of about 1 % of the entire workspace range.

The ability to augment the PSOM embedding space allows to easily add a "virtual sensor" space to the usual sensorimotor map. In conjunction with the ability of rapid learning this opens the interesting possibility to demonstrate desired robot task performance. After this learning by demonstration phase, robot tasks can also be specified as perceptual expectations in this newly learned space.

The coefficients p_k can weight the components relative to each other, which is useful when input components are differently confident, important, or of uneven scale. This choice can be changed on demand and can even be modulated during the iterative completion process.

Internally, the PSOM associative completion process performs an iterative search for the *best-matching* parameter location in the mapping manifold. This minimization procedure can be viewed as a recurrent network dynamics with an *continuous attractor manifold* instead of just attractor points like in conventional recurrent associative memories. The required iteration effort is the price for rapid learning. Fortunately, it can be kept small by applying a suitable, adaptive second order minimization procedure (Sect. 4.5). In conjunction with an algorithmic formulation optimized for efficient computation also for high-dimensional problems, the completion procedure converges already in a couple of iterations.

For special purposes, the search path in this procedure can be directed. By modulating the cost function during the best-match iteration the PSOM algorithm offers to partly comply to an additional, second-rank goal function, possibly contradicting the primary target function. By this means, a mechanism is available to flexibly optimize a mix of extra constraints on demand. For example, the six-dimensional inverse Puma kinematics can be handled by one PSOM in the given workspace. For under-specified positioning tasks the same PSOM can implement several options to flexibly

resolve the redundancies problem.

Despite the fact that the PSOM builds a *global parametric model* of the map, it also bears the aspect of a *local model*, which maps each reference point exactly (without any interferences by other training points, due to the orthogonal set of basis functions).

The PSOM's character of being a local learning method can be gradually enhanced by applying the *"Local-PSOMs"* scheme. The L-PSOM algorithm constructs the constant sized PSOM on a dynamically determined sub-grid and keeps the computational effort constant when the number of training points increases. Our results suggest an excellent cost–benefit relation when using more than four nodes.

A further possibility to improve the mapping accuracy is the use of *"Chebyshev spaced PSOM"*. The C-PSOM exploits the superior approximation capabilities of the Chebyshev polynomials for the design of the internal basis functions. When using four or more nodes per axis, the data sampling and the associated node values are taken according to the distribution of the Chebyshev polynomial's zeros. This imposes no extra effort but offers a significant precision advantage.

A further main concern of this work is how to structure learning systems such that learning can be efficient. Here, we demonstrated a hierarchical approach for context dependent learning. It is motivated by a *decomposition of the learning phase* into two different stages: A longer, initial *"investment learning"* phase "invests" effort in the collection of *expertise* in *prototypical context situations*. In return, in the following *"one-shot adaptation"* stage the system is able to extremely rapidly adapt to a new changing context situation.

While PSOMs are very well suited for this approach, the underlying idea to "compile" the effect of a longer learning phase into a one-step learning architecture is more general and is independent of the PSOMs. The META-BOX controls the parameterization of a set of context specific "skills" which are implemented in a parameterized box - denoted T-BOX. Iterative learning of a new context task is replaced by the *dynamic re-parameterization* through the META-BOX-mapping, dependent on the characterizing observation of the context.

This emphasizes an important point for the construction of more powerful learning systems: in addition to focusing on output value learning,

we should enlarge our view towards *mappings which produce other mappings as their result*. Similarly, this embracing consideration received increasing attention in the realm of functional programming languages.

To implement this approach, we used a hierarchical architecture of mappings, called the *"mixture-of-expertise" architecture*. While in principle various kinds of network types could be used for these mappings, a practically feasible solution must be based on a network type that allows to construct the required basis mappings from a rather small number of training examples. In addition, since we use interpolation in weight/parameter space, similar mappings should give rise to similar weight sets to make interpolation of expertise meaningful.

We illustrated three versions of this approach when the output mapping was a coordinate transformation between the reference frame of the camera and the object centered frame. They differed in the choice of the utilized T-BOX. The results showed that on the T-BOX level the learning PSOM network can fully compete with the dedicated engineering solution, additionally offering multi-way mapping capabilities. At the META-BOX level the PSOM approach is a particularly suitable solution because, first, it requires only a small number of prototypical training situations, and second, the context characterization task can profit from the sensor fusion capabilities of the same PSOM, also called Meta-PSOM.

We also demonstrated the potential of this approach with the task of 2D and 3D visuo-motor mappings, learnable with a single observation after changing the underlying sensorimotor transformation, here e.g. by repositioning the camera, or the pair of individual cameras. After learning by a single observation, the achieved accuracy compares rather well with the direct learning procedure. As more data becomes available, the T-PSOM can be fine-tuned to improve the performance to the level of the directly trained T-PSOM.

The presented arrangement of a basis T-PSOM and two Meta-PSOMs further demonstrates the possibility to split the hierarchical "mixture-of-expertise" architecture into modules for independently changing parameter sets. When the number of involved free context parameters is growing, this factorization is increasingly crucial to keep the number of pre-trained prototype mappings manageable.

The two hierarchical architectures, the "mixture-of-expert" and the introduced "mixture-of-expertise" scheme, complement each other. While

the PSOM as well as the T-BOX/META-BOX approach are very efficient learning modules for the continuous and smooth mapping domain, the "mixture-of-expert" scheme is superior in managing mapping domains which require non-continuous or non-smooth interfaces. As pointed out, the T-BOX-concept is not restricted to a particular network type, and the "mixture-of-expertise" can be considered as a learning module by itself. As a result, the conceptual combination of the presented building blocks opens many interesting possibilities and applications.

Bibliography

Anderson, J. and e. E. Rosenfeld (1988). *Neurocomputing: Foundations of Research*. MIT Press.

Arbib, M. (1995). *The Handbook of Brain Theroy and Neural Networks*. Bradford MIT Press. (ed.).

Atkeson, C. (1992, 10-1990). Memory Based Approaches to Approximating Continous Functions. In M. Casdagli and S. Eubank (Eds.), *Nonlinear Modeling and Forecasting*, pp. 503–521. Addison-Wesley.

Baader, A. (1995). *Ein Umwelterfassungssystem für multisensorielle Montageroboter*. Meß- Steuerungs- und Regeltechnik, Nr. 486. VDI-Verlag Düsseldorf.

Bauer, H.-U. and K. Pawelzik (1991). Quantifying the Neighborhood Preservation of Self-Organizing Feature Maps. *IEEE Transactions on Neural Networks 3*(4), 570–579.

Breimann, L., J. Friedman, R. Olshen, and C. Stone (1984). *Classification and regression trees*. Wadsworth Inc.

Cleveland, W. (1979). Robust locally weighted regression ans smoothing scatter plots. *J. Amer. Statist. Assoc. 74*, 828–836.

Cleveland, W. S. and S. J. Devlin (1988). Locally weighted regression: An approach to regression analysis by local fitting. *J. Amer. Statist. Assoc. 83*, 598–610.

Craven, P. and G. Wahba (1979). Smoothing noisy data with spline functions. Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numer. Math. 31*, 317–403.

Cun, Y. L., J. Denker, and S. Solla (1990). Optimal Brain Damage. In D. Touretzky (Ed.), *NIPS*89*, Volume 2, pp. 598–605. Morgan Kaufmann.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems 2*, 303–314.

Davis, P. (1975). *Interpolation and Approximation*. Dover Pub., New York.

Dücker, C. (1995). Parametrisierte Bewegungsprimitive für ein Roboter–Kraft/Momenten–Sensor Handsystem. Diplomarbeit, Technische Fakultät, Universität Bielefeld.

Fahlman, S. and C. Lebiere (1990). The cascade-correlation learning architecture. In D. Touretzky (Ed.), *NIPS*89*, Volume 2, pp. 524–532. Morgan Kaufmann.

Farmer, J. D. and J. J. Sidorowich (1988, mar). Exploiting Chaos To Predict The Future And Reduce Noise. Tech. Rep. LA-UR-88-901, Los Alamos National Laboratory.

Frean, M. (1990). The upstart algorithm: a Method for constructing and training feedforward neural networks. *Neural Computation 2*, 198–209.

Friedman, J. H. (1991). Multivariate Adaptive Regression Splines. *The Annals of Statistics 19*(1), 1–141. (with discussion).

Fritzke, B. (1991). Let it grow – self-organizing feature maps with problem depended cell structure. In t. Kohonen et al. (Ed.), *Proc. Int. Conf. on Artificial Neural Networks (ICANN-91), Espoo, Finland*, pp. 403–408. North-Holland, Amsterdam.

Fritzke, B. (1995). Incremenal Learning of Local Linear Mappings. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN-95), Paris*, Volume 1, pp. 217–222.

Fu, K., R. Gonzalez, and C. Lee (1987). *Robotics : Control, Sensing, Vision, and Intelligence*. McGraw-Hill.

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

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Hämmerlin, G. and K.-H. Hoffmann (1991). *Numerical Mathematics*. Springer, New York.

Hanson, S. and L. Pratt (1989). A Comparison of Different Biases for Minimal Network Construction with Back-Propagation. In *Advances in Neural Information Processing Systems II*, pp. 177–185. Morgan Kaufman.

Hastie, T. and R. Tibshirani (1991). *Generalized additive models*, Volume 43 of *Monographs on statistics and applied probability*. Chapman and Hall.

Hayward, V. and R. Paul (1986). Robot Manipulator Control under Unix RCCL: A Robot Control "C" Library. *Int. Journal of Robotics Research 5*(4), 94–111.

Hebb, D. O. (1949). *The Organization of Behavior*. Wiley, New York.

Hertz, J., A. Krogh, and R. Palmer (1991). *Introduction to the Theroy of Neural Computation*. SFI Lecture Notes. Addison-Wesley.

Hinton, G. (1986). Learning Distributed Representations of Concepts. In *Proc 8. Ann Conf Cog Sci Soc*, pp. 1–12. Erlbaum, Amherst.

Hirzinger, G., B. Brunner, J. Dietrich, and J. Heindl (1994). ROTEX – the first remotely conrolled robot in space. In *Intern. Conf. on Robotics and Automation (San Diego)*, pp. 2604–2611. IEEE.

Hopfield, J. J. (1982). Neural Networks and Pysical Systems with Emergent Collective Computational Abilities. *Proc Natl Acad Sci USA 79*, 2554–2558.

Hopfield, J. J. (1984). Neurons with Graded Response have Collective Computational Properties Like Those of Two-State Neurons. *Proc Natl Acad Sci USA 81*, 3088–3092.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks 2*, 359–366.

Jockusch, J. (1996). Taktile Sensorik für eine Roboterhand – Ein microcontroller basiertes integriertes Sensorsystem. Diplomarbeit, Technische Fakultät, Universität Bielefeld.

Jockusch, J., J. Walter, and H. Ritter (1997). A Tactile Sensor System for a Three-Fingered Robot Manipulator. In *Proc. Int. Conf. on Robotics and Automation (ICRA-97)*, pp. 3080–3086.

Jockusch, S. (1990). A neural network which adapts its structure to a given set of patterns. In R. Eckmiller, G. Hartmann, and G. Hauske

(Eds.), *Proc. Parallel Processing in Neural Systems and Computers, Düsseldorf*, pp. 169–174. North-Holland, Amsterdam.

Jolliffe, I. (1986). *Principal Component Analysis*. Springer-Verlag, New-York.

Jordan, M. I. and R. A. Jacobs (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation 6*(2), 181–214.

Kawato, M. (1995). Bi-directional Neural Network Architecture in Brain Functions. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN-95), Paris*, Volume 1, pp. 23–30.

Kohonen, T. (1984). *Self-Organization and Associative Memory*. Springer Series in Information Sciences 8. Springer, Heidelberg.

Kohonen, T. (1990). The Self-Organizing Map. In *Proc. IEEE*, Volume 78, pp. 1464–1480.

Kohonen, T. (1995). *Self-Organizing Maps*, Volume 30 of *Springer Series in Information Sciences*. Berlin, Heidelberg: Springer.

Kummert, F., E. Littmann, A. Meyering, S. Posch, H. Ritter, and G. Sagerer (1993a, 15. DAGM-Symposium). A Hybrid Approach to Signal Interpretation Using Neural and Semantic Networks. In S. Pöppl and H. Handel (Eds.), *Mustererkennung 1993*, pp. 245–252. Springer-Verlag, Berlin.

Kummert, F., E. Littmann, A. Meyering, S. Posch, H. Ritter, and G. Sagerer (1993b). Recognition of 3D-Hand Orientation from Monocular Color Images by Neural Semantic Networks. *Pattern Recognition and Image Analysis 3*(3), 311–316.

Kuperstein, M. (1988). Neural Model of Adaptive Hand-Eye Coordination for Single Postures. *Science 239*, 1308–1311.

Littmann, E. (1995). *Strukturierung Neuronaler Netze zwischen Biologie und Anwendung*. Dissertation, Technische Fakultät, Universität Bielefeld.

Littmann, E., A. Dress, and H. Ritter (1996). Visual gesture-based robot guidance with a modular neural system. In *NIPS*95*, pp. 903–909. MIT Press.

Littmann, E., A. Meyering, J. Walter, T. Wengerek, and H. Ritter (1992). Neural Networks for Robotics. In K. Schuster (Ed.), *Applications of Neural Networks*, pp. 79–103. VCH Verlag Weinheim.

Lloyd, J. (1988, January). RCI User's Guide. report CIM-88-22, McGill Reseach Center for Intelligent Maschines, McGill University, Montréal.

Lloyd, J. and V. Hayward (1992, April). Multi-RCCL User's Guide. Technical report, McGill Reseach Center for Intelligent Maschines, McGill University, Montréal.

Lloyd, J. and M. Parker (1990, July). Real Time Control Under Unix for RCCL. In *Robotics and Manufacturing ISRAM'90*, Volume 3, pp. 237–242. ASME Press, New York.

Marquardt, D. W. (1963). *J. Soc Appl. Math. 11*, 431–441.

Martinetz, T., S. Berkovich, and K. Schulten (1993). "Neural-Gas" Network for Vector Quantization and its Application to Time-Series Prediction. *IEEE TNN 4*(4), 558–569.

Martinetz, T. and K. Schulten (1991, June). A "Neural-Gas" Networks Learns Topologies. In *Proc. ICANN, Espoo, Finland*, Volume 1, pp. 747–752.

Mason, M. and J. Salisbury (1985). *Robot hands and the mechanics of manipulation*. MIT Press.

McCulloch, W. and W. Pitts (1943). A Logical Calculus of Ideas Immanent in the Nervous System. *Bulletin of mathematical Biophysics 5*, 115–133.

Menzel, R., K. Woelfl, and F. Pfeiffer (1993, Sept). The Developement of a Hydraulic Hand. In *Proc. of the 2nd Conf. on Mechanotronics and Robotics*, pp. 225–238.

Meyering, A. and H. Ritter (1992). Learning to recognize 3D-Hand Postures from Perspective Pixel Images. In I. Aleksander and J. Taylor (Eds.), *Proc. Int. Conf. on Artificial Neural Networks (ICANN-92)*, Volume 2, pp. 821–824.

Mézard, M. and J.-P. Nadal (1989). Learning in Feedforward Layered Networks: The Tiling Algorithm. *J. of Physics A 22*, 2191–2204.

Miller, G., P. Todd, and S. Hegde (1989). Designing Neural Networks Using Genetic Algorithms. In J. Schaffer (Ed.), *Proc 3rd Int Conf on Genetic Algorithms, Arlington*, pp. 379–384. Morgan Kaufmann.

Minsky, M. and S. Papert (1969). *Perceptrons : an introduction to computational geometry*. MIT Press, Cambridge.

Montana, D. and L. Davis (1989). Training Feedforward Networks Using Genetic Algorithms. In N. Sridharan (Ed.), *Proc 11th Int Joint Conf on Artificial Intelligence, Detroit*, pp. 762–767. Morgan Kaufmann.

Moody, J. and C. Darken (1988). Learning with Localized Receptive Fields. In *Proc. Connectionist Models Summer School*, pp. 133–143. Morgan Kaufman Publishers, San Mateo, CA.

Murphy, R. (1995). Sensor Fusion. See Arbib (1995). (ed.).

Obermayer, K., H. Ritter, and K. Schulten (1990, nov). A principle for the formation of the spatial structure of cortical feature maps. In *Proc. Natl. Acad. Sci., USA Neurobiology*, Volume 87, pp. 8345–8349.

Parker, D. (1985). Learing Logic. Tech. Rep. TR-47, Center for Computational Research in Economics and Mangement Sciencs, MIT.

Paul, R. (1981). *Robot Manipulators: Mathematics, Programming, and Control*. MIT.

Platt, J. (1991). A Resource-Allocating Network for Function Interpolation. *Neural Computation 3*, 213–255.

Poggio, T. and F. Girosi (1990). Networks for Best Approximation and Learning. *Proc. of IEEE 78*, 1481–1497.

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

Press, W., B. Flannery, S. Teukolsky, and W. Vetterling (1988). *Numerical Recipes in C – the Art of Scientific Computing*. Cambridge Univ. Press.

Rankers, S. (1994). Steuerung einer hydraulisch betriebenen Roboterhand unter Echtzeitbedingungen. Diplomarbeit, Technische Fakultät, Universität Bielefeld.

Ritter, H. (1991). Asymptotic Level Density For A Class Of Vector Quantization Processes. *IEEE Trans. Neural Networks 2*, 173–175.

Ritter, H. (1993). Parametrized Self-Organizing Maps. In S. Gielen and B. Kappen (Eds.), *Proc. Int. Conf. on Artificial Neural Networks (ICANN-93), Amsterdam*, pp. 568–575. Springer Verlag, Berlin.

Ritter, H. and T. Kohonen (1989). Self-Organizing Semantic Maps. *Biol. Cybern 61*, 241–254.

Ritter, H., T. Martinetz, and K. Schulten (1992). *Neural Computation and Self-organizing Maps*. Addison Wesley.

Ritter, H. and K. Schulten (1986). Topology Conserving Maps for Learning Motor Tasks. In J. Denker (Ed.), *Neural Networks for Computing*, pp. 376–380. AIP Conf Proc 151, Snowbird, Utah.

Ritter, H. J., T. M. Martinetz, and K. J. Schulten (1989). Topology-Conserving Maps for Learning Visuo-Motor-Coordination. *Neural Networks 2*, 159–168.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan, New York.

Rumelhart, D., G. Hinton, and R. Williams (1986). Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, 2 vols.* MIT Press, Cambridge.

Schaal, S. and C. Atkeson (1994, September). Robot Learning by Nonparametric Regression. In *Intelligent Robots and Systems (IROS), Munich*, pp. 478–485.

Schutter, J. D. (1986). *Compliant Robot Motion: Task Formulation and Control*. Ph. D. thesis, Katholieke Universiteit Leuven, Belgium.

Selle, D. (1995). Realisierung eines Simulationssystems für eine mehrfingerige Roboterhand zur Untersuchung und Verbesserung der Antriebsregelung. Diplomarbeit, Technische Fakultät, Universität Bielefeld.

Stoer, J. and R. Bulirsch (1980). *Introduction to Numerical Analysis*. Springer New York,Heidelberg,Berlin.

Stokbro, K., D. K. Umberger, and J. A. Hertz (1990). Exploiting Neurons with Localized Receptive Fields to Learn Chaos. Tech. Rep. Nordita-90/28 S, Nordisk Institut for Teoretisk Fysik, Danmark.

v.d. Malsburg, C. (1973). Self-organization of orientation sentive cells in the striata cortex. *Kybernetik 14*, 85–100.

v.d. Malsburg, C. and D. Willshaw (1977). How to label Nerve Cells so that they Can Interconnect in an Ordered Fashion. *Proc Natl Acad Sci USA 74*, 5176–5178.

Walter, J. (1991). Visuo-motorische Koordination eines Industrieroboters und Vorhersage chaotischer Zeitserien: Zwei Anwendungen selbstlernender neuronaler Algorithmen. Diplomarbeit, Physik Department, Technische Universität München.

Walter, J. (1997). SORMA: Interoperating Distributed Robotics Hardware. In *Proc. Int. Conf. on Robotics and Automation (ICRA-97)*, pp. 3511–3518.

Walter, J., T. Martinetz, and K. Schulten (1991, June). Industrial Robot Learns Visuo-Motor Coordination by Means of the "Neural-Gas" Network. In *Proc. Int. Conf. Artificial Neural Networks (ICANN), Espoo Finland*, Volume 1, pp. 357–364. Elsevier, New York.

Walter, J. and H. Ritter (1995). Local PSOMs and Chebyshev PSOMs – Improving the Parametrised Self-Organizing Maps. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN-95), Paris*, Volume 1, pp. 95–102.

Walter, J. and H. Ritter (1996a). Associative Completion and Investment Learning using PSOMs. In C. v.d. Malsburg, W. v. Seelen, J. Vorbrüggen, and B. Sendhoff (Eds.), *Artificial Neural Networks – Proc. Int. Conf. ICANN 96*, Lecture Notes in Computer Science 1112, pp. 157–164. Springer.

Walter, J. and H. Ritter (1996b). Investment Learning with Hierarchical PSOM. In D. Touretzky, M. Mozer, and M. Hasselmo (Eds.), *Advances in Neural Information Processing Systems 8 (NIPS*95)*, pp. 570–576. Bradford MIT Press.

Walter, J. and H. Ritter (1996c). The NI Robotics Laboratory. Technical Report SFB360-TR-96-4, TF-AG-NI, Universität Bielefeld, D-33615 Bielefeld.

Walter, J. and H. Ritter (1996d). Rapid Learning with Parametrized Self-Organizing Maps. *Neurocomputing 12*, 131–153.

Walter, J. and H. Ritter (1996e). Service Object Request Management Architecture: SORMA Concepts and Examples. Technical Report SFB360-TR-96-3, Universität Bielefeld, D-33615 Bielefeld.

Walter, J., H. Ritter, and K. Schulten (1990, June). Non-linear Prediction with Self-organizing Maps. In *Int. Joint Conf. on Neural Networks (IJCNN), San Diego, CA*, pp. 587–592.

Walter, J. and K. Schulten (1993). Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an Industrial Robot. *IEEE Transactions in Neural Networks 4*(1), 86–95.

Wan, E. A. (1993). Finite Impulse Response Neural Networks for Autoregressive Time Series Prediction. In A. Weigend and N. Gershenfeld (Eds.), *Time Series Prediction: Forecasting the Future and Understanding the Past*, pp. 195–218. Addison-Wesley.

Wengerek, T. (1995). *Reinforment Lernen in der Robotik*. Dissertation, Technische Fakultät, Universität Bielefeld.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciencs*. Ph. D. thesis, Harvard University.

Widrow, B. and M. E. Hoff (1960). Adaptive switching circuits. In *IRE ESCON Convention Record*, Chapter 10, pp. 123–137. IRC, New York.

Yeung, D.-Y. and G. A. Bekey (1993). On Reducing Learning Time in Context-Dependent Mappings. *IEEE Transaction on Neural Networks 4*(1), 31–42.

Some of the author's publications, including this book, are available online via: *http://www.techfak.uni-bielefeld.de/~walter/*