

*Exploration based on Neural Networks
with Applications in Manipulator Control*



Ján Jockusch

Dipl.-Phys. Ján Jockusch
Universität Bielefeld
Arbeitsgruppe Neuroinformatik
Technische Fakultät

<http://www.techfak.uni-bielefeld.de>
<mailto:jan@techfak.uni-bielefeld.de>

Vollständiger Abdruck der
von der Technischen Fakultät der Universität Bielefeld
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
genehmigten Dissertation

Prüfungsausschuß:
Prof. Dr. Helge Ritter
Prof. Dr. Christopher Brown
Prof. Dr.-Ing. Gerhard Sagerer
Dr. Gunther Heidemann

Die Dissertation wurde am 9. Februar 2000 bei der Universität Bielefeld eingereicht und
durch die Technische Fakultät am 19. Mai 2000 angenommen.

*Exploration based on Neural Networks
with Applications in Manipulator Control*

Dissertation

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften

der Technischen Fakultät der Universität Bielefeld

vorgelegt von Ján Jockusch am 9. Februar 2000

Contents

1	Introduction	1
1.1	Robotics Lab Setup	2
1.2	Human-Machine Interfacing	3
1.3	Exploration and Control	3
1.4	Previous Work on Robotic Control	5
1.5	Introduction to a Novel Approach	6
I	The Controller Architecture	9
2	Manipulator and Sensor Hardware	11
2.1	Mechanical Construction	11
2.2	Basic Sensory Equipment	12
2.3	The Tactile Sensor System	13
2.4	The Fingertip Sensors	16
2.4.1	Construction Principle	17
2.4.2	Amplification Circuitry	19
2.4.3	Experimental Results	20
3	A Layered Controller Architecture	23
3.1	Design Fundamentals	23
3.2	Levels of Timing and Abstraction	25
4	The Controller Layer	29
4.1	The Controller Structure	30
4.2	Mechanisms for Safety and Reliability	33
4.3	Interfacing with Other Processes	34

4.4	Implementation Details	36
4.5	A Visual Controller Interface	37
4.6	Performance Evaluation	40
5	The State Machine Layer	43
5.1	The Programming Principle	43
5.2	States and State Transitions	45
5.3	Implementation Details	46
5.4	An Example State Graph	47
5.5	Interleaving State Machines	48
5.6	Conclusions	50
II	Exploration with Vector Quantization Networks	53
6	Approaching Intertwined Tasks	55
6.1	Introduction	55
6.2	Literal Interpretation of Topological Maps	56
6.3	Evaluation of Existing Models	57
6.4	Critical Aspects in Robotics	60
6.5	Preparation of Input Data	61
6.6	Adaptive Metrics for Input Rescaling	63
6.7	Expansive Adaptation	65
7	The Instantaneous Topological Map	67
7.1	Improving the GNG for Correlated Stimuli	67
7.2	The Instantaneous Topological Map (ITM)	71
7.3	Results	74
7.4	Statistical Distributions	76
7.5	Architectural Comparison	77
7.6	Dimensionality of Input Data	79
7.7	Conclusions	81
8	Path Finding and Obstacle Avoidance with the ITM	83
8.1	Graph Distance Labeling	83
8.2	Trajectory Generation	84
8.3	Experimental Validation	86

9	An Active Exploration Engine	91
9.1	Basic Ingredients	91
9.2	Results	93
10	Discussion and Outlook	95
10.1	Uses Beyond Robotics	95
10.2	Future Perspectives of the Control Architecture	96
10.3	Future Perspectives for the ITM	98
10.4	Closing Remarks	99

List of Figures

1.1	Laboratory Setup	2
1.2	Typical Action Sequence in a Communication Scenario	4
1.3	System Structure Comparison	7
2.1	The TUM Hydraulic Hand	12
2.2	Finger Kinematics	13
2.3	Oil Hydraulics Actuation System	14
2.4	Hardware Setup	15
2.5	First Fingertip Design	16
2.6	New Fingertip Design	18
2.7	Differential Amplification Principle	19
2.8	Tactile Sensor Amplification Circuit	20
2.9	Experimental Setup for Tactile Sensor Evaluation	20
2.10	Oil Pressure vs. Applied Force	21
2.11	Tactile Sensor Readout vs. Applied Force	21
3.1	Layered Architecture Diagram	24
3.2	Time Scales and Abstraction Levels	28
4.1	Controller Layer Diagram	30
4.2	Outlier Filtering	31
4.3	Example Maintenance Procedure	35
4.4	Interfacing Mechanism	36
4.5	Visual Controller Interface	38
4.6	Startup Controller Behavior	40
4.7	Mixed Feedback Controller Response Without Contact	41
4.8	Mixed Feedback Controller Response With Contact	42
4.9	Compliant Control	42

5.1	Example State Machine Code	47
5.2	State Graph for Grasping	48
5.3	Communication with State Machines	49
5.4	Photo Series of a Grasping Action	51
6.1	Interplay of Different Tasks	56
6.2	Topological Warping in SOMs	58
6.3	Mean and Deviation Estimation	64
6.4	Expansive Adaptation	66
7.1	Random Walk Example	69
7.2	Adaptation of a Standard GNG with Correlated Stimuli	70
7.3	Improved GNG Using an Error Threshold	70
7.4	Edge Update in the ITM	72
7.5	Node Update in the ITM	73
7.6	Reference Vector Adaptation in the ITM	74
7.7	Adaptation Phases of the ITM	75
7.8	Error Comparison of Three Network Models	76
7.9	Graph Comparison of Three Network Models	77
7.10	Non-Uniform Stimulus Density	78
8.1	Graph Distance and Path Generation	85
8.2	Cluster Identification	85
8.3	Two-Dimensional Robot Simulation	86
8.4	Trajectory Generation Example	88
8.5	Trajectory Reinforcement Experiment	88
9.1	Active vs. Random Walk Exploration	94

List of Tables

4.1	Controller Parameter Set	38
7.1	Statistical Input Sequence Evaluation	78
7.2	Network Parameterization for the Dimension Test	79
7.3	Dimension Tests on the ITM and the GNG	80

Chapter 1

Introduction

An old dream of cybernetics has motivated the work reported on in this thesis. It is that of the creation of a robotic system which would at first glance be perceived by its human operators as a living being rather than a cold machine. While this may not sound very scientific, it is an idea that can inspire many new constructions and algorithms.

Naturally, one must first stand back from such an idealistic view and contemplate the adequate methods to at least come close to achieving such an outstanding goal. In the course of developing the system presented here, we followed several different trails, but only very few ideas survived the test of time.

That which survived is a novel layered system that performs robust control of a robotic hand, behavior simulation in a state machine of arbitrary complexity, exploration of its surroundings with the aid of a vector quantization network, and path planning and obstacle avoidance with the topological map represented by that network.

The most remarkable features of this system are its flexibility and simplicity. Much care was taken to make it complete in the sense that it can be used without further development efforts, and expandable in the sense that it can function as a basis for further work in this fascinating research area.

The system's overall design is generic and therefore applicable to different situations, far beyond the field of robotics, and the algorithms developed, especially the ITM (Instantaneous Topological Map), have potential applications even reaching into the realm of data mining.

This introductory chapter first presents the laboratory setup which has evolved to accommodate various research efforts in the fields of computer vision, human-machine interfacing and tactile exploration. Later sections cast more light on the research topics that this work focuses on, human-machine interfacing and exploratory control of a robotic manipulator. A subsequent overview of previous related work on robotic control motivates the introduction of a new approach, based on the notion of reactive motion control.

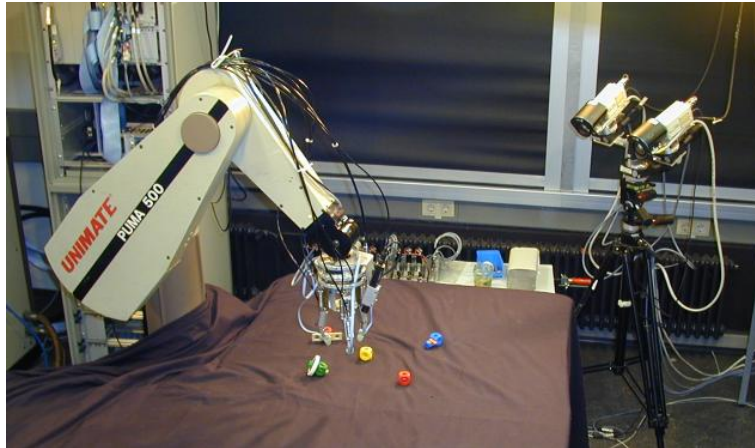


Figure 1.1: **Laboratory Setup:** A standard PUMA robot arm carries a force/torque sensor, a wrist camera, and the hydraulic hand that this work focuses on. The robot is used to manipulate objects lying on the table, possibly in cooperation with a human operator. A binocular camera head provides vision capabilities with depth perception.

1.1 Robotics Lab Setup

Our robotics laboratory offers several research facilities (see figure 1.1). The most prominent is an industry standard robot arm with six degrees of freedom. It is equipped with a wrist force/torque sensor and a wrist camera as feedback sensors for maneuvering the end effector to a designated work area. The end effector is a three-fingered robotic hand fitted with custom-designed fingertip sensors. Throughout this work, this manipulator is used as a testing ground for the new ideas and methods presented.

Several cameras are installed to allow computer vision experimentation. One remarkable camera setup in the lab is the binocular camera head consisting of two cameras with common pan and tilt, and separate vergence control. Extensive research has been carried out with this device concerning the simulation of human visual attention mechanisms [33].

Clearly, this setup lends itself to host different research fields, like gesture and object recognition, robotics and control, visual and tactile exploration, among others. Since the main work areas of our group are neural networks and adaptive systems, we try to look at the aforementioned fields from our own perspective.

As far as computer vision is concerned, a very robust holistic object recognition system has been developed [12], as well as neural networks based camera calibration algorithms, face and eye tracking [35], zero disparity tracking [34], and saccadic scene exploration. All of these research efforts contribute to a human-machine communications system of remarkable complexity [22].

In the field of robotic control, our neuro-informatics perspective means taking a different starting point than most other robotics research, which gives us the opportunity to depart from traditional methods in the hope to reach our ambitious goals.

Especially, the work presented here was originally triggered by the enormous difficulties that classical positional control brought up when applied to our three-fingered manipulator. Replacing this traditional approach completely with a system conforming to strict design goals of simplicity, flexibility, and robustness proved to be remarkably productive.

1.2 Human-Machine Interfacing

In a scenario involving the facilities just described, we wish to study new possibilities for intuitive human-machine interfacing. A human operator talks to the system, describes a task and makes according natural gestures. The system observes the operator and the scene with its binocular camera head and other cameras, and reacts by picking up or pointing at objects using its robotic arm. Together, the human and the machine perform a task, during which the human occasionally teaches the machine and corrects its actions. A preliminary result of these efforts can be appreciated in figure 1.2 on the next page. This subsystem already contains the controller architecture introduced in the present work.

A whole special research project has been devoted to this demanding task, which has inspired many new ideas already. Robotic control, though, especially that of the three-fingered hand, has so far only been performed using positional control and trajectory pre-calculation.

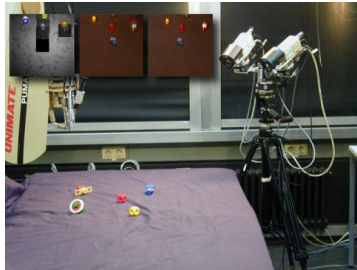
But for our robotic hand this is not a very useful approach. We humans are so sensitive and careful with our own hands, that a numb robotic hand executing planned motions must seem awkward. At the very least, it does not make communicating with it, for example by touching it while in operation, very attractive.

In this work, we will never switch a controller from or to a “compliant mode”. The robotic hand will always be compliant to some extent, and it will always be ready to depart from an intended motion if, for instance, the human interferes to adjust the fingers’ positions.

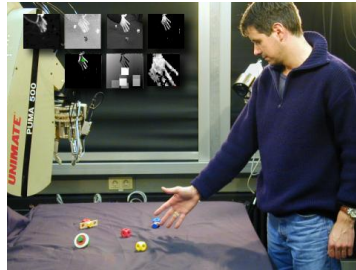
We hope that by making the end effector extremely compliant and sensitive we can achieve a much more natural behavior of the robotic system. One beneficial effect of this approach is that we may learn which behavior patterns humans find natural and intuitively understandable. The other is that a technical device that has so far been viewed as one of the most hazardous in our laboratory may now be used by many researchers with little or no special knowledge.

1.3 Exploration and Control

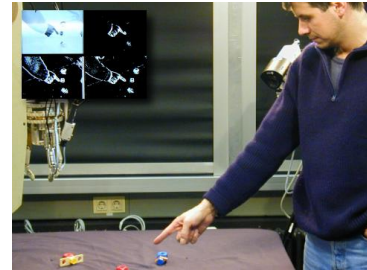
Apart from simulating natural behavior to enhance interaction with humans, we wish a robotic system to explore its surroundings and at the same time learn about its own degrees



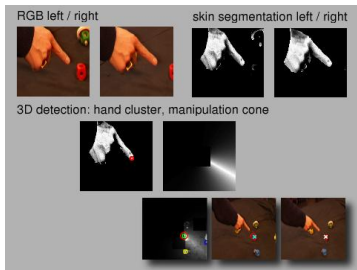
1. *scene exploration*: the insert shows the attention map and the fixation point.



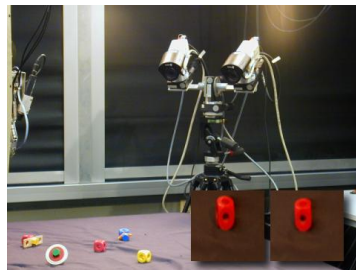
2. *skin and motion detection*: several feature maps are used to focus on moving hands.



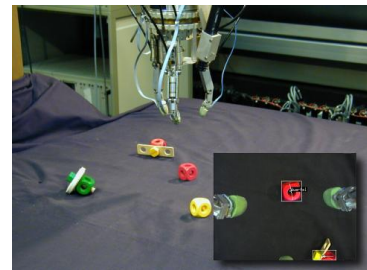
3. *ZDF tracking*: the insert shows the edge detectors and the disparity filter.



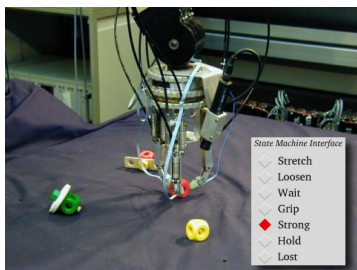
4. *gesture recognition*: the pointer direction defines a manipulation cone.



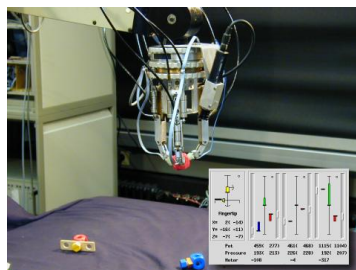
5. *object fixation*: the object's position is estimated by fixating it in the camera foveae.



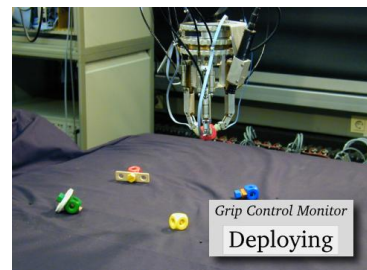
6. *robot positioning*: the wrist camera object recognition prepares a corrective motion.



7. *coordinated grasping*: the arm and hand synchronize via the state machine.



8. *picking up and holding*: the insert shows the controller activity for one finger.



9. *object deployment*: another state machine coordinates the robot arm and the hand.

Figure 1.2: **Typical Action Sequence in a Communication Scenario**: Simulated visual attention, real-time tracking based on zero disparity filtering (ZDF), holistic object recognition, and the control architecture described in this thesis join forces to build a simple human-machine communication system. The operator points at an object in the scene, and the robot hand picks it up and puts it back on the table. This system is the result of the cooperation of many researchers (Robert Rae, Nils Jungclauss, Gunther Heidemann, and Christof Dücker, among others).

of freedom. Like a small child first learning how to move his fingers, then how to grab and manipulate objects, the robotic system should start out with no special knowledge on the geometry and joint limits of its manipulator but instead acquire this knowledge from exploratory motions.

The two major design issues for this project are the data representation, i.e., the memory which stores the knowledge gained from exploring, and the control algorithm for exploration. As far as data representation is concerned, the most fundamental decision is to use vector quantization networks as a storage medium. This enables us to take advantage of many neural network architectures and learning algorithms that have been developed based on this representation.

As far as exploration is concerned, we will first use a simple random walk algorithm to evaluate the performance of different neural network architectures, and develop more intelligent active exploration techniques based only on one special network type.

1.4 Previous Work on Robotic Control

The traditional approach to making a robot move from one position to the other is to calculate a set of points along the desired path in either Cartesian or joint angle space, to smooth this path, e.g. with spline interpolation, and, finally, to move the robot from point to point along this pre-calculated path with a given speed [11].

Since the joint motors can only deliver a finite force, much calculation is necessary to ascertain whether a calculated path can physically be realized. Many different methods are employed to limit joint angles, velocities and torques.

The major focus of this type of robotic control is precision. The properties of robotic manipulators are carefully recorded and simulated in kinematics and dynamics calculations, thus allowing complex motions to be planned and evaluated in advance. This is the approach of choice in most of today's robotics applications, e.g. in assembly lines, where positioning and welding of parts must be achieved with high accuracy.

But adding compliance to motions planned and executed in the way described above is a complex problem. Most approaches involve adjusting the trajectory by small deviations according to sensor signals, which usually does not cover changing the trajectory altogether to steer clear of a sudden obstruction.

Endowing this type of system with the ability to recalculate its planned trajectory if necessary involves much overhead. Additionally, it is difficult to find a formal description for path recalculation simple and predictable enough to be truly useful.

The overall reason for this difficulty in producing a reactive system is the strong focus on motion precision common to most path planning and execution research in robotics. As a result of enforcing a pre-calculated trajectory and formulating compliance as a disturbance to this trajectory, force control (e.g. in force sensor guided motions) is done indirectly through positional control (the trajectory disturbance), which in turn results in force control of the joint motors (the current passing through the coils). Performing force control directly, without an intermediate stage of positional control, may have significant advantages in typical compliance scenarios, most notably grasping of objects.

For a long period of time, traditional positional control was the basis for motion control of our robotic hand. Due to the disadvantages inherent to this mechanism, we implemented a novel control scheme which incorporates reactive changes in behavior much more naturally and easily than would be possible in the former approach.

1.5 Introduction to a Novel Approach

The topic of this thesis is a complete control system based on neural networks which incorporates path planning, obstacle avoidance, and exploration. To implement the neural networks approach, though, extensive infrastructural work had to be done first (see figure 1.3 on the facing page). Therefore, this thesis is divided into two parts, the first one of which capitalizes on the design of the overall control architecture and its lower level components. The second part is devoted exclusively to the neural network layer, which builds on some of the properties of the underlying controller architecture. The system presented in the first part has been especially designed to suit the TUM hydraulic hand [29]. It is a complete and versatile control package for use by other researchers, and it is a novel interactive communication element for use in a larger human-machine interface. The neural networks layer presented in the second part is formulated in a more general way. The emphasis lies on a concise solution to a series of problems usually treated separately, and this approach leads to a new perspective on topographic mapping networks.

The controller architecture must enable us to communicate and work with the robotic system in an efficient and natural way. Therefore, it needs to be highly responsive and sensitive to many forms of feedback. The system layout and the design goals chosen to guide its development are the topics of chapter 3.

To achieve these goals, a new controller was implemented which is based solely on force control with a feedback mixture delivered by several sensors. Positional control can still be programmed as a special case, but doing so is generally discouraged. The controller contains several safety measures and plausibility checks that greatly simplify diagnostics and maintenance of the hardware equipment. This also includes blocking exceedingly jerky motions of the manipulator to ensure that the human operator will not be harmed. The controller layer will be the topic of chapter 4.

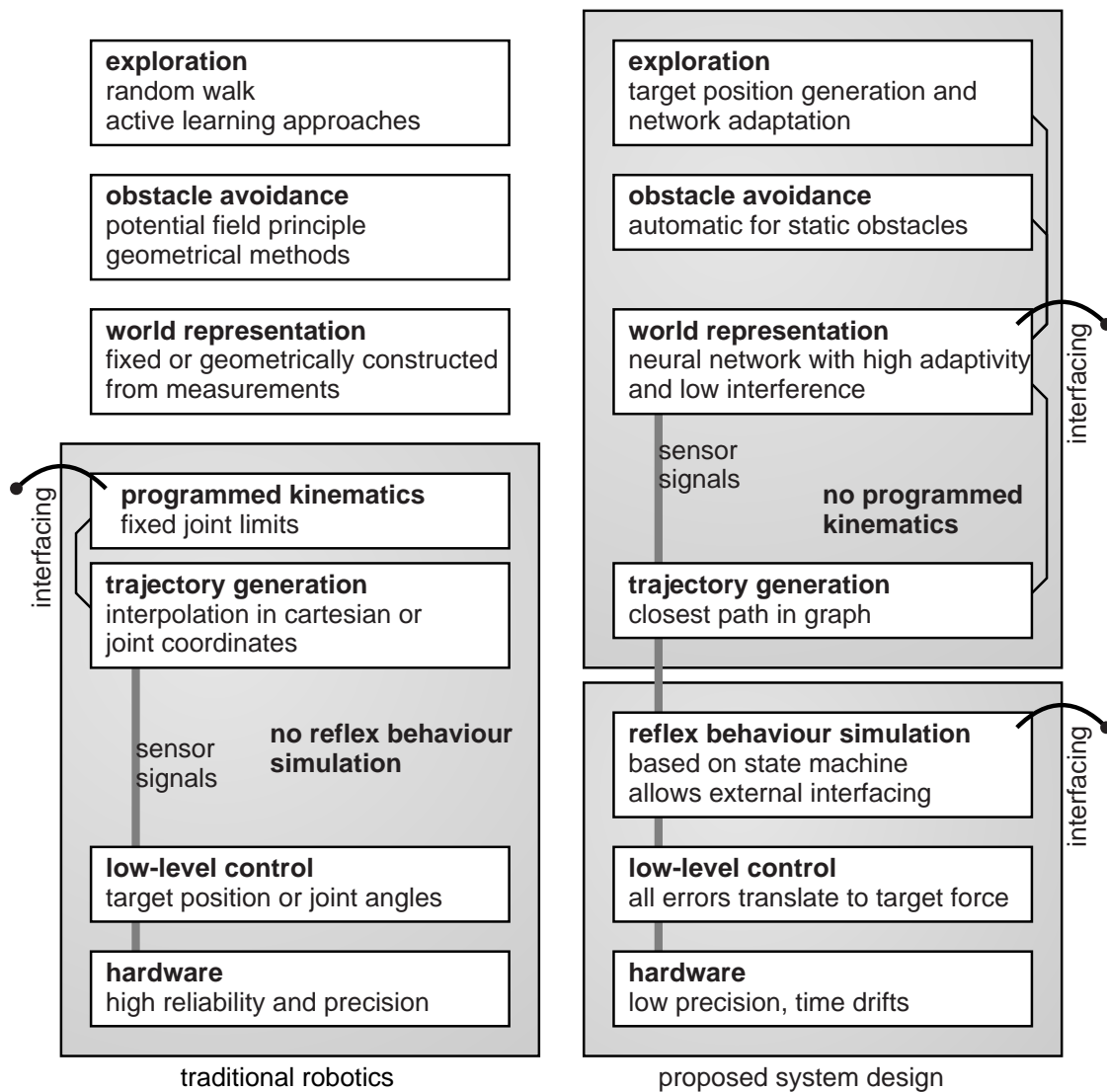


Figure 1.3: **System Structure Comparison:** Traditional robotic systems usually interact with clients through inverse and forward kinematics. This implies the use of interpolation for trajectory generation and the use of precise positional control at the lowest level. Tasks like exploration and obstacle avoidance are considered substantially different and are therefore left for external systems to solve. The approach presented in this thesis assumes a different point of view. Starting out from imperfect hardware which defies positional control, we are forced to find another method of interfacing. Our force controlled system is equipped with a state machine which reacts on sensor patterns and external commands. This behavior simulator, which is missing in standard systems, gives us intuitive access to the manipulator. Still, trajectory generation requires some representation of the manipulator's "world". We turn to a neural network to find that seemingly unrelated tasks can be jointly solved. Because of the abstract world representation, kinematics calculations are unnecessary in this approach. Note that the two systems are not fundamentally different, but only stress different aspects of robotic control. The traditional system needs precision and repeatability of motions, while the proposed system provides permanent interactivity and compliance.

The next step was to implement a “reflex” behavior in the sense that sensory input of a specific pattern would trigger a certain behavior within a short reaction time. This mechanism is not very intelligent, but it makes the system responsive and produces the desired behavior in most situations. A detailed account of the state machine used to realize this idea will be given in chapter 5.

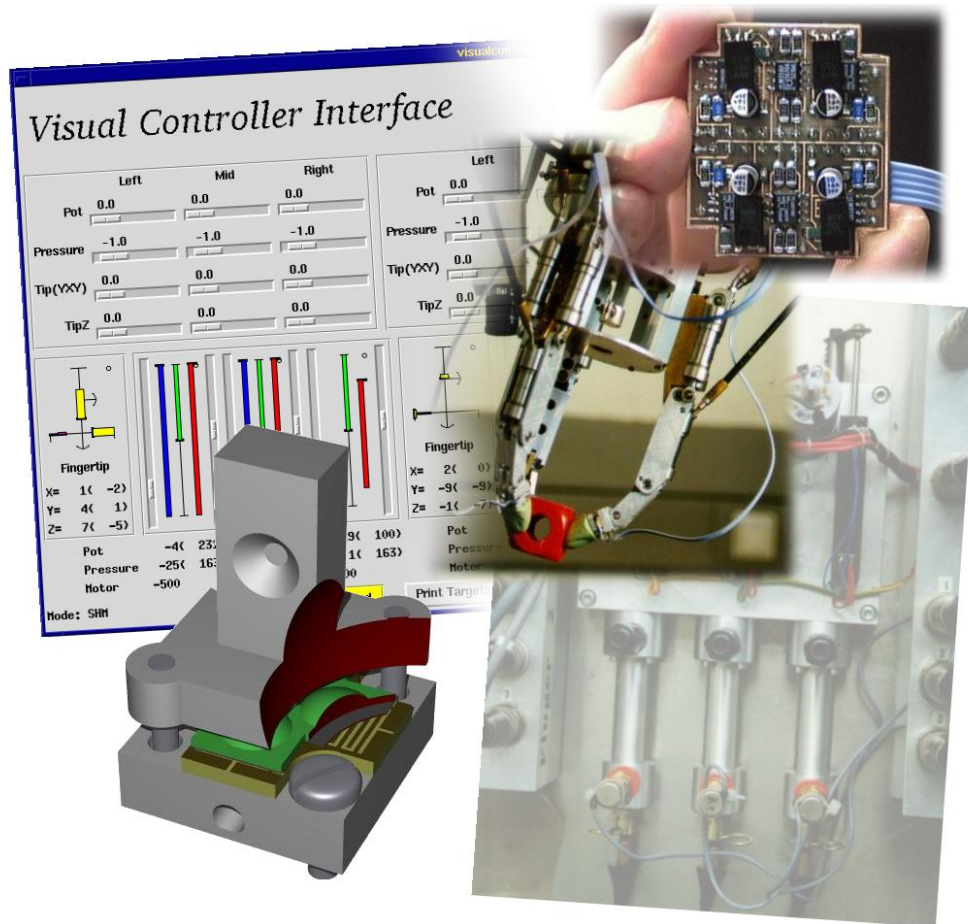
In the second part of the thesis, a topological mapping mechanism is presented which allows the system to gain knowledge about its surroundings. The requirements of this special un-supervised learning scenario, accounted for in chapter 6, lead us to a novel type of vector quantization network, the Instantaneous Topological Map, or ITM for short. Chapter 7 gives a detailed account of the algorithms, possible applications, and some of the more esoteric properties of the ITM.

Using an ITM as a literal road map for a manipulator can easily be achieved with a simple graph distance calculation and trajectory generation routine. We implement this trajectory generation scheme using our state machine formalism and demonstrate its operation in chapter 8.

As the ITM originates from un-supervised learning, a system using such a network as a memory device may passively absorb stimuli from the outside world and use this as its only source of information. But sometimes, active exploration is a more appealing way of acquiring knowledge. With the state machine system and the ITM learning algorithms at hand, we are able to build a simple active scheme for exploring previously uncharted areas, which will be introduced in chapter 9.

At first sight, this design may appear to have two distinctly separate modes of operation, one for reflexes, the other for exploration. But this is not the case. To our knowledge, this is the first system to incorporate reflex-like behavior, path planning, obstacle avoidance, and active exploration in a relatively straightforward way.

Much of the material developed in later chapters builds on the experiences we gathered with one special robotic manipulator in our laboratory. Since this manipulator is also used as a proving ground for many of the algorithms presented, the next chapter will give an introduction and brief analysis of our three-fingered robotic hand.



Part I

The Controller Architecture

Chapter 2

Manipulator and Sensor Hardware

As shown in the introductory chapter, our laboratory can be used as an experimental platform for human-machine interfacing. Among other equipment, it features a robot arm with a wrist force/torque sensor, a wrist camera, and a three-fingered manipulator. In the present chapter, we will concentrate on the manipulator and its sensors, which alone is a nine degrees-of-freedom robotic system.

The mechanics of the hand as well as its sensor electronics will be shown in detail and evaluated to motivate the control methods which are the topic of subsequent chapters.

2.1 Mechanical Construction

The hand consists of a variable number of equal fingers [29]. In our case, three fingers are mounted in an equilateral triangle, pointing in parallel directions (see figure 2.1 on the next page). In contrast to other anthropomorphic robotic hands, the TUM hand uses oil hydraulics to drive the joints, and is therefore remarkably small. Each of its fingers is only about ten percent larger than a human's index finger.

Figure 2.2 on page 13 shows front and side views of one finger, demonstrating the three degrees of freedom and the approximate action radius. Three motor pistons press oil through a long conduit into the finger pistons to move the finger, as sketched in figure 2.3 on page 14.

For direct interaction with a human, the small scaling factor is a strong advantage. The hand is mounted on a robot arm as an end effector, which also contrasts to some other experimental robotic hands which are too bulky to be put on a reasonably sized robot. We are thus in the favorable position to have an arm and hand combination of approximately human size, ideal for human-machine communication.

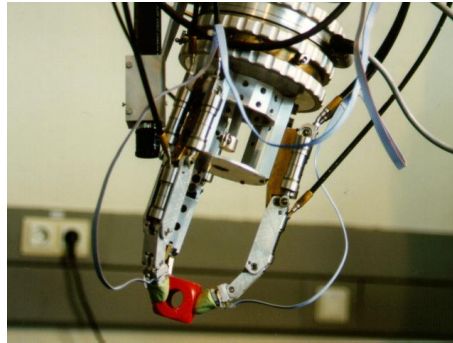


Figure 2.1: **The TUM Hydraulic Hand:** The hand consists of three fingers, arranged in an equilateral triangle. The wrist camera (left) and the cylindrical force-torque sensor (top) deliver additional feedback for the motion of the robot arm.

2.2 *Basic Sensory Equipment*

In terms of sensory feedback, the small size of the fingers disallows the incorporation of position detectors like joint angle encoders. In their basic configuration, the only source of information on the state of the fingers are the oil pressure sensors at the driver pistons and the driver piston position sensors. All sensing is thus done at the driver pistons, which are separated from the joint pistons by long oil conduits. Any feedback pressure exerted at the joints is damped and filtered by friction, the oil's compressibility, and the conduit's elasticity.

Former work has attempted to use only the pressure sensors at the driver pistons for the measurement of external forces at the fingertips. Knowing the oil pressure at the joint and the joint position, the net force at the joint lever can be calculated by subtracting the force exerted by the spring inside the joint piston from the oil pressure times the action area [42].

Sadly, the oil pressure at the joint piston cannot be deduced easily from pressure measurements at the driver piston, because oil compressibility, mechanical hysteresis, and stick-slip effects form a low-pass filter with some additional random or history-dependent components. In an attempt to obtain useful results, a mathematical model of the oil system, which covered the variable compressibility of oil depending on the amount of diluted air and the elasticity of the oil conduit, was created and carefully adjusted to the physical system [42]. But because the model had to be carefully readjusted when oil leaks occurred (which they often did and still do), and the mechanical hysteresis and the statistical stick-slip effects were not yet accounted for, the fingertip force measurements were only possible with huge errors.

The physical limitations of the basic equipment also overshadows the use of the standard positional control algorithms for posture control, like those that were taken from the robot control software RCCL [26]. Without proper joint angle encoders or other similarly precise instruments, attempting to perform reproducible posture control is rather futile.

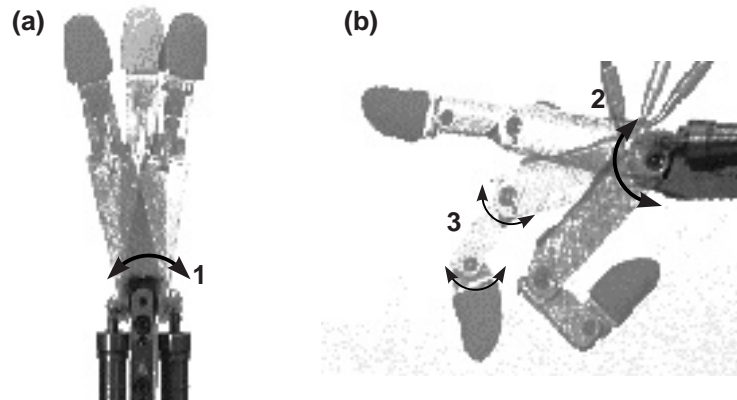


Figure 2.2: **Finger Kinematics:** Each finger is actuated by three pistons, two at the base joint, and one inside the first segment which drives the coupled second and third joints. The left and right pistons bend the finger sideways (arrow **1**) in differential mode, and inward (arrow **2**) in common mode. Together with the flexing motion produced by the middle piston (arrows **3**), the finger thus obtains three degrees of freedom.

The topmost necessity to improve control performance for these hydraulically actuated fingers is to short-circuit the filtering effect of the oil conduit and the mechanical components by placing sensors as close to the action as possible. And because joint angle encoders are exceedingly difficult to construct for the TUM hand, we chose to focus on tactile sensors instead.

2.3 *The Tactile Sensor System*

Several design goals for the construction of a tactile sensor system for the TUM hand have been established. These are high robustness and flexibility, simplicity and ease of use, high performance, and low cost. After a short description of the different components, we will go through these aspects and show how each requirement has been met.

The hardware infrastructure consists of two devices, a multi-channel analog sampler close to the sensors (MASS), and a random-access ring buffer (BRAD) attached to a VME bus (see figure 2.4 on page 15). MASS collects sensor data by sampling up to 64 channels with an amplitude resolution of 8 Bits in turn and immediately transferring the digital data to BRAD via a serial line. BRAD stores the data in a history buffer 127 entries deep, putting timestamps on all entries. A workstation attached to the VME bus may then retrieve entries at random and post-process data as required.

This system has already proven its **robustness**, as it has been continuously running for the last thirty months. The Motorola controllers obviously live up to their excellent reputation, and the peripheral components seem to do equally well. The software on MASS

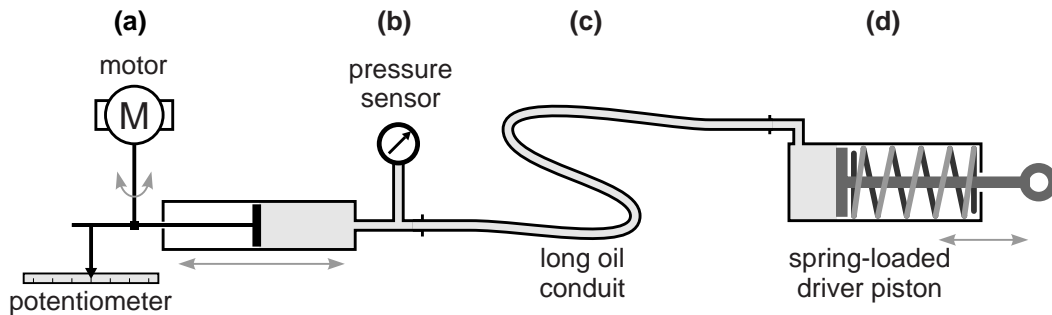


Figure 2.3: **Oil Hydraulics Actuation System:** As a force transmission, a closed hydraulic system connects the motor at the base (a) to the driver piston at the finger (d). The motor piston position sensor and the oil pressure sensor (b) reside at the base, separated from the actor by a long conduit (c), which acts as a complex low-pass filter. The spring-loaded driver piston moves slowly inside its cylinder, which additionally gives rise to frequent sticking-sliding transitions, difficult to predict or even to identify from the motor piston motion and pressure measurement alone.

and BRAD has been extensively optimized and tested. Since information flow is strictly uni-directional, there are no protocols by which either controller can be crashed or halted by the other one or by the workstation. If, for example, MASS crashes or is restarted in the middle of transmitting a packet, BRAD will resynchronize to match the data stream and leave the faulty packet untouched. So, although being tied closely together to ensure fast information transmission, the components of this system operate totally independently of each other.

To allow experimentation with new sensing techniques, **flexibility** is needed. MASS has four slots with connectors for 16 analog channels each. These slots hold amplification and pre-processing circuitry for different sensors. By replacing the amplification modules along with the sensors, a large variety of detectors can be interfaced to the system. Our research work has already seen the benefits of this modular construction. The fingertip sensors now in use are the third generation of tactile sensors which were built and evaluated with this infrastructure.

Using the system is **simple** from the point of view of an applications programmer. Interfacing with BRAD involves nothing but reading the appropriate memory cells of the VME bus. The most recent measurements need not be located in the ringlike history list, a copy is always present at the same location. This allows the user to choose a simple measurement routine if it suffices, or he may write a time series analysis algorithm that uses the history buffer to its full potential.

In terms of **performance**, the system is capable of delivering data at up to 3.6 kHz (four sensors sampled), with a typical value being around 500 Hz. This may seem low compared to other sampling devices, but since the data must also be retrieved and further processed at higher levels, where the usual update frequency is about 100 Hz, the system's performance

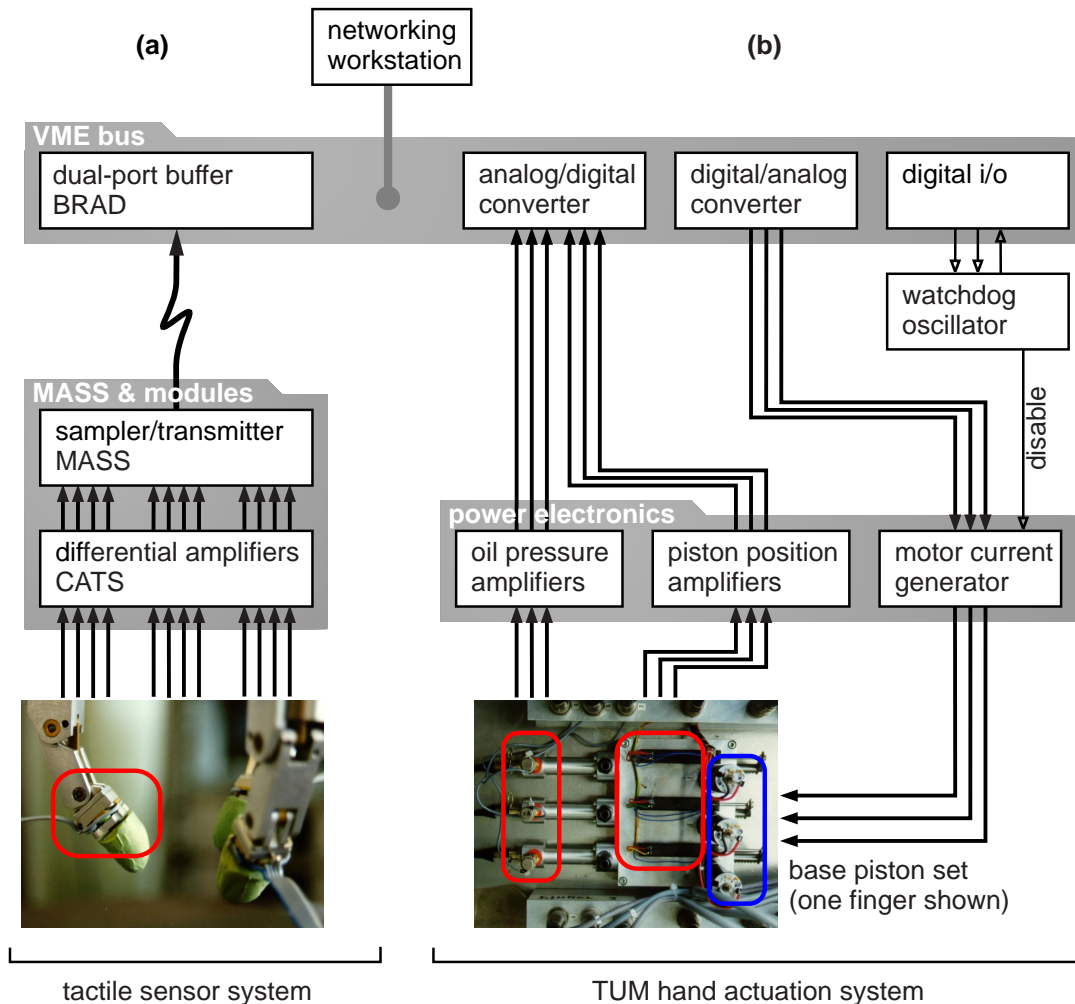


Figure 2.4: **Hardware Setup:** The hardware structure consists of the tactile sensor system developed by the author (a) and the components of the hydraulic hand built by Pfeiffer et.al. at TUM (b). The TUM hand's actoric and sensoric components are all mounted on a base separated from the hand by oil conduits. The motor currents are controlled via the d/a converter board. They are shut down by a watchdog circuit unless the controller program toggles a signal bit regularly. The motors drive pistons whose positions can be determined by reading the voltage of a set of potentiometers attached to an a/d converter. Additionally, the oil pressure at the base is measured by silicon sensors mounted at the far end of the driver pistons. The tactile sensors (a), in contrast, are mounted directly at the fingertips, providing feedback that bridges the filtering effect of the oil and the mechanical structure. The sensors' signals are pre-processed electronically, then sampled and transmitted from the robot arm to a dual-ported buffer which can be read out in the same simple fashion as the other components of the VME bus system.

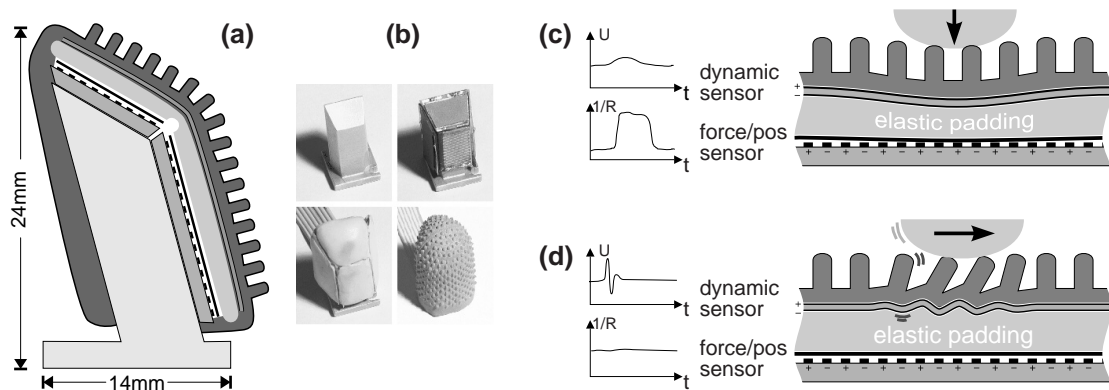


Figure 2.5: **First Fingertip Design:** Here, four FPSR pads were placed on an aluminum body, then covered with an elastomer and a rubber membrane with knobs for slippage sensing. A cut view (a) shows the layer structure, and (b) shows the manufacturing steps. The two sensor types employed in this sensor are able to detect both static pressure (with the FPSR resistors, (c)) and sliding motions (with PVDF foil attached to the membrane, (d)).

is sufficient. In case the detection of short pulses below the millisecond range should become necessary, edge detectors are available which can fill this need.

The maximum delay between sampling and data arrival in the VME memory segment is only about $35 \mu\text{s}$, which is approximately one order of magnitude lower than the smallest possible reciprocal sampling frequency. Therefore, client applications may ignore this latency and treat the sensor values as if they were immediate measurements.

An experimentation platform is often subject to higher-than-average stress, and repairs and changes are costly. We had to construct MASS and BRAD, because similar commercially available products did simply not match our requirements. The **cost** aspect came naturally, because all development had to be done in our lab. The use of simple circuitry and standard components shortens the development time and makes spare parts affordable. In the case of this system, the burden of hardware development turned out to be an advantage [19].

This is particularly true for the sensors themselves, which emerged from several more or less successful development steps. The latest model, which has been in use constantly since about twelve months before the time of writing, has many appealing properties for daily use, like easy servicing and automatic calibration.

2.4 The Fingertip Sensors

Force sensors shaped like fingertips may be easily imagined, but force sensors *the size of* human fingertips pose a difficult problem of miniaturization. Searching the literature for

different sensor materials turned up two kinds of pressure sensitive foil, one piezo-resistive (FSR, [15]), and one piezo-electrical (PVDF, [1]) kind.

These sensor types are prolific among tactile sensing researchers. Some groups also specialize on the miniaturization of strain gauges [5, 4], but many use the FSR sensors in tactile imaging [13, 25], and the PVDF foils for dynamic sensing [47, 3]. A brilliant and unusual approach to tactile sensing based on ultrasound transmission in elastic material has been developed by Shinoda et. al. [45, 44, 43]. Sadly, this ingenious technology is still too difficult to implement in our laboratory.

Initially, the literature suggested that the FSR foil would be useful only for sensing slowly changing pressure profiles, while the PVDF would yield much better response to fast pressure changes [28, 16]. Since both measurements are desirable, a layering technique was first employed which stacked the PVDF sensor on top of an FSR basis (see figure 2.5 on the preceding page). The fast PVDF sensor acts like a microphone membrane responding to the characteristic noise made by moving rubber knobs on the surface, and thus detects sliding motions. The inner layer of FSR sensors locates the center of mass and measures the amount of applied force.

The construction was quite successful, but not durable enough for long term use in the laboratory. The intricate wiring and difficult resistor fabrication for the FPSRs (the position-sensitive variant of the FSR) raised the manufacturing time for one fingertip to over twenty hours. Additionally, after filling in the rubber padding, there were no more possibilities for repair. Several fingertips were ruined by the capillary suction of the FPSR sensors, which flooded the sensitive electrodes with hydraulic oil.

A newer, much more durable design emerged based only on FSR sensors. Surprisingly, our measurements indicate that the frequency range of the FSR is much greater than assumed. It may even suffice to perform successful slippage detection.

2.4.1 Construction Principle

Figure 2.6 on the following page shows the fingertip design currently in use. Each fingertip is equipped with four FSR sensor pads. The electrodes are etched onto one small piece of PCB in a square layout, covered by just one piece of piezo-resistive foil. A rubber pad with four protruding fields ensures an even force distribution from the aluminum stick which presses on the sensors with a pre-loading force determined by tightening the bracket screw. This screw also holds the whole construction together.

Five wires need to be connected to the PCB, one common current source and four resistance measurement wires. A rubber cover gives the fingertip its natural shape and endows it with a small amount of additional compliance.

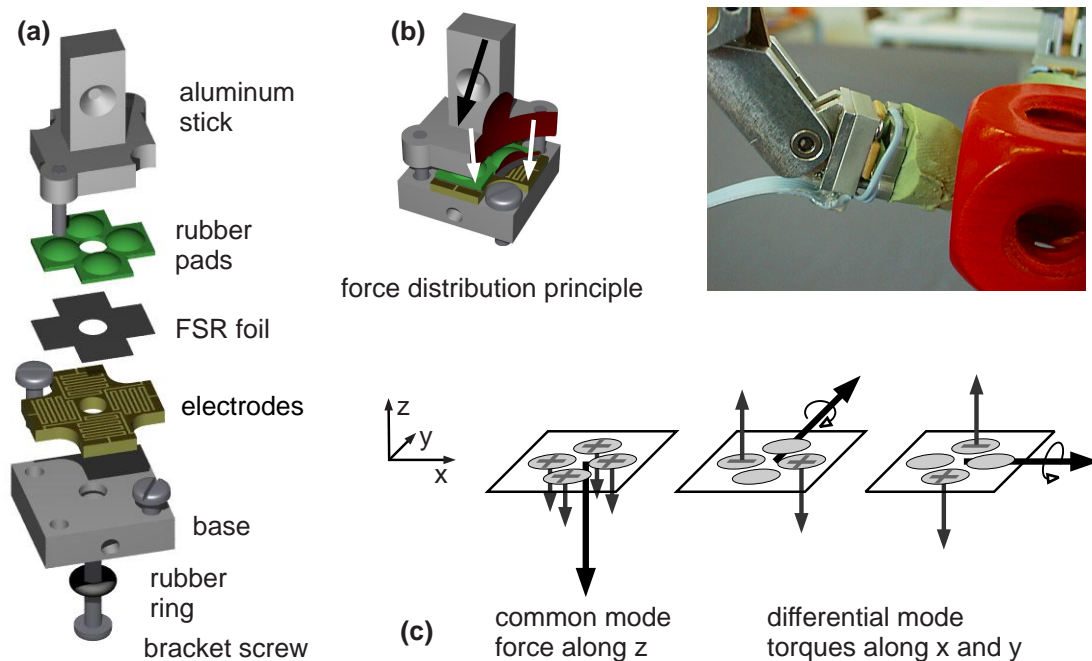


Figure 2.6: **New Fingertip Design:** The exploded view (a) shows the construction of the sensor. It consists of a rigid stick attached to a rigid base by elastic pads, allowing it to move independently by a small amount. This construction distributes an external force to the four pressure sensing fields as shown in (b). Three degrees of freedom can be discriminated in this way (c): one force component, perpendicular to the electrode plane, and two torques along that plane. The normal force results in common sensor responses in all four fields, while the torques result in differential sensor responses in two facing fields.

The rubber pad covering the sensors and rubber rings in the aluminum base mechanically decouple the stick and the base. In this way an additional force in the direction of the bracket screw can be detected in the common mode reaction of all four sensors, while the two torque components perpendicular to the axis of the screw affect only two of the four sensors in differential mode.

Each finger thus delivers four sensors values and can detect a total of three independent external actions: two components of the external torque, and one component of the external force.

The reasons for choosing this construction are mostly practical. Because of the pre-loading of the FSR sensors, the electrodes are in constant contact with the piezo-electric surface, keeping the omnipresent hydraulic oil at bay. Experience shows that this works much better than trying to shield or duct the fingertips to make them oil-proof.

The main reason for *not* choosing this construction would be that the pre-loading of the FSR sensors lessens their sensitivity, since they have logarithmic force response characteristics.

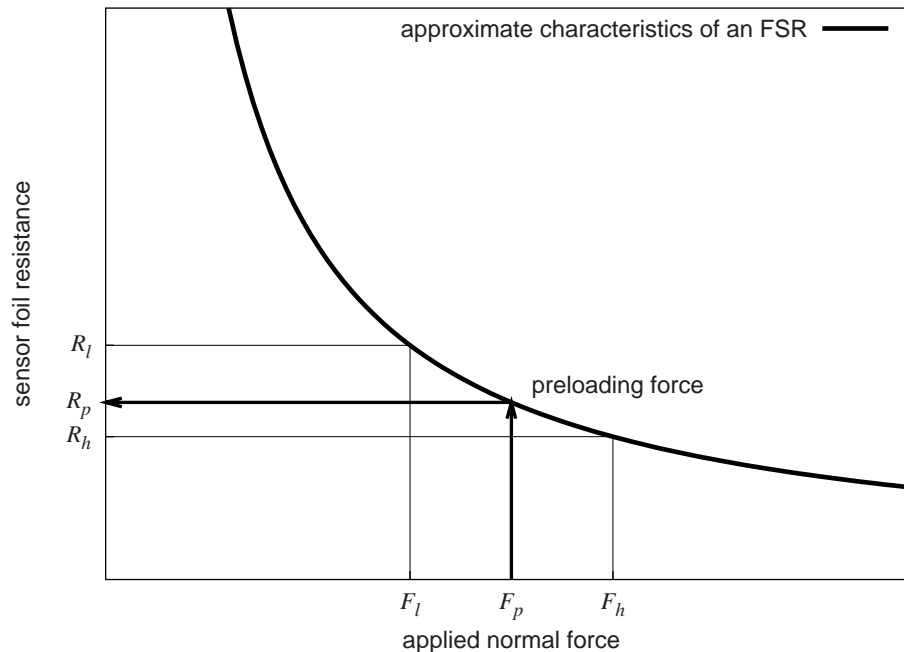


Figure 2.7: **Differential Amplification Principle:** The figure shows the typical characteristics of an FSR foil sensor. Its sensitivity is largest for small forces, but because of the surface properties of the foil the measurements vary considerably. For large forces, the sensitivity of the foil steadily decreases. Using a moderate pre-loading force and a differential high-gain measurement circuit, we can still produce adequate sensitivity with better repeatability, and, additionally, the ability to measure small negative forces as well.

We address this issue by providing special amplification electronics which counteract this effect.

2.4.2 Amplification Circuitry

The resistance of an FSR sensor varies over four orders of magnitude depending on the applied force. Much of this change takes place at low forces, according to the schematic plot in figure 2.7. Measuring forces applied to a pre-loaded sensor can be achieved by centering the amplification around the static resistance. The amplification factor for the differential resistance is then chosen to match the sensitivity and force range requirements.

The Centering Amplifiers for Tactile Sensors (CATS) amplify a small current flowing through the FSR. A low-pass filter with a very long relaxation time of about 5 minutes draws the amount of current representing the static resistance, thereby correctly centering the total amplification, which has a large enough gain to reliably measure forces down to about 50 mN (see figure 2.8 on the following page).

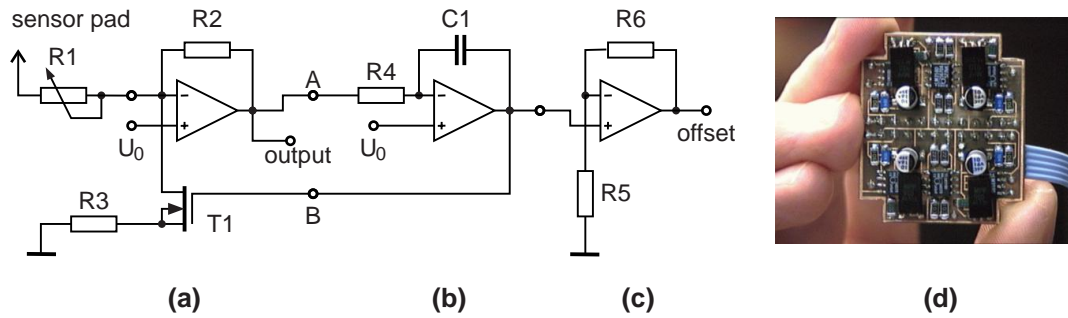


Figure 2.8: **Tactile Sensor Amplification Circuit:** Differential amplification is implemented in a current amplifier (a) with a current drain through T1, which draws the offset current. This offset current is generated by the low-pass filter (b), which accumulates an offset voltage across C1 until the average output voltage at A becomes approximately U_0 . Finally, the offset voltage is amplified into a readable range (c) for additional information. (d) Four such amplifiers are bundled in the PCB for one fingertip sensor.

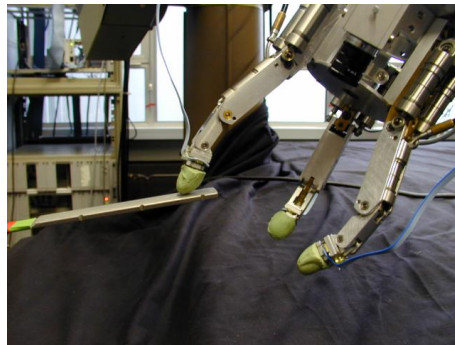


Figure 2.9: **Experimental Setup for Tactile Sensor Evaluation:** A balance with strain gauges is used to measure the applied force at the fingertip while the middle motor current oscillates at 1 Hz. Only the two outer coupled joints move in this experiment.

This measurement technique allows large tolerances for tightening the bracket screw, because each sensor pad locks in to its individual static pre-loading force, and therefore there is no need for further calibration. The static force of each pad can be separately read out by the sensor sampler, albeit with lower precision than the differential measurement.

2.4.3 Experimental Results

In an attempt to find the characteristic frequencies of the oil system and the fingertip sensors, one motor was driven with a sinusoidal signal to periodically bend one of the fingers inward, while the corresponding oil pressure and fingertip response were recorded. Additionally, we used an electronic balance to simultaneously measure the actual force applied

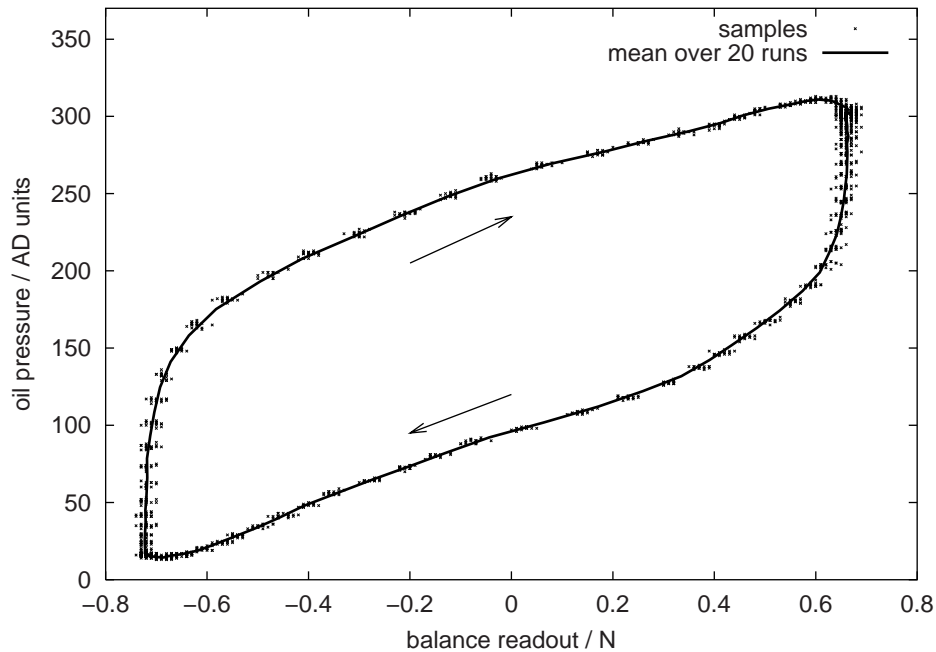


Figure 2.10: **Oil Pressure vs. Applied Force:** During a motion induced by a sinusoidal motor current at 1 Hz, the large hysteresis of the oil hydraulics system becomes apparent in this plot. It shows the oil pressure sensor value while a given force is exerted at the fingertip.

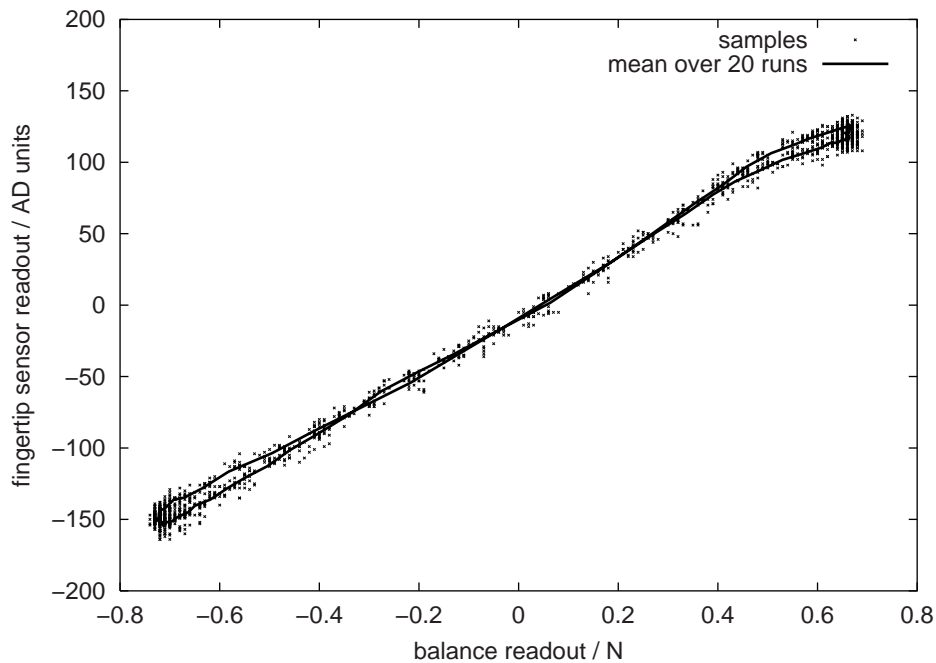


Figure 2.11: **Tactile Sensor Readout vs. Applied Force:** The measured hysteresis of the fingertip sensor during the same motion as in figure 2.10 shows no significant hysteresis. The statistical variations are generally larger than the hysteresis gap.

to the fingertip to ascertain the quality of the FSR sensor and of the amplification circuitry (see figure 2.9 on page 20).

The comparison of the actual force and the oil pressure reveals the drastic filtering effect of the oil conduit and the mechanical components (see figure 2.10 on the preceding page). Due to the highly repetitive motion, the hysteresis plot exhibits relatively small statistical variation. Still, the gap is so large that reliable measurements are impossible if the motion history is unknown.

In contrast, the hysteresis of the fingertip sensor in relation to the force measured by the balance is smaller than the noise level (see figure 2.11 on the page before). This justifies the use of the sensors to bridge the filtering effect of the hydraulic actuation system and build more sensitive control loops.

Because of its realistic dimensions and its high sensitivity, which compares well to the human role model, the fingertip sensor presented in this chapter is very well suited for interaction scenarios. It adds a substantial component to the total sensory feedback of the low-level controller, which will be described in chapter 4 on page 29. Nonetheless, one fundamental detail should be pointed out already. It is technically very difficult to add joint angle sensors to the TUM hand, which has led us to focus on force sensors instead. Therefore, control will take place in terms of forces, in contrast to most robotics applications, which are based on positional control modulated with force feedback.

These two concepts may seem similar at this point, but they have very different consequences and ramifications. This will become clear as we further develop the controlling system for the robotics hardware just described. The following chapter will reveal the overall concept of this control architecture.

Chapter 3

A Layered Controller Architecture

The controller mechanisms we create for the TUM hand shall perform several tasks which are usually treated separately. These are behavior simulation, exploration, path planning, and obstacle avoidance. These tasks are to be solved with one common architecture consisting of few, well-defined elements with narrow interfaces.

In this chapter, we motivate the choice of these elements and show how their interaction is organized. An analysis of the time scale domain of each layer pairs up with a comparison of the scope of each layer's action and error recovery responsibility. The resulting system is modular and expandable, and still simple enough to make both using and maintaining it as easy as possible.

There are only few previously reported control architectures comparable to the one presented here. Fagg et. al. present a promising biologically motivated low-level control approach which already takes into account that a higher level layer might supply via points for the generation of trajectories [6]. The resulting system would be similar to the architecture presented here. The account of a remarkable control infrastructure for a robotic hand, which encompasses stable grip control and hand-arm coordination, but does not touch the exploration and obstacle avoidance topics, can be found in [48].

3.1 Design Fundamentals

Figure 3.1 on the next page shows the overall system layout for the present work. The detailed descriptions of the ingredients follow this illustration from the bottom to the top.

The most fundamental task a low-level controller can perform is to zero a given error function by building an output signal which will reach the target position in as short a time

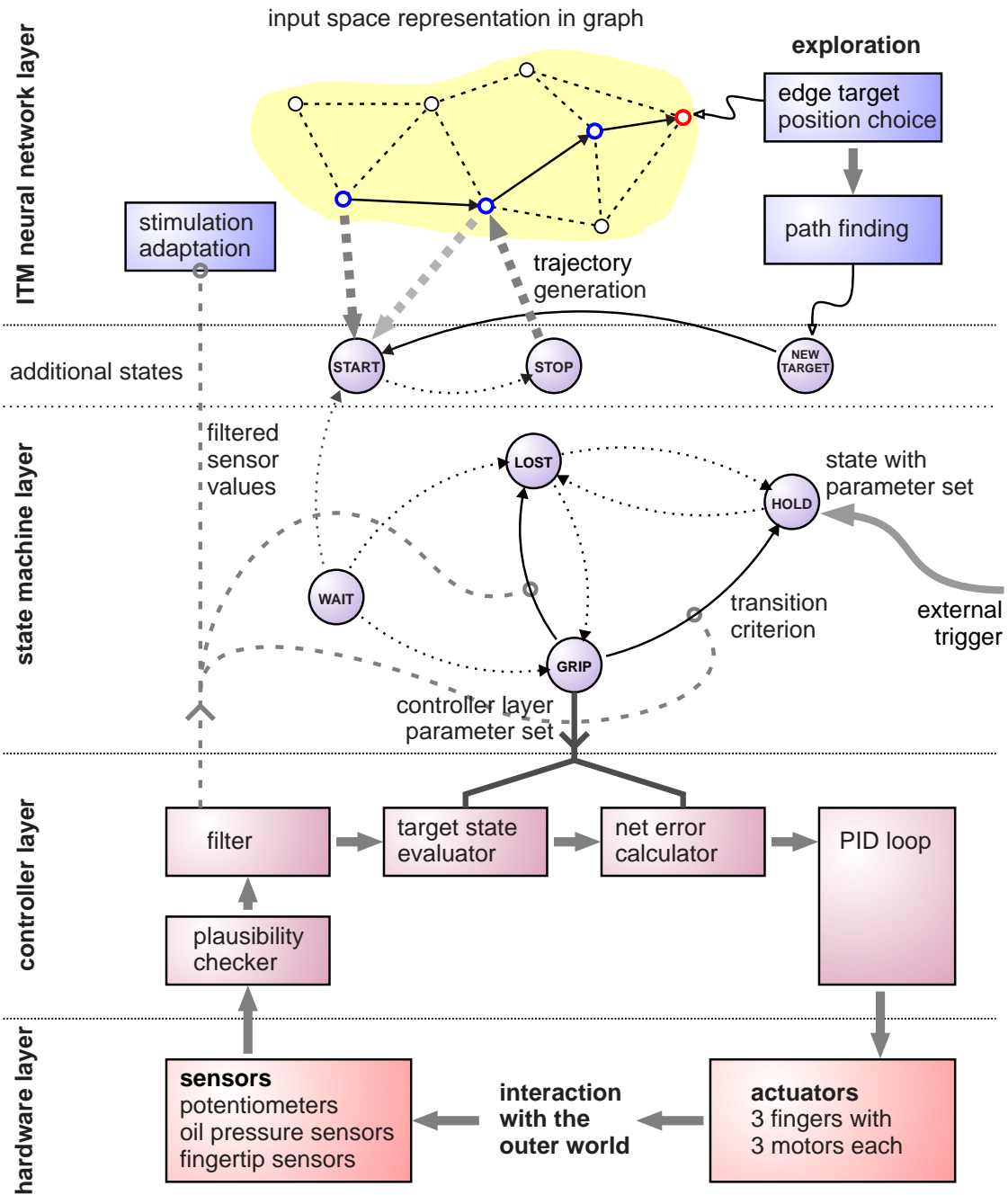


Figure 3.1: **Layered Architecture Diagram:** The hardware and the controller layer form a closed control loop which is influenced and reacted upon by the state machine. This layer implements both an autonomous behavior simulator and a universal interfacing engine. Its expandability allows higher-level layers, like the ITM neural network layer, to integrate their algorithms into the state machine by contributing new states and state transitions.

and with as few oscillations as possible. This is a well-understood problem which can often be solved with carefully parameterized standard control algorithms, most notably PID (proportional/integral/differential) control.

The error function is simple in most cases, usually the joint position error, but in our robotics setup feedback comes from many sensors (driver piston position, oil pressure, fingertips), each of which becomes more or less relevant in different situations.

Because of this, the parameterization of the low-level controller can easily become cumbersome. The target position no longer suffices to define the controller's behavior, the weight of each component needs to be specified, too.

A further simplification of the parameterization task can be achieved with templates, and switching from one template to another can be achieved with sensor-driven criteria. This motivates the addition of a state machine layer, where each state is represented by a set of controller parameters, and a set of criteria that indicate the conditions under which to switch to a new state.

This constellation is not unlike separating a spline-based trajectory generator from the underlying controller, but it is slightly more general. Here, we can choose both the target sensor value constellation *and* the corresponding amount of compliance.

Client applications use labels attached to the individual states to force certain behavior patterns. Nevertheless, the machine will still switch from state to state automatically due to sensory feedback. This produces quite an unusual programming paradigm. There is no clear distinction between input channels and output channels. The current state can both be set to trigger an action, or queried to find out the result of the triggered behavior.

Additionally, we gain an open architecture which allows the implementation of higher levels of abstraction, like path planning or active exploration. These interface to the state machine, allowing them to run concurrently with client programs. For example, the state machine may trigger active exploration via an idle timeout. If a client issues another command by forcing a state change, the exploration is temporarily abandoned.

3.2 *Levels of Timing and Abstraction*

Although dividing the system into layers in the manner shown may seem artificial at first glance, the construction follows a hierarchical design in several aspects. The two foremost of these are the separation of time scales, and the separation of different levels of abstraction. Notably, the same type of time scale separation has also been shown to be present in the perception-action cycle in humans [32, 2, 37].

The number of layers itself affects the number of distinct time scales that are present. Special attention was given to choosing the timing properties of each layer so that they blend harmoniously into the time environment of the sensors, actors, and not least, the human operator.

An additional benefit of splitting the system in this way is an isolation of responsibilities and of error recovery mechanisms. In the following, we will characterize the different elements along these lines (see figure 3.2 on page 28).

The hardware interfacing layer is the **controller**, which is responsible for reading the sensors, for pre-processing or filtering these signals, for performing plausibility and hardware protection checks, and, finally, for delivering motor current output signals to reach a given target position with specified weighting of the feedback components. The controller must be stable under all circumstances, so that higher level engines need no special measures to avoid spontaneous oscillations of the motor currents.

The controller runs at 100 Hz, which is still much faster than the hydraulics actuation system can react. The reaction times of all sensors included so far are shorter than the controller's, with the exception of the oil pressure sensors, which are adversely affected by the hydraulic system's filtering effect.

Other hardware dependent time scales, most notably the typical self-calibration time, do not have as much relevance for the controller process itself. They are included in the graph because there are code segments in the controller layer which compensate for oil leakage, for example.

The **state machine** runs roughly one order of magnitude slower than the controller. This ensures that its dynamics are well decoupled from the dynamics of the controller layer, so that oscillations between the two should not happen. Since the actuator's reaction time is still slightly longer than the state machine's loop duration, the reactivity of the total system is not significantly deteriorated by the seemingly low update frequency of about 10 Hz.

The foremost task of the state machine is managing controller templates and changing the controller settings according to a set of criteria. A template along with these criteria is denominated a "state". The set of states creates a behavior pattern which handles a number of fixed reflexes and fallback action patterns. The states are labeled to provide an abstract interface in which client applications or users trigger an action by naming the appropriate state.

The set of states can be dynamically expanded by layers in higher levels, which can thus add to the set of possible actions or reactions of the system without affecting the behavior already implemented.

All higher-level layers should run in time scales of at least one order of magnitude greater than that of the state machine, because otherwise spontaneous oscillations caused by the

immediate coupling could confuse the system. This limitation applies only if state changes are regularly triggered by higher-level layers *and* the state machine itself. In most cases, however, state changes issued by either party are sporadic, and the consideration stated above becomes less critical.

The **path planning** layer is one such high-level engine. Its task is to watch the motion and sensor feedback pattern and use it to build a representation of the manipulator's properties and its surroundings. It should be able to reproduce a given motion pattern and to find new paths, or shortcuts, from one manipulator state to the other. The representation of the manipulator state space and its surroundings is implemented in a vector quantization neural network, which addresses the special requirements of this problem with a very fast training algorithm, efficient graph generation methods, and the ability to accommodate external graph modifications.

The network is trained with a feature vector composed of the sensor signals from the controller and other state encoding data. This data is not statistically distributed, but has a trajectory-like structure. Therefore, special measures must be taken to avoid destructive interference during the learning process. The path planning layer drives the state machine when triggered to find a path to a given node. In that case, it uses a path finding algorithm on its internal map of the surroundings, feeding the state machine with successive intermediate stages to reach the given target.

The characteristic frequency of the path finding mechanism varies considerably, but typical values are about one order of magnitude lower than the sensor probing loop inside the state machine. Interference of these two layers is thus unlikely, and even if it happens it will only abort the path finding process, instead of producing spurious oscillations.

Finally, the **target chooser** layer enables the total system to actively explore the manipulator's workspace. It has access to the current map of the surroundings and tentatively invents a new node which it then attempts to reach with the aid of the path planner. The target chooser is activated externally by the usual interfacing method of the state machine, or automatically after a given idle time. The target chooser observes the performance of the path finder and forces graph alterations to the map to accommodate the newly acquired knowledge.

The higher level layers can be thought of as small additional parts of the state machine's transition graph. Because they add criteria to the current states, their logic can be triggered automatically, and because they provide fallback state switches, they can automatically abort their actions if anything unexpected happens.

External clients can always query and modify the current state, as has been explained above. These clients need not be human operators. Other programs may use this interface just as well.

In the following chapters, we will examine the two basic layers, the controller and the state machine, in more detail. Together, they endow the robot hand with elementary reflex behavior along fixed programmed patterns.

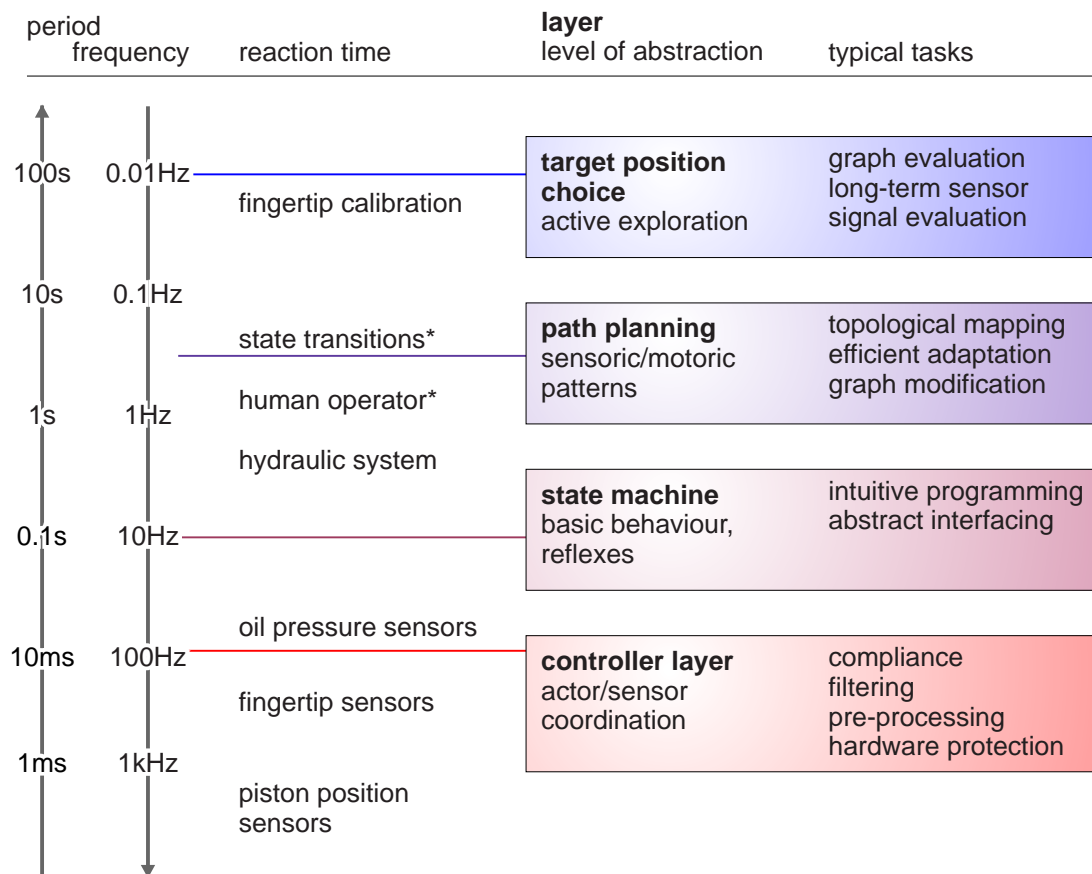


Figure 3.2: **Time Scales and Abstraction Levels:** The choice of the segmentation into layers is motivated by distinct tasks that correspond to different time scales. The controller layer is placed close to the typical reaction time of its associated sensors. To provide natural interfacing for the human, the state machine has to be placed in the 10 Hz domain, which is still faster than the hydraulic system can react. All further layers are not strictly tied to external time scales, but they are separated from each other by approximately one order of magnitude, to decouple them and thus prevent spontaneous oscillations.

Chapter 4

The Controller Layer

To maneuver the robotic hand introduced in chapter 2, a controller process is used which operates independently from all other related layers. From a technical viewpoint, the only difference of this controller when compared to other robotic systems' low-level controller, is that it uses direct force control. As stated in the chapter on the sensory equipment of our laboratory, one reason for this is the absence of reliable positional sensors. The main reason, though, is that we aim at performing intelligent compliant grasping of objects. A logical consequence is to leave the problem of reproducing positions to a higher-level process and perform reliable force control at the low level. Other works, centered around controlling flexible manipulators or adaptively generating gaits for walking machines [46, 14, 20], indicate that unreliable positional control properties can be compensated with reliable adaptive force control, which encourages this approach.

The controller layer is the backbone of the whole system introduced in the previous chapter. It interfaces directly with the motors and sensors and is thus responsible of more than just reaching a defined target position in as short a time as possible.

It has to provide diagnostics and failsafe procedures to protect both the machinery and the human operator from the consequences of erroneous sensor feedback, cable breakage, oil leaks, and so forth.

It must take into consideration that the output it gives to the hardware may itself be lost, either because of broken parts or because an operator switched the motors off.

It must even expect to be temporarily stopped itself, because of other processes taking up CPU time, and be prepared to compensate for such timing glitches.

These few examples clarify the essence of the problem: we want a controller process that can safely run at all times and which is watchful enough to prevent harm to the users or to the machines. For example, expecting the operator to change the controller into a special

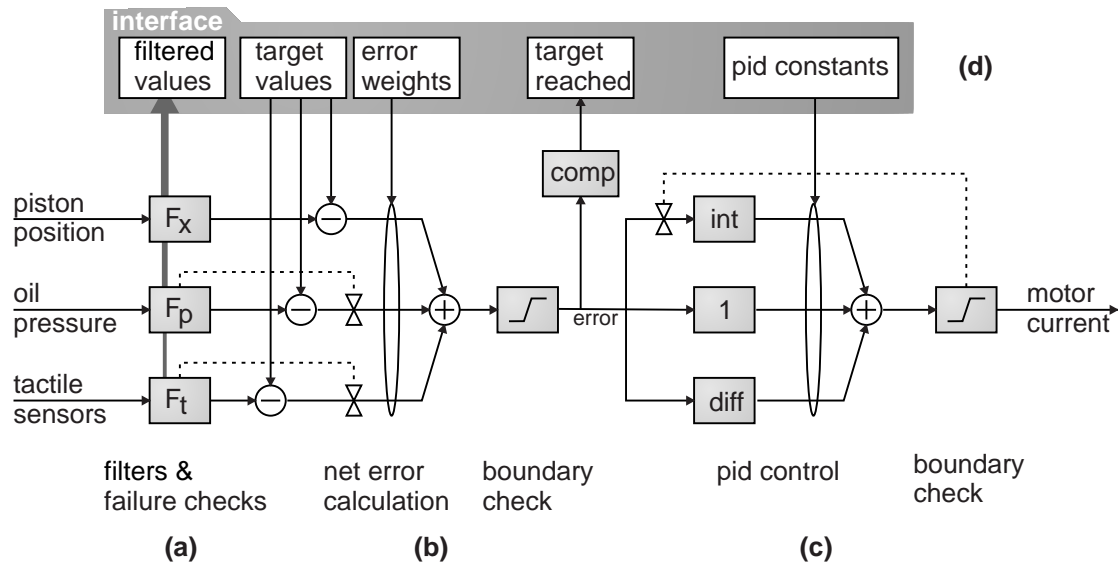


Figure 4.1: **Controller Layer Diagram:** In each iteration of the controller process, (a) the sensor signals are first pre-processed and checked for plausible signals, and (b) the individual errors (possibly partly blocked by the failure checkers) are calculated and superimposed with a given weight vector. (c) A PID controller provides the motor current, which is first passed through a limiter that may also block the integrator. (d) External processes may influence the controller’s behavior by supplying target values and error weights. The convergence of the controller can be rudimentarily queried with a “target reached” signal. The pre-processed sensor signals are made available to other applications as well.

“maintenance mode” before being able to work on the hydraulics is a source of potential problems, because the human might forget to follow this regulation. Therefore, one major design aim while producing the controller program was to reduce the number of necessary steps in maintenance and in normal operation to an absolute minimum.

This chapter first gives a structural overview of the controller and explains its basic operation. It then introduces the many small safety mechanisms that make the controller remarkably reliable and presents a small maintenance operation as an example. The next section explains the interfaces with other processes and shows the detailed structure of the shared data. Finally, after treating miscellaneous topics like the choice of programming language and operating system, an example application is shown which visualizes the controller’s operation and allows interactive parameterization.

4.1 The Controller Structure

The controller as shown in figure 4.1 consists of three sections, sensor signal preprocessing, error superposition, and output calculation, which are called in turn once every 100th

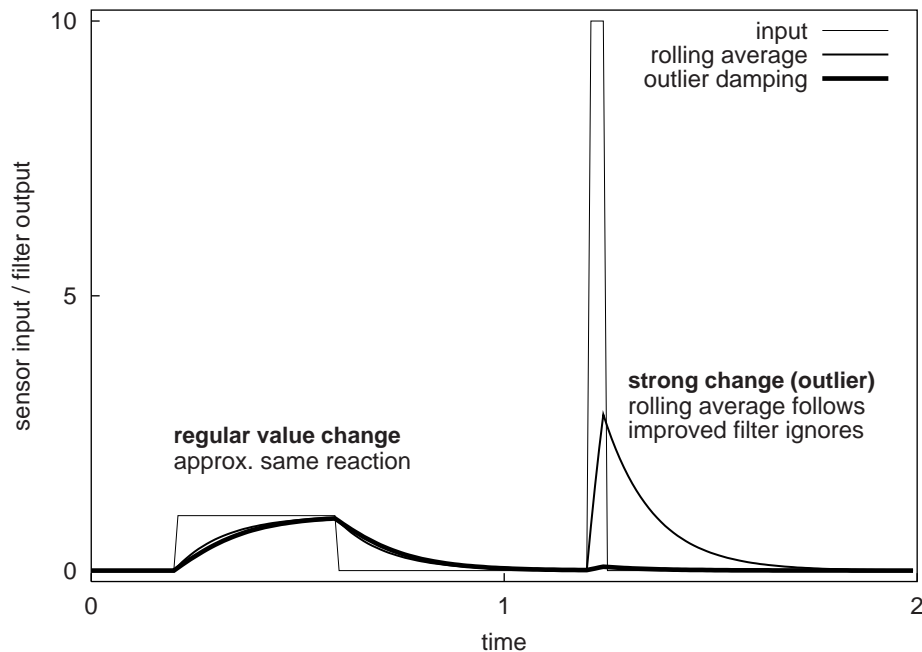


Figure 4.2: **Outlier Filtering:** The addition of a damping term to the rolling average formula yields much better rejection of outliers. This is due to the statistical distribution underlying the two filters. While the rolling average filter is based on Gaussian statistics, the damping term formula emerges from a Cauchy distribution, which has larger “tails” toward positive and negative infinity. The damping term therefore considers outliers more probable and lessens their influence on the current filter output.

of a second. One such processing chain exists for each of the nine motors of the TUM hand. As shown before, the seemingly low frequency is in fact well adapted to the dynamics of the hydraulic actuation system to be controlled. We can even take advantage of the low-pass filtering effect of this system to make the controller resistant to small timing glitches.

During the first stage, the latest valid sensor values are retrieved. In the case of the potentiometers the value is reliable and far less than 1 ms old, so there is no need for filtering. The only preparation needed is an offset equalization, because the potentiometer readout is subject to drifting due to oil leaks. Still, the sensor’s speed and reliability stand in stark contrast to their de-facto unreliability for positional control, as explained in chapter 2.

The oil pressure sensors are equally frequently sampled, so filtering is unnecessary, but they are much more likely to fail because of cable breakage or membrane destruction. A plausibility check has been introduced here which detects both conditions.

The fingertip sensors are by far the most fragile parts in the ensemble, which makes plausibility checks a must. Because of the constant oil leaks, the sensors need cleaning from time to time; the checks indicate the need for this simple servicing task. The fingertip

sensors are sampled at roughly 600 Hz, which means the samples are at most 1.7 ms old. To reduce the effect of noisy data around the neutral position, the output of these sensors is passed through a low-pass filter with a cut-off frequency of about 50 Hz.

As a filtering algorithm, we use a promising approach by Liano [24], originally intended for improving neural network training algorithms by minimizing the mean log squared error (MLSE) rather than the mean squared error (MSE). Instead of using a standard finite impulse response low-pass filter, we use a slight modification known to provide better outlier rejection. This is achieved with a quadratic damping term in the influence function. The new filter output x_{t+1} is thus calculated from the previous output x_t and the input y as follows:

$$\begin{aligned}\Delta &= y - x_t \\ x_{t+1} &= x_t + \frac{\lambda}{1 + \gamma \Delta^2} \Delta\end{aligned}\tag{4.1}$$

This approach emerges from a mathematical analysis of the formula given a random variable as input. The statistics of the standard version ($\gamma = 0$) follow an assumed Gaussian distribution of the input values, while $\gamma \neq 0$ results in an assumed Cauchy distribution. The consequence is that outliers are considered more likely and thus cannot influence the filter output as much as if underlying Gaussian statistics were expected, as depicted in the small experiment in figure 4.2 on the preceding page.

The current sensor values are subtracted from target values yielding a set of individual errors, which are subsequently combined in a weighted sum and passed through a boundary limiter to produce the net error. The boundary checker has the same beneficial effect on the controller's performance as a sigmoid transfer function has on the performance of a perceptron. It allows high sensitivity of the system in a defined area, while avoiding extreme reactions outside this area.

In a final step, the net error is passed through an integrator and a first derivative calculator to produce PID control inputs. These are superimposed with weight constants K_P , K_I , and K_D to produce the output, which again passes through a boundary limiter to produce the motor current.

The sections overlap to some extent, i.e., the plausibility checks during preprocessing can block error components in the superposition section, and the output boundary checks can block the integrator's operation.

To obtain an approximate calibration of the piston position sensors, an internal minimum value is kept for each potentiometer. These minimal values are hidden from client applications, which can only use the calibrated sensor readings.

The controller additionally provides a flag which it sets if the net error drops below a given threshold. This flag can be used by clients to trigger an action as soon as the controller has

reached its equilibrium. Clients reset this flag upon changing the controller parameterization.

It is a trivial fact that this controller can zero a given mix of errors with the appropriate parameterization. But how does it behave under partial failure conditions? We will discuss this topic in the next section by examining some of the most likely hazards.

4.2 Mechanisms for Safety and Reliability

The controller layer features a few additions which combine to protect against a set of potential dangers. These additions are (i) the filters and plausibility checkers in the pre-processing section, (ii) the boundary limiter for the net error, and (iii) the boundary checker for the motor current output. The controller loop itself runs in real time with a loop frequency of 100 Hz. An overload detector detects and reports timing faults. Some of the most common dangers to smooth control are:

Sensor failure: A broken cable or a defective electronic component can inflict severe damage on the total system, because the net error contains large artifacts that cannot be compensated by the output. Some of the most common sensor failure situations are broken membranes of oil pressure sensors, broken fingertip sensor ribbon cables, and oil-soaked fingertip sensors. Special diagnostic logic has been incorporated into the controller to gracefully react in those situations by switching the appropriate error channel off and notifying client applications.

But it is infeasible to produce detector code that correctly diagnoses all possible kinds of erroneous sensor behavior. The above detectors are only the first stage of protection against feedback failure. If the net error cannot be compensated, the output will grow indefinitely because of the integrator component controlled by K_I . The last boundary checker therefore not only limits the motor current to a safe level, but also stops the integrator from further increasing the accumulated error. The integrator's internal value is left unchanged, though, so that it will smoothly re-enter normal operation once the feedback channels are repaired.

Motor failure: This error condition is similar to the one previously discussed, because the controller cannot zero the net error anymore. The output current limiter shuts down the integrator as described above, and the controller's internal state freezes. If the motor is switched back on, it will initially receive a safe current and the controller will operate normally with no further intervention.

Wrong parameterization: Many internal parameters can be changed from the outside, and some of them can cause undesirable controller behavior. Although there is no

warning against erroneous parameterization, the net error limiter makes the configuration of the error mixture much easier. The motor current limiters provide additional protection against mis-configuration.

Controller oscillations: The controller parameters K_P , K_I , and K_D are not part of the standard client interface structure, because wrong configuration can produce spontaneous oscillations. Instead, these parameters have been preset to a near-optimal setup which cannot easily be made to oscillate. Still, if the external force is varied with a frequency near the controller's characteristic frequency, small damped oscillations can be observed.

Timing glitches: If the controller process itself is temporarily halted or if it crashes and needs to be restarted, the output current values freeze, possibly with high values, which would result in a sudden contraction of the respective finger joints. An electronic watchdog circuit, which has to be disarmed periodically with write operations, switches off all motors if the controller program crashes. Upon restarting, the controller will recover, either with default parameters or with the parameter set left over by the previous instance. The internal error limiter helps to smooth out the initial motion.

Note that in all the situations described above, there is no need to configure or otherwise change the controller logic from the outside. The process continues running under all circumstances, partly shutting down and restarting automatically as the need arises.

This design makes maintenance exceedingly easy, especially compared to the solution previously in use [36]. As an example, consider the operation of refilling one or more oil pistons (see figure 4.3 on the next page).

Formerly, it was necessary to shut down the controller software, fill the oil piston while the motors were disabled, then start a special helper program to reduce the oil volume to a defined value, restart the controller, and go through a series of calibration steps in order to use the potentiometers as replacements for joint angle encoders.

Since we do not use positional control anymore, the last step is unnecessary. The main advantage of the new software system is that it is sufficient to switch off the motor currents with the hardware watchdog blocking switch, refill the oil pistons, and switch the motor currents back on. The controller will recover and re-calibrate automatically.

4.3 *Interfacing with Other Processes*

The controller layer communicates with other processes through a set of parameters which reside in a UNIX shared memory segment. Almost all controller state variables are contained in this segment, which serves as a fast communication channel with client applications (see figure 4.4 on page 36).

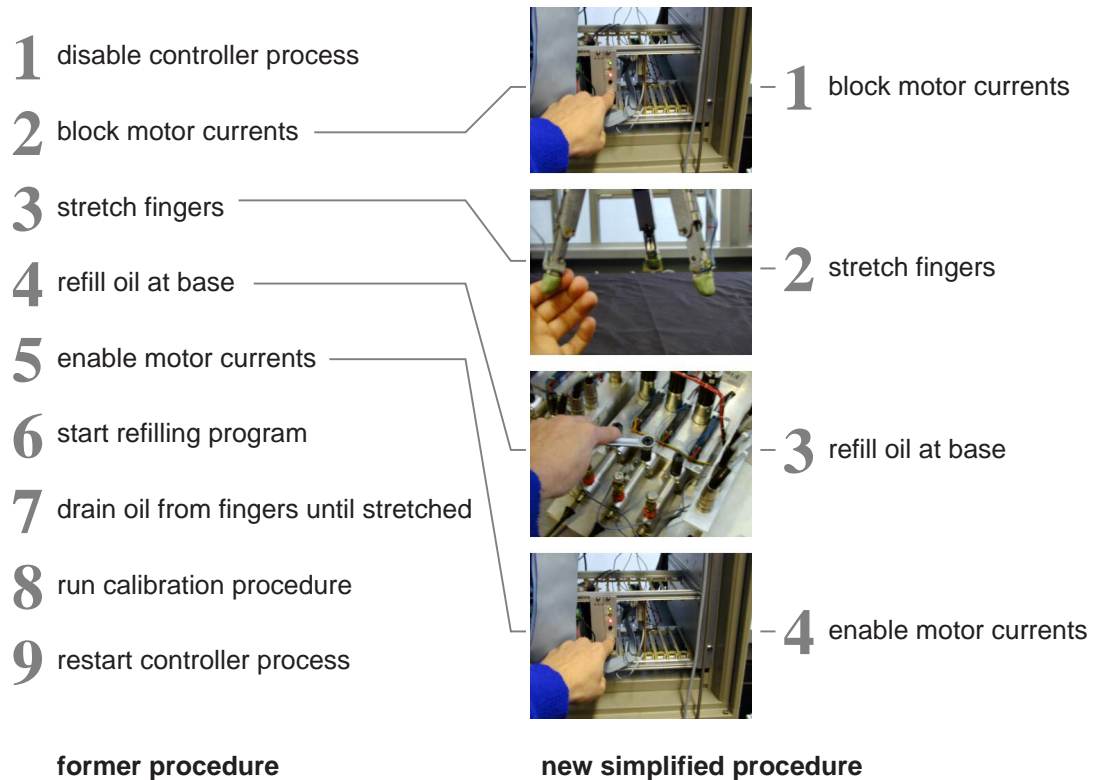


Figure 4.3: **Example Maintenance Procedure:** With the new controller, there is no need to start special software for maintenance operations. Switching the motors off is enough to make the controller degrade into an idle state. The formerly necessary calibration procedure after oil refills is obsolete by an automatic internal calibration which is part of the control loop.

The shared memory segment survives controller process crashes, thus allowing the next controller process to use the same data as the defunct process and use it to take over smoothly.

Because all entries in shared memory are atoms, i.e., small data types which are read and written by the kernel in an uninterruptable operation, there is no need for mutexes¹ or other signaling mechanisms which would potentially enable malignant client applications to block the controller. Instead, the controller process runs completely independently of its clients, which may change parameters at any time.

Therefore, clients have no control over transitional behavior during re-parameterization. But because, as a consequence of the hydraulic system's inertia, the controller generally has loose timing restrictions, such precise control of transitionals is unnecessary.

¹This POSIX synchronization method relies on applications to only allocate resources for short periods of time. Thus, errors in the allocation and release scheme may block the controller from reading its parameter set. Double buffering or other more refined methods were abandoned because the use of atoms achieves the same goal.

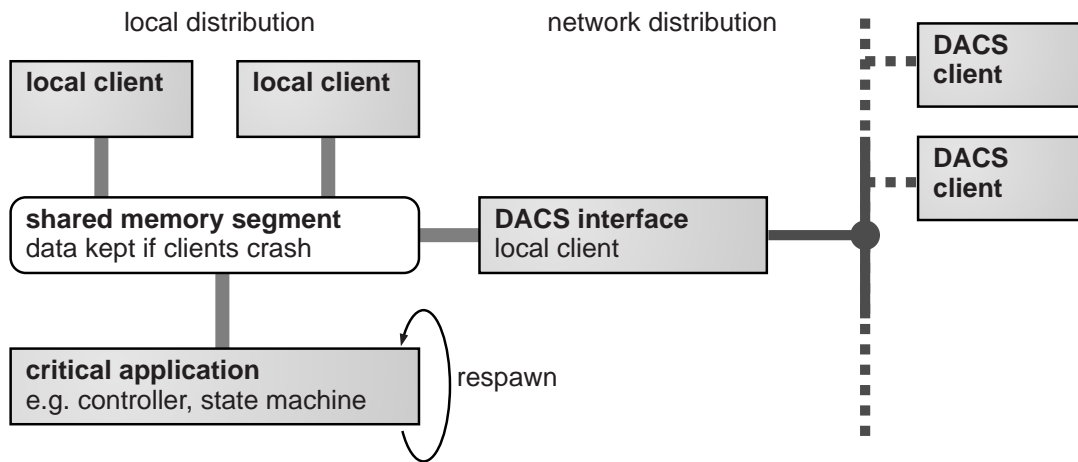


Figure 4.4: **Interfacing Mechanism:** To ensure reliable operation of a critical application (e.g. the controller), it is detached from its communication interfaces. All vital variables are kept in a shared memory segment which survives crashes of both the underlying program and of client applications. The main application is re-spawned automatically, allowing the successor to take over smoothly. Wherever possible, kernel atoms are used for data storage to obsolete the use of mutexes, since they might destroy the decoupling effect of the shared memory segment. One local client implements a DACS communications host, providing clients across the network with access to the current data.

Accessing the parameter set by means of the shared memory segment is naturally only possible on the same host. For access over the network, we rely on the communications tool DACS [21]. One special local client has been implemented which interfaces the shared memory data to a DACS demand stream and a message port. Clients around the network may query the parameter set by tapping the stream, and may feed new parameter sets by sending an appropriate message.

4.4 Implementation Details

The construction of the controller layer made many design decisions necessary, some of which shall be discussed in this section.

Choice of host machine: Former controller software has been implemented on an embedded controller board, while this system runs entirely on a general-purpose workstation. One reason for choosing an embedded controller is reliability of communication channels: embedded controllers should communicate much more intensely with the hardware they are attached to than with the outside world, i.e., the higher-level

client applications. In our case, the advantage of using an embedded controller becomes marginal in this respect, because communication with the hardware and with the higher-level applications are almost equally intense.

Another reason is the added reliability of a dedicated computer for just one task, especially if close attachment to real time is necessary. Luckily, we can be a bit loose about keeping track of time, as shown above. Our control program may share the computer resources with other processes since it will only put a small amount of load on the processor.

The advantages of using a general-purpose machine are many. Development of the software is much easier because debuggers are available and the hardware can be simulated. The same development system can be used for all the layers, which simplifies project management. Finally, standard UNIX system resources such as signals and shared memory are available, making the finished product at least partly portable, although it is specialized for one specific hardware setup.

Choice of programming language: The programming language used for the implementation of the controller is ANSI-C. The graphical user interfaces were written in Tcl/Tk. At the beginning of this project, Java was already in the discussion, but we abandoned it because of its lack of performance and the necessity to build new hardware interfacing components. User interfacing with Tcl/Tk loadable modules is straightforward to write, and the GUIs produced are appealing and responsive. This combination proved so powerful and fast in terms of development time, that we kept it throughout the project.

NEO compatibility: Being the most frequently used and most intensely developed programming tool in our team, NEO deserves special attention. The data structure of the DACS interface has been chosen in such a way that DACS_NEO can access the controller layer easily using the standard units.

One aim of this work is to provide a usable system for communicating with the robotic hand. With respect to client applications, the quality of the communication channel is crucial. The interfaces presented here provide different programming environments with the controller parameter set as shown in table 4.1 on the next page. The visual controller front-end introduced in the next section shows the Tcl/Tk interface, which works both locally, i.e. on the same host, and network-wide, using DACS.

4.5 A Visual Controller Interface

As an example application for experimenting with the controller, we provide an application which visualizes the internal state of the finger controllers. Target values can be changed

per finger (×3):						
<table border="1"> <tr> <td>per piston (×3: L,M,R):</td> </tr> <tr> <td><i>PID controller (K_P, K_I, K_D)</i></td> </tr> <tr> <td><i>feedback mixer (\vec{m})</i></td> </tr> <tr> <td>targets and errors:</td> </tr> <tr> <td>potentiometer (target, error, <i>zero calibration</i>)</td> </tr> <tr> <td>pressure (target, error, <i>zero calibration</i>, failure flag)</td> </tr> </table>	per piston (×3: L,M,R):	<i>PID controller (K_P, K_I, K_D)</i>	<i>feedback mixer (\vec{m})</i>	targets and errors:	potentiometer (target, error, <i>zero calibration</i>)	pressure (target, error, <i>zero calibration</i> , failure flag)
per piston (×3: L,M,R):						
<i>PID controller (K_P, K_I, K_D)</i>						
<i>feedback mixer (\vec{m})</i>						
targets and errors:						
potentiometer (target, error, <i>zero calibration</i>)						
pressure (target, error, <i>zero calibration</i> , failure flag)						
fingertip:						
targets (T_x, T_y, F_z)						
errors (T_x, T_y, F_z)						
<i>z axis zero calibration, failure flag, outlier filter</i>						

target reached flag
motors active flag, *currents attenuation filter*

Table 4.1: **Controller Parameter Set:** All state variables listed are present in the local shared memory segment, although some are hidden by standard application interfaces, e.g. the Tcl and DACS interface used for the controller GUI (see figure 4.5). Among those hidden variables are the PID parameters, the sensor filters, and the zero calibration entries (shown in slanted print).

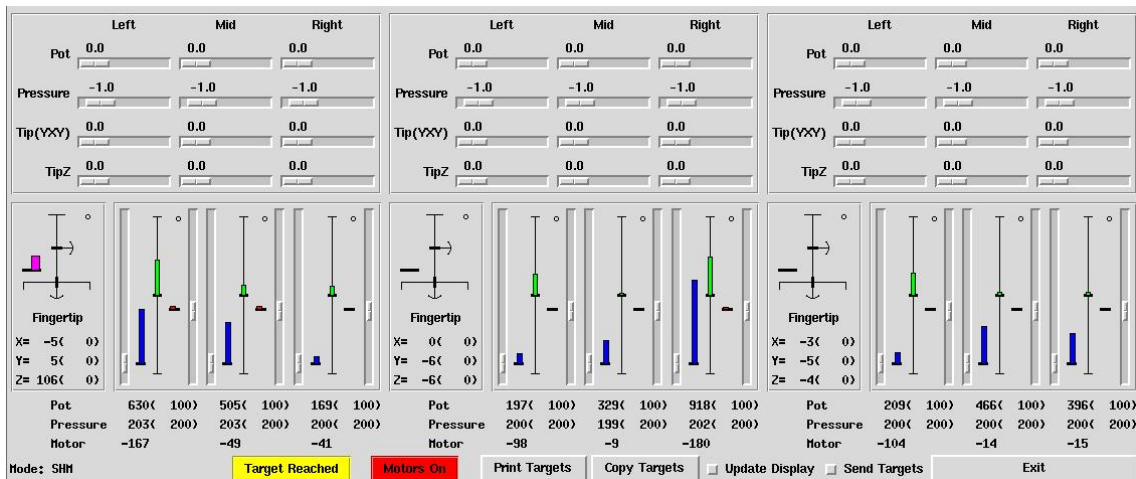


Figure 4.5: **Visual Controller Interface:** The sensor values, the current target configuration, and the error mixture settings are shown in this interactive panel. With the “send targets” option, the targets and the mixer settings can be changed and sent to the controller layer. Additionally, the complete target set can be printed for later use, e.g. in the state machine.

interactively and subsequently printed out in a format compatible with the Tcl procedure for target setting (see figure 4.5 on the facing page).

The display panel shows three equal sections in the upper area, one for each finger, and a few control switches in the lower area.

Each finger panel consists of mixer settings (upper portion), fingertip sensor readouts (left bar gauge), and three oil piston displays (lower right portion). The left and right piston displays of each finger belong to the first joint flexing and bending degrees of freedom, the middle piston belongs to the coupled second and third joint flexing motion.

Each of the colored bars on the panel shows an error. When a bar is invisible, the corresponding error is zero. The thick protruding line on one end of the bar is the target value, the other end is the current sensor value.

The piston displays show the potentiometer error to the left, the motor current output from the controller in the middle, and the oil pressure error to the right. The sliders to the far left and right of each piston display snap to the target value of the potentiometer and the oil pressure respectively, serving as both input and output devices.

The fingertip displays show the torque in the inward flexing direction as a vertical bar, in the sideways bending direction as a horizontal bar, and the pulling/pushing force as an extra vertical bar.

The mixer sliders in the top portion show the influence of each individual error on the total error reported to the PID controller of each motor. These are the potentiometer error, the oil pressure error, the fingertip x or y component, and the fingertip z component.

As soon as the net errors of all controllers reach zero, the field labeled “Target Reached” lights up. It is switched off automatically when a new target is sent to the controller.

Each slider in the displays described above can be used to alter the target settings of the controller. To prevent the casual user from unwillingly disturbing the controller, the checkbox labeled “Send Targets” must be activated before any changes can take place.

The other global controls switch the periodical display updating on and off, transfer the current sensor value profile into the target values, and print the current target setup to the standard output.

The current condition of the fingertips and the oil pressure sensors is shown as small warning “lamps”. If one of these lamps lights up and the corresponding error bar is grayed out, the corresponding sensor value is not being used for error calculation. Further diagnostic tools can then be used to spot the problem, but in most cases the necessary maintenance work is rather obvious.

This simple program helps to understand the behavior of the controller and has been extensively used in the course of programming the state machine, which we will examine in the following chapter.

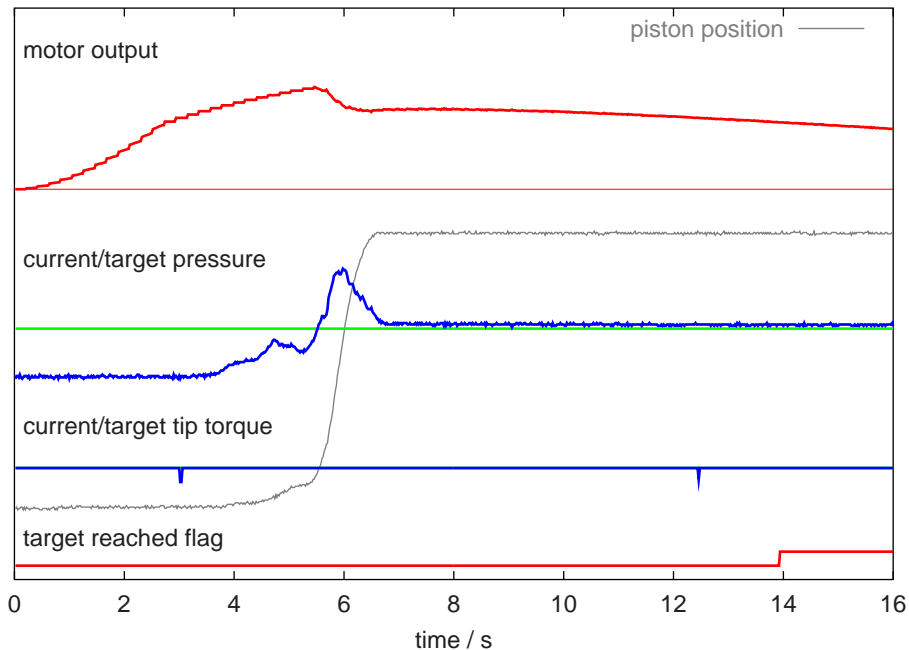


Figure 4.6: **Startup Controller Behavior:** The trace shows the first 16 seconds after switching the controller on, during which it zeroes only the oil pressure error. The oil system's peculiar behavior produces a time-lag larger than would be expected from the hysteresis alone. This is due to the stick/slip transitions in the finger piston, which are frequent because motion is slow.

4.6 Performance Evaluation

Figures 4.6, 4.7, and 4.8 show the behavior of the controller in different situations. During the startup phase, depicted in figure 4.6, the driver piston slowly moves inward to establish a safe oil pressure. The finger piston reacts by moving in several small jerks, which can be recognized in the non-monotonic trace of the oil pressure. The controller must not let the motor current output reflect the oil pressure pattern, or spurious oscillations may be the result. The trace shows that the controller successfully generates the desired oil pressure with a smooth, conservative motor current output.

Compliant gripping can easily be achieved by letting both the oil pressure and the fingertip pressure influence the net error. Figure 4.7 on the facing page shows this configuration if no object can be found. The equilibrium state of the controller is determined by the fingertip force. If no object touches, the error remains large and is only compensated by raising the oil pressure proportionally. The result is that, like a virtual spring, the finger will bend inward by a small amount.

Using the same configuration with an object produces a different outcome (see figure 4.8 on page 42). The net error is still reduced to zero, but the sensor configuration in the equilibrium state is different. In this way, client applications can examine the equilibrium

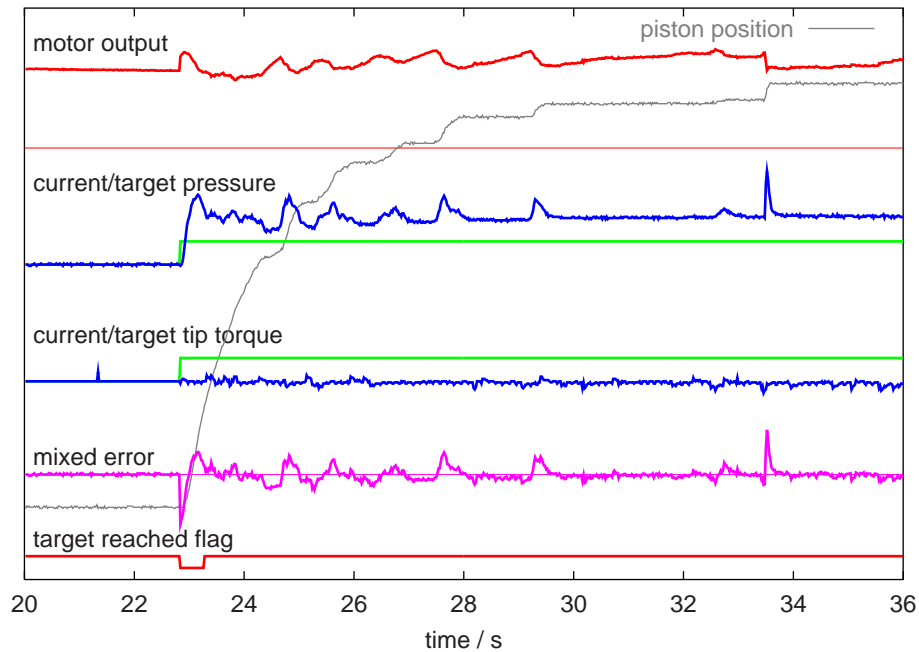


Figure 4.7: **Mixed Feedback Controller Response Without Contact:** The controller zeroes the mixed error from the fingertip and the oil pressure. The sudden changes in oil pressure are due to stick/slip transitions, and the unsteady signal from the fingertip is due to vibrations of the robot arm with a moderate amount of superimposed noise. Nevertheless, the controller configuration produces a sufficiently steady motor current output.

state as soon as the target reached flag is activated by the controller layer, and evaluate the sensor pattern to take the appropriate action.

The controller configuration shown in the traces suffices to make the hand hold an object compliantly, also tolerating some degree of intervention by a human, as shown in figure 4.9 on the following page. The nine virtual springs implemented by the controller can help to grasp objects in a robust manner. Higher level processes profit from this robustness, as we shall see in the evaluation of the state machine.

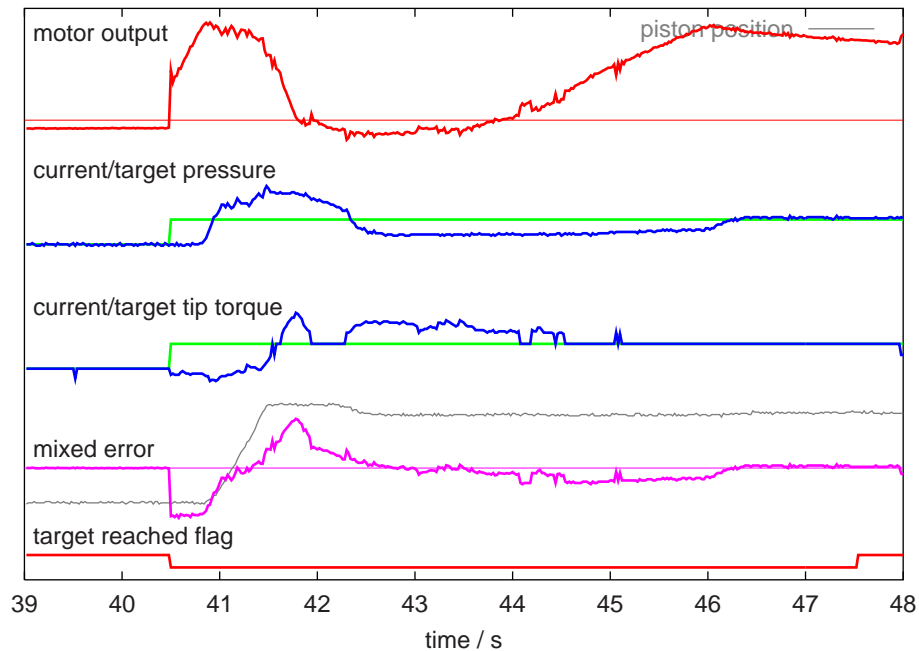


Figure 4.8: **Mixed Feedback Controller Response With Contact:** With contact to a rigid surface, the same controller configuration as in figure 4.7 on the preceding page obtains a better match for the oil pressure and the fingertip torque, but this is purely coincidental, because the controller ignores the individual errors and only zeroes the mixed error. The difference in the sensory response patterns can instead be used by higher-level processes to discriminate contact and non-contact situations.

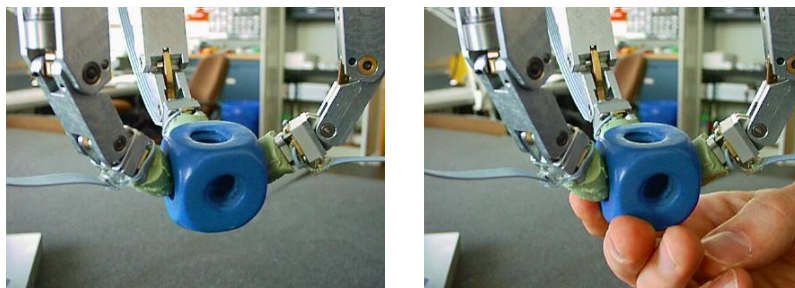


Figure 4.9: **Compliant Control:** The controller layer alone already creates some degree of interactivity, because the immediate force control produces virtual springs which grasp an object compliantly, as shown above. Higher-level processes can rely on this robustness and therefore operate on coarser time scales. The total behavior is still perceived as smooth and immediate in its reactions.

Chapter 5

The State Machine Layer

At this stage we have a controller layer which allows us to determine the level of compliance of the manipulator, a target posture, or a mixture of both. The controller seemingly shows some degree of action, but this is purely due to the inherent compliance of force control. The controller can do nothing more than simulate a set of springs of variable toughness around a target sensor feedback profile.

To implement active motion changes triggered either by client applications or by specific sensor feedback patterns, we present a state machine formalism which is simple in its basic structure but can be expanded to arbitrary complexity if required. This engine can even become a universal communication platform for integrating the robotic hand into other applications.

The present chapter first discusses the formal principle of the state machine, which incorporates simultaneous communication with several client applications along with its ability to encode behavior patterns. The following sections detail the definition of a state and of a state transition, the basic building blocks of a complete system. The chapter closes with details on the actual implementation, an example state graph, and its performance evaluation.

5.1 The Programming Principle

The most obvious task that has to be accomplished after the successful implementation of a controller which can drive the manipulator hardware is to switch from one set of controller parameters to another according to a set of pre-defined rules. Many other robotics control systems use a trajectory generation program which interpolates between the current and the target positions and feeds the intermediate steps into the controller.

We choose a different approach which does not generate trajectories, but instead encodes a certain behavior pattern. The aim is to be able to program reflex-like reactions entirely without trajectory generation. In some respects, the state machine layer might be viewed as an intermediate step between low-level control and trajectory generation which is absent in most other systems (see figure 1.3 on page 7).

Consider a typical robotics control problem for the TUM hand: an external command signals the hand to grasp an object. A guarded motion follows, during which the tactile sensors must constantly be watched to determine when to end the motion and whether the object was in fact successfully grasped. Once the object is in position, checking the sensor values is still necessary. If the object is slipping, fast re-gripping with stronger contact forces may be necessary. If the object is lost altogether, the hand may as well relax and notify other applications.

This scenario shows that especially for the robotic hand there are virtually no movements that can be done without constant sensor surveillance. This contrasts to the habitual robotics scenario where guarded motions are an exception to the rule of pre-programmed trajectories. It is therefore a straightforward step to define a programming environment in the manner of a Turing machine, with states and state transitions. The state transitions need not partition time into equidistant slices. The time between one state transition and the next may vary widely depending on the amount of external disturbances.

Naturally, the above toy problem can be solved with conventional programming methods, but the state machine formalism makes the code much clearer. As is often the case with programming languages, a given goal can be reached in many ways, but some solutions look neater and are more easily understandable than others. In this case, extending the set of situations that the program can handle is rather complicated in the traditional approach. The state machine paradigm facilitates the implementation of new behavior patterns in a purely additive way. Formerly written code remains unchanged as well as formerly designed behavior patterns.

By programming only in terms of states and state transitions, the actual control flow is hidden. Therefore, a dynamic implementation is possible, in which states and their respective transitions may be added during operation. The removal of transitions is equally possible, while removing states requires some provisions. We will turn to these topics in the section about implementation details.

The state machine model is a general programming paradigm, although we have developed it with a very special problem in mind. Naturally, only a small family of problems uses this method to its full potential. For example, loops and counters are extremely cumbersome to implement in this framework. States and state transitions have to fit into a relatively simple pattern, because otherwise the problem's complexity is only deferred into the design of the states and the programming task does not become significantly simpler. In view of these restrictions, the analogy (if not equivalence) of the Turing machine becomes evident.

5.2 *States and State Transitions*

The state machine knows a dictionary of states, one of which is the current state. Each state owns a set of possible transitions to other states, and each transition happens if some criteria are met. Note that as long as the state does not change, no action is taking place. The main loop only periodically checks the transition criteria corresponding to the current state.

A **state** consists of the following elements:

Name: This is a universal identifier for the state. State transitions address the target state by its name.

Parameter Set: Each state corresponds to a set of controller parameters, which are loaded into the controller layer when the state is activated. In more general uses of the state machine formalism, such an attached data structure may either be omitted or replaced with a generic action taken upon entering the state.

State Transitions: A list of possible state transitions. This list may be dynamically expanded or reduced.

Each **state transition** contains:

Target State Name: The identifier of the state that can be activated by this transition.

Switching Criterion: An expression which returns a boolean value, determining whether the state switch should take place or not. This expression has access to all sensory information from the controller layer and can use this information to respond to certain feedback patterns.

In our scenario, transition templates are provided for some typical situations, where triggers are needed at a certain timeout, upon reaching a target, or after surpassing an error threshold. These templates can be parameterized and used in several states.

The state machine can also accept state transitions from outside its own graph. This is necessary for creating a user interface in which states may be switched by hand, but its more universal use comes into play when other complex programs, possibly state machines themselves, introduce new state transitions. We will discuss this topic at the end of this chapter.

5.3 *Implementation Details*

The state machine implementation has to take the exceptional flexibility of the concept into account and still offer adequate performance for the task. The timing diagram (see figure 3.2 on page 28) shows that a polling loop frequency of about 10 Hz is sufficiently fast in view of the reaction time of the hydraulic actuation system. It also compares fairly well to the reaction time of a human, which makes the state machine's behavior seem natural to the operator.

This timing restriction is easily met even in script languages. We chose Tcl/Tk for the implementation of the state machine because of its remarkable flexibility and because it was already in use for other parts of the project. One of the benefits of using Tcl (or almost any other interpreted language) is that it is relatively easy to implement the addition of states and state transitions because program code can be contributed as a text which is then evaluated by the interpreter.

The interfacing code for inter-process communication consists of an ANSI-C Tcl plug-in which handles a shared memory segment. This segment contains the name of the current state as a string, a mutex¹ for this string, and a set of flags which other processes may use as signals for change notifications. Because of the state machine principle, the identifier string suffices for all external communication. Because the state changes at irregular intervals, a change notification mechanism is necessary to eliminate polling loops in client applications.

As in the controller layer, network-wide communication is handled by DACS. The DACS interfacing code is executed in a separate process, which attaches to the shared memory segment described above. It provides two functions, "getState" and "setState", with obvious meanings, and a demand stream, "newState", which allows client applications to receive event triggers as soon as a state change takes place. Even several interconnected state machines can be implemented in this way, which work totally independent of each other most of the time, but still synchronize and react to each other depending on the situation. We will return to this in a later section.

The example code snippet (see figure 5.1 on the next page) shows that the state machine can in fact be very easily programmed. The versatile expression evaluation capabilities of any modern high-level programming language are also available in Tcl, allowing us to formulate complex condition expressions for the state transitions. And the scripting even allows the complete state execution mechanism to be re-implemented for individual states.

Let us now examine the performance of the state machine in a real-world scenario.

¹A mutex is a standard POSIX mechanism for controlling distributed simultaneous access to selected memory areas.

```
set states(Stretch) {
    State_run $targets(Stretch) {
        {Crit_timeout 3000 Loosen}
        {Crit_targetReached Loosen}
    }
}
set states(Loosen) {
    State_run $targets(Loosen) {
        {Crit_pressureErrorIn 4 Wait}
    }
}
set states(Wait) {
    State_run [copyTarget "$currentProfile"] {
        {Crit_potErrorOut -300 Grip}
    }
}
```

Figure 5.1: **Example State Machine Code:** The Tcl implementation of the three preparatory states shown graphically in figure 5.2 on the following page consists of the definition of entries in an array called “states”, each of which is a script starting with the generic state evaluation command “State_run”. This command takes the controller parameterization and the list of transition criteria as arguments. The criteria use templates for watching a timer, the pressure sensors and potentiometers, and the controller’s target reached flag.

5.4 An Example State Graph

In the human-machine communication scenario described briefly in the introductory chapter, the robot hand shall pick up an object pointed at by the human. In this example, the vision algorithms for gesture recognition and object fixation deliver an approximate estimate of the object’s position, and trigger the first approach motion of the robot arm. The wrist camera then performs object recognition in its own right to correct the approach position for grasping, and commands the hand to grasp the object (see figure 1.2 on page 4).

Figure 5.2 on the next page shows the hand’s state graph for this simple application. The two commonly traversed threads are shown, one preparatory sequence, and one gripping sequence. The hand obtains no orientation or size information on the object, which simplifies the state diagram. For different grasping strategies, different state threads could be designed. A classification algorithm, which decides which strategy to adopt, would then replace the simple gripping trigger used now.

This minimal state graph already proves the usefulness of the layered control architecture concept. The compliance of the controller allows the hand to grasp and hold a large variety of objects, and the state machine identifies exceptional situations and handles communication with client applications. A typical grasping sequence is shown in figure 5.4, which demonstrates the coupling of robot motion and hand reflexes. A simple user interface can

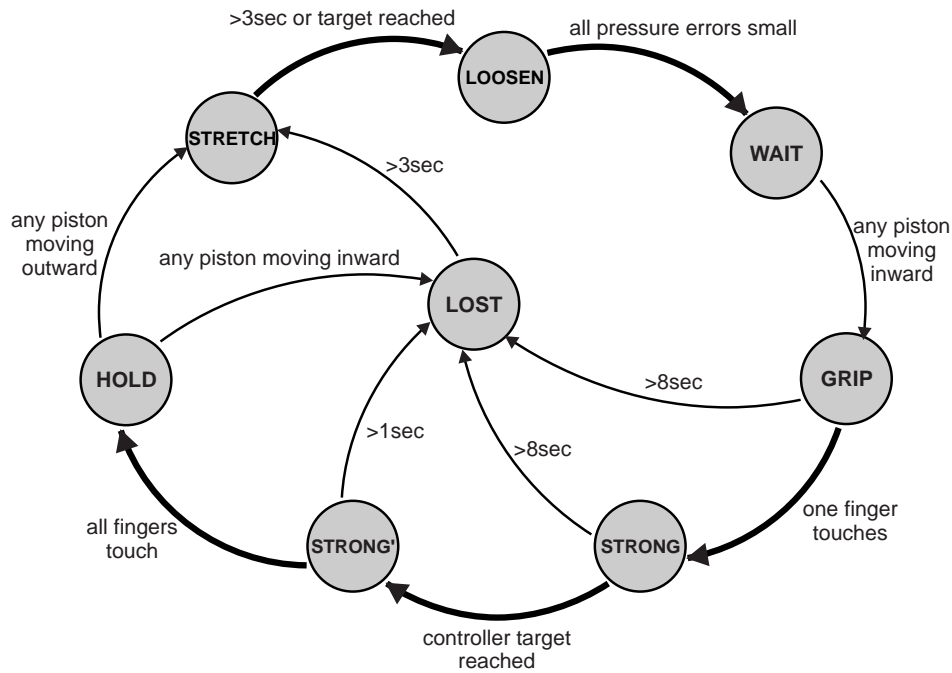


Figure 5.2: **State Graph for Grasping:** The figure shows frequent state transitions in thicker lines than infrequent ones, which exhibits two sequences, one for moving the hand into a defined waiting position, and one for grasping an unknown object. The top sequence, stretch–loosen–wait, moves the fingers outward quickly, then contracts into a relaxed posture, waiting for a signal to grasp an object. This signal either comes from a client application, or it is triggered by an operator bending a finger inward. The bottom sequence, grip–strong–strong’–hold, first carefully establishes contact with at least one finger, then raises the gripping force to safely hold the object with all three fingers. If successful, the mechanism stays in the holding state. At several stages there are fallback transitions to the state “lost”, which serves as a failure signal to outside applications.

be used to monitor the inner workings of the state machine, and to force state transitions externally.

5.5 Interleaving State Machines

Being able to communicate with the state machine by querying the current state and forcing state transitions has some noteworthy implications. In the above toy problem, an external application may command the state machine to grasp an object by issuing a forced state transition to a state labeled “Grip”. This may well be compared to a function call in common programming languages. The return value is not delivered instantly, though, because it cannot be. After some time, the state machine might enter a state labeled “Hold” or a state labeled “Relax”. The external application will register this change and treat it

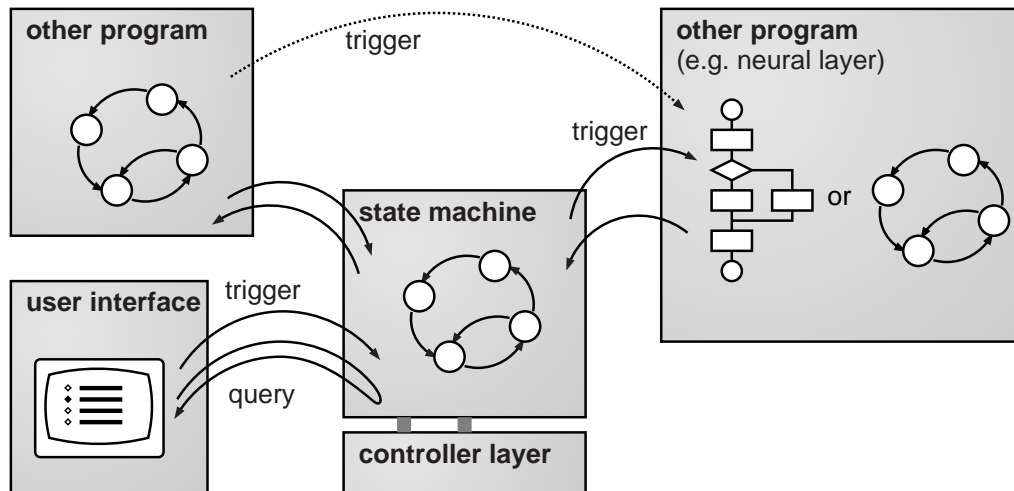


Figure 5.3: **Communication with State Machines:** The trigger mechanism of the state machine’s communication interface allows complex interaction with other applications and state machines. Many entities can operate independently, reacting on triggers from other entities and triggering state changes in those entities. The result is a much more complex state machine on a higher level of abstraction, possibly yielding a surprisingly versatile system based on relatively simple components.

as a return value, because it reflects the current situation as a consequence of the situation triggered by “Grip”.

In this way, the state machine blurs the distinction between *function calls* or *inputs*, and *return values* or *outputs*. Several restrictions of conventional function calls are absent in this model:

Ambiguous Responses: Which return value should the “Grip” command described above deliver, if the object was successfully picked up after one second? It could just as well be lost in two seconds. How long should the function wait to be sure the return value is correct? In an environment of constant external disturbances, e.g., in the presence of a human operator who may interfere at any time, these questions cannot be answered satisfactorily. In contrast, the state machine approach does not even ask these questions. The current state may change several times, and it is up to the client application how to react to those changes.

Interfering Commands: The system may receive commands from several sources, e.g., its own transition set, the user interface, and a client application. With a function call interface, one source of commands may block the other, resulting in a degradation of the total robustness and reactivity of the system. The question arises when to use blocking calls, and when to use non-blocking ones. Since the state machine delivers no

conventional return values, all commands are non-blocking, and interference between command sources cannot erode the system's performance.

Busy Loops and Events: Client applications can decide individually whether they wish to poll the state machine or rather be notified of any changes. This allows programming environments with different underlying paradigms to communicate with the state machine in the fashion they prefer.

One considerable drawback of the state machine technique should be mentioned, though. Because the problem formulation with states and state transitions hides the control flow, it is difficult to exactly predict the behavior of a state machine. With coupled state machines, the danger of deadlocks becomes more prominent. There is no simple scheme to avoid deadlocking, and loop identification itself can become difficult in the running system. There are provisions in the state machine to introduce a relaxation time which inhibits external state transitions if they follow up on another too fast. This at least can make some deadlocks visible for the human eye.

5.6 *Conclusions*

In the past chapters, we have introduced a control and behavior simulation system engineered to fit the special hardware equipment in our laboratory. Nevertheless, the structure of the system has been formulated in a general way, and it is thus applicable to a large number of control problems.

Especially in cases where fast reactions are valued higher than the exact tracing of pre-programmed trajectories, this system can provide a powerful and simple solution.

The system further offers an interfacing scheme which enables an arbitrary number of clients to command the control system and to query its state, all without impairing the overall performance or increasing the reaction delays. In distributed systems, where many components work asynchronously and simultaneously, this feature is extremely useful.

In the following chapters, we will turn to the topic of trajectory planning, which cannot be solved efficiently by the state machine alone. We will find that, along with trajectory planning, obstacle avoidance and exploration, both passive and active, almost automatically come into play. The close relationship between these disciplines shall be discussed first.

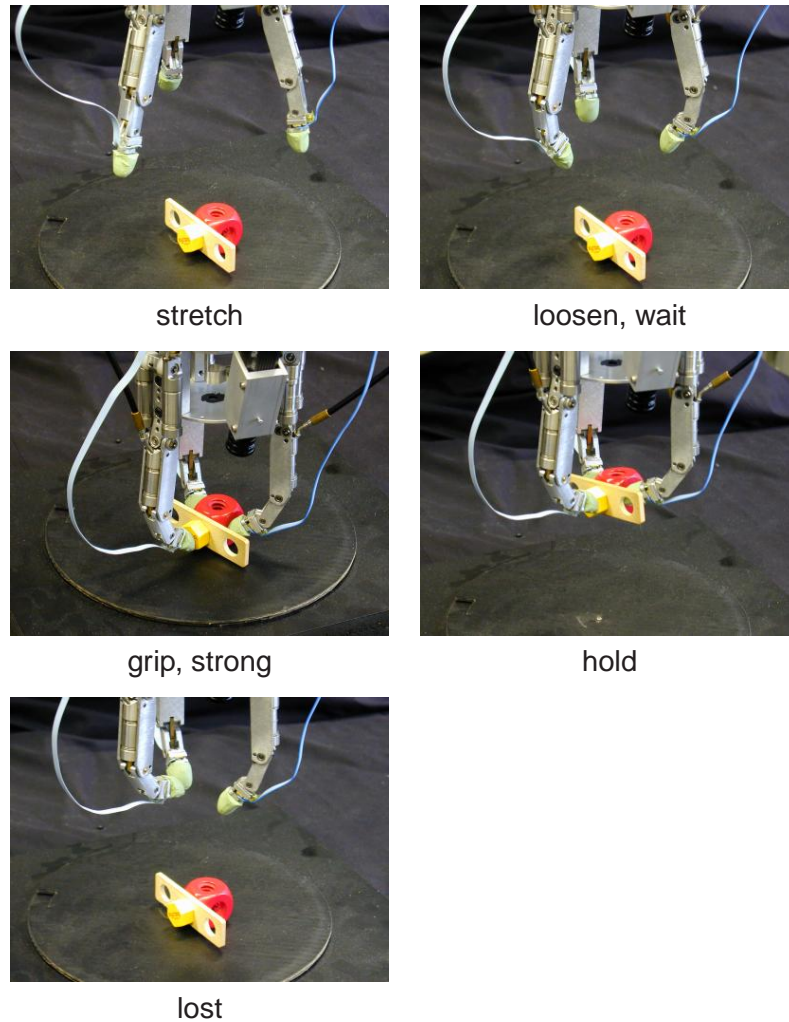
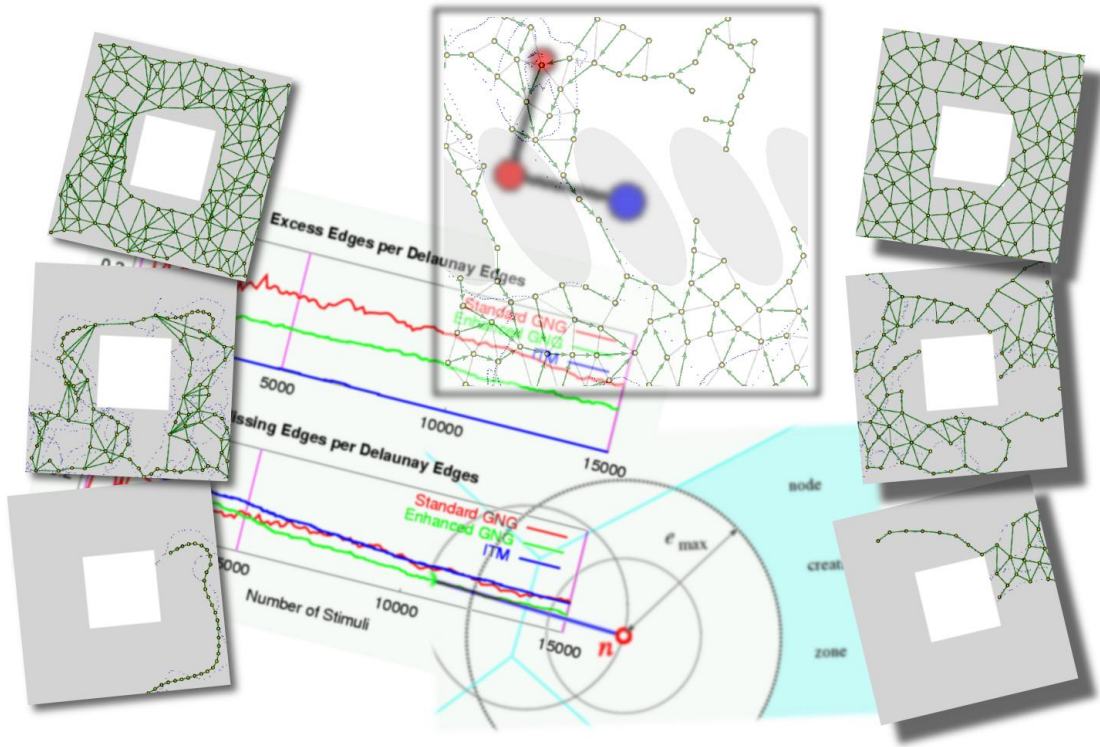


Figure 5.4: **Photo Series of a Grasping Action:** The top row shows pictures during the preparatory state sequence stretch–loosen–wait (see figure 5.2 on page 48). In the middle row, the sequence grip–strong–hold can be seen when grasping a simple object. In this case, the robot arm is controlled by a second state machine. When the operator touches a finger, this machine lowers the arm onto the object and triggers the grasping sequence. If the human steals the object or the manipulator loses it, the hand’s state machine signals this condition (bottom row) and then re-enters the sequence with a stretching motion.



Part II

*Exploration with Vector Quantization
Networks*

Chapter 6

Approaching Intertwined Tasks

6.1 Introduction

With the low-level controller and state machine layers described in the previous chapters, we have built a system with reflex capabilities and fixed, albeit re-programmable, behavior patterns. The system so far has no knowledge about its surroundings and is unable to perform long-term planned motions. Since we wish to change this, let us first briefly recall the most commonly used method of trajectory generation.

In traditional robotics control systems, the natural thing to do after having built a positional controller for a manipulator is to make the manipulator move from one designated position to another. This is achieved by interpolating between a starting and a target position while taking position, velocity and acceleration limitations into account. Especially the acceleration limits pose some problems in hitting the target position, sometimes producing oscillating or orbiting trajectories.

This task of trajectory generation is surprisingly intricate in itself, and is therefore generally treated independently of obstacle avoidance or exploration. These are considered higher level tasks which themselves take control over the trajectory generation.

But viewed from a different angle, the tasks of trajectory generation, exploration, and obstacle avoidance in fact form a group of strongly coupled problems. What joins them is the fact that they all work with some representation of the manipulator's workspace (see figure 6.1 on the following page).

Trajectory generation requires knowledge about the manipulator's state space in order to find a path which is optimal in some given respect, like short execution time, or short spatial distance. Exploration provides this knowledge, for instance about joint angle limits, long-term obstacles, or even information about manipulator dynamics. Obstacle avoidance

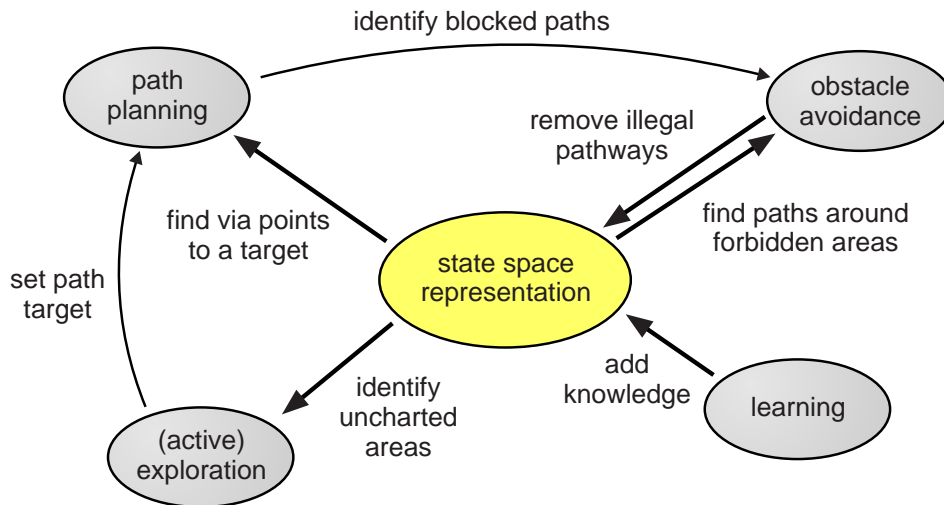


Figure 6.1: **Interplay of Different Tasks:** Although they may at first seem unrelated, state space mapping through learning, path planning, obstacle avoidance, and exploration interact closely through their knowledge of the state space, which then acts as a communication hub in the star-shaped structure shown. Choosing the state space representation in a way that satisfies the requirements of each of these tasks can provide a simple and powerful high-level control architecture.

is a special case of trajectory generation. It uses knowledge about possible obstructions to find the best path to a given target position while at the same time avoiding collisions.

The motivation for the research presented in the following chapters is to find a set of algorithms which enables us to solve these tasks in a way that takes advantage of their close relationship.

To this end, we first propose a vector quantization representation of the workspace, a “map” which some algorithms expand and modify, and which others use to maneuver the manipulator. One consequence of this approach is the creation of a novel neural network type, the ITM, which is especially tailored for trajectory control problems. But let us first review the most prominent vector quantization network types which inspired this development [17].

6.2 *Literal Interpretation of Topological Maps*

Neuro-informatics offers a wide variety of vector quantization and topological mapping networks, most notably, of course, Kohonen’s Self-Organizing Map (SOM), which has inspired an astounding number of scientific publications.

The interpretation of the mapping produced by such a network is usually rather abstract, indicating a close relationship between the vectors corresponding to neighboring cells, or proposing a lower-dimensional ordering of the input data. This abstract interpretation has led to simple and effective means of data analysis. We will return to the data mining aspects of topological maps in chapter 7.

But from a roboticist's point of view, interpreting a topological map literally can provide a score of new possibilities. The nodes represent selected (quantized) positions in the manipulator's state space, while the edges represent possible pathways from one node to the other. This viewpoint also allows us to benefit from graph theory, which provides efficient path finding algorithms, for example.

This approach requires the use of a vector quantization network with considerable flexibility and adaptability in its graph structure. We will examine several potentially useful network types in the next chapter.

Note that keeping a "list" of vectors that span the state space we wish to map means that we have to accept a quantization error introduced by the spacing of those vectors. Geometrical state space descriptions do not have this drawback, but there is no easy way of building such descriptions for arbitrary state spaces. The quantization error is therefore a small price to pay for the simple and general formulation of topological maps produced by the networks described.

6.3 *Evaluation of Existing Models*

We need to find a network model which will perform well in a robotics scenario. The following list introduces the most prominent existing network types and outlines their adaptation algorithms. The list is not complete, but focuses on algorithms which exhibit different operation principles.

The Self-Organizing Map (SOM)

First introduced by Kohonen [23], the SOM consists of a fixed number of nodes arranged in a fixed topological order, e.g., a grid or a chain. The adaptation consists of moving the weight vector closest to the stimulus and its topological neighbors toward the stimulus ξ by a small fraction, the learning rate ϵ . The learning rate and the smoothness term σ , which defines the topological radius of the influence of a single stimulus, are lowered systematically throughout the learning process. This procedure, called "simulated annealing", is crucial for the quality of the final map. Especially lowering the smoothness term too fast is likely to produce unwanted topological warps, as shown in figure 6.2 on the next page.

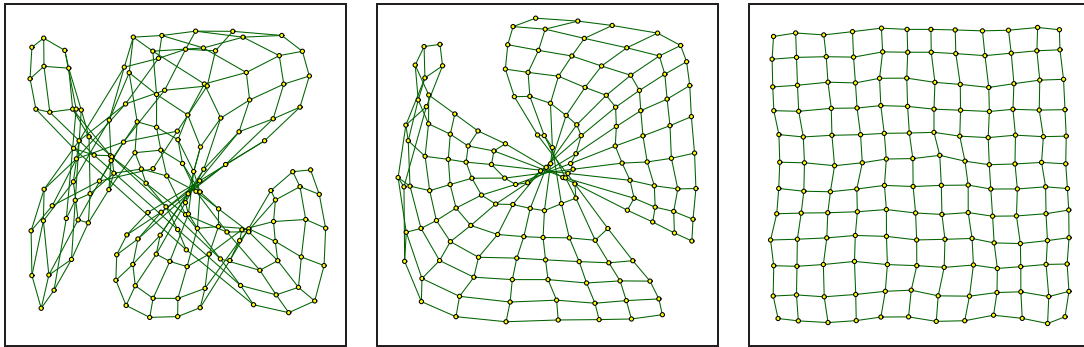


Figure 6.2: **Topological Warping in SOMs:** This two-dimensional square SOM grid exactly matches the topology underlying the stimuli, a square two-dimensional uniform random distribution. Nevertheless, the resulting mapping does not necessarily be ideal, as in the rightmost example. Fast annealing of the smoothness term σ and the learning rate ϵ results in topological warps, shown in the left and middle pictures. The relationship between the neighborhood in the feature space and in the grid becomes locally disrupted.

The Parameterized Self-Organizing Map (PSOM)

The PSOM, introduced by Ritter [41], does not require a long iterative adaptation process like the SOM. Based on assumptions about the smoothness of the input space, the PSOM generates an interpolation map based on a fixed number of nodes arranged in a fixed topological order. Because of its interpolation capabilities, the PSOM can be used for learning smooth transformations with few input samples. Like the SOM, the PSOM can be used as a simple associative memory to fill in missing entries in an input vector, but the PSOM generates missing entries by evaluating the interpolation of all nodes at the position of the stimulus. The quality of the input stimuli during training is therefore crucial for the resulting map.

The PSOM has been successfully applied in learning geometrical transformations, because its interpolation feature blurs the quantization effect visible in standard SOMs. Its major drawbacks are the strong dependency on reliable samples, and the impossibility of incremental learning.

The Hyperbolic Self-Organizing Map (HSOM)

Yet another variant of Kohonen's SOM, the hyperbolic SOM, introduced by Ritter [39], presents an elegant way of circumventing the dimensionality problem. Its topology is still fixed, but it is hyperbolic in that the neighborhood of a single node rises exponentially with the neighborhood radius (as opposed to Euclidean neighborhood relations, which rise by the power of the dimension). In theory, hyperbolic SOMs adapt equally well to data of arbitrary dimensionality, if the available neighborhood is sufficiently large, i.e., the network contains enough nodes.

When used to map Euclidean input spaces, the hyperbolic SOM always produces warps to some extent. If a hierarchical structure of the input space is to be found, the hyperbolic SOM can provide that information due to its neighborhood structure.

The Growing Grid

Endowing an SOM with the ability to insert new rows or columns of neurons produces this network type [7, 9]. It is especially useful if the dimensionality of the map remains fixed, but its optimal size is unknown. The growing grid must be carefully parameterized, just like the SOM, to avoid topological warps. The growth speed of the grid must also be tuned to match the overall adaptation speed of the network, because a grid growing too fast becomes increasingly susceptible to topological warping.

The Neural Gas

In some situations, a topological graph structure is not necessary. The Neural Gas by Martinetz [27] is a collection of nodes which adapt to a set of input stimuli. This neural network type has been the basis for many newer variants of vector quantization networks, like the Growing Neural Gas.

The Locally Linear Map (LLM)

Since vector quantization networks generally map the input stimulus onto a node index, they always deliver a quantization error. The LLM, introduced by Ritter [38], is a cross between the error-prone vector quantization approach and the simple linear interpolation approach. Each node of an LLM carries a linear equation instead of a constant vector. The adaptation rules for the LLM are only slightly more complex than those of the neural gas. The LLM has proven its usefulness in many real-world mapping tasks, e.g. in hand or head posture recognition [30, 35, 40]. Like the neural gas, the LLM does not carry topological knowledge in the form of a graph connecting its nodes.

The Growing Neural Gas (GNG)

This network type by Fritzke [8] starts out with a minimal connected graph of two nodes, and builds an arbitrarily complex graph by periodically inserting new nodes where the potential of lowering the error is considered highest. During this growing process, the nodes' weight vectors are adapted as in the SOM, but using the current graph structure

instead of a fixed grid. The graph structure itself is built by connecting the two nodes closest to a stimulus and by removing edges that have not been rebuilt for some time.

The number of nodes in a GNG grows linearly with the number of input stimuli, making the appropriate timeout parameter critical in almost any application. The GNG's greatest advantage is its ability to map almost any topological arrangement efficiently.

6.4 *Critical Aspects in Robotics*

In a robotics setting, the neural network's adaptation algorithm defines its suitability for the exploration aspect, while the quality of the generated graph is crucial for the path planning aspect. The critical requirements for a suitable neural network are summarized below.

Flexible number of neurons: Topologically rigid networks need much a-priori knowledge about the problem at hand. Finding the optimal number of nodes means having to run many tests. Therefore, we require a network able to grow or shrink as necessary. Nevertheless, we shall carefully observe the features of topologically rigid networks to find ways to further improve the flexible networks.

Local topological flexibility: The state space we wish to map can have varying local dimensionality, and the mapping must be able to reflect this fact. Therefore, models with uniform dimensionality, like the SOM, the PSOM, and the Growing Grid, do not qualify. The HSOM can cope with variable dimensions more easily, but nevertheless has a fixed topological arrangement.

Insensitivity to short-term correlations: The input data originating from a real-world manipulator is not uniformly distributed across the input space. The strong short-term correlation must be accounted for in the adaptation process, because we do not want to employ special measures to reduce the correlation. The network model must be able to handle the data and still adapt efficiently. The SOM and the Growing Grid need a lower setting for the learning rate to produce useful mappings in this situation, and thus adapt less efficiently. All network models introduced here are susceptible to correlations in the input stimuli, because they are based on the assumption of a statistical input distribution. But in the GNG, this shortcoming can be alleviated to some extent with a node generation enhancement which we will introduce in the next chapter.

Fast adaptation: The network shall deliver a useful representation of the knowledge gathered from input signals from the very beginning of the learning process. Of all the models presented here, the GNG is the only one which can be optimized to this end, but the high number and interaction of the necessary parameters makes this optimization cumbersome and fragile.

Graph usefulness: The graph constructed by the network during adaptation must be suitable for path finding. Too many stray or missing edges can severely deteriorate the quality of the paths represented in the graph. Generally, stray edges are less favorable than an equal number of missing edges, with respect to a perfect Delaunay triangulation. With its edge aging mechanism, the GNG has a major disadvantage in this respect, which we will discuss in comparison with the ITM.

Incremental learning capability: Because a manipulator may stay in the same area for a long time, and then move on to totally uncharted areas, the network must be able to incorporate this new knowledge without prior changes to its parameters, like the learning rate. The network must also be immune against destructive interference, i.e., mapping a new area must not destroy the mapping already established. The node generation enhancement provides the GNG with the necessary capabilities, but the parameterization introduces a typical time scale for the adaptation process which must be known in advance.

Simple overall structure: The network will be embedded in a larger set of algorithms, which will also modify the graph structure. The network algorithms' simplicity can help to improve the robustness of the total system. The GNG has many internal state variables, including age counters and error accumulators which are kept separately for each edge and node, respectively. This disadvantage has led us to search for modifications and simplifications of this algorithm.

Of all the network types outlined here, the GNG is the most promising for delivering a state space representation from short-term correlated input data. But its cumbersome configuration and the large number of internal state variables motivates the design of a new network type, the Instantaneous Topological Map, which is tailored to fulfill all of the requirements specified above. In the next chapter, we will introduce this novel network type and compare it with its strongest competitor, the GNG with node generation enhancement.

Before turning to the ITM, let us consider two problems found when using vector quantization networks on ill-prepared, serially correlated data. One is input preparation, a standard discipline in neuro-informatics, the other is the well-known contraction effect found in the SOM and other related network types.

6.5 Preparation of Input Data

It is a well-established fact that careful preparation of input data is an integral part of the training process. This involves (i) choosing a suitable subset of the available data, (ii) transforming the data into feature vectors of a chosen dimension, and (iii) rescaling the data so that a chosen metric provides meaningful distance information necessary for many learning algorithms.

The preparation of input data has a strong impact on the training process, and careless preparation can easily inhibit convergence.

The choice of a data subset is the logical starting point in most applications, because most machine learning algorithms are rooted in statistics while only few data sets originate from statistical processes. This discrepancy has led to some “recipes” to enable the learning algorithms to cope with such data. These involve randomizing the order of the input data and feeding a set of data repeatedly. Randomizing the order makes the data “look” more statistically distributed than the sorted version of the same data set, because most learning algorithms contain hidden low-pass filters, and therefore ordered sequences of slow-changing input data produce strong destructive interference. On the other hand, repeated presentation of a set of data allows the learning algorithm to converge more slowly to a local minimum of its error function, by choosing a smaller learning rate, for example. This results in a better overall stability of the learning process.

In a control scenario like the one presented in this thesis, we wish the adaptation process to be as fast as possible, in order to obtain a useful topological map of the input space from the very beginning of the training. Additionally, we do not wish to implement a special “training” phase which we would have to stop to begin using the map. Instead, the neural network shall continue to adapt incrementally using a steady stream of input stimuli. Therefore, we will not use re-ordering or repeated presentation as described above. Instead, we will change the adaptation algorithm to cope with input data which exhibits strong serial correlation and is thus far from the statistical ideal.

Another key to successful machine learning is the wise choice of the feature vectors’ components. In computer vision, most notably, Gabor filters and jets, blob analysis, color transformations, and many others form a huge arsenal of feature extraction methods, all designed to produce meaningful components for the input vectors. The feature extraction phase often drastically reduces the dimensionality of the data and focuses on elements in the data known (or supposed) to be necessary for successful training. Reducing the dimension of the input vectors also helps to reduce the number of degrees of freedom of the associated neural network (e.g., the number of neurons in a multi-layer perceptron (MLP), or the size of the weight vectors in vector quantization networks), and therefore speeds up the adaptation process considerably.

An astounding amount of knowledge has been gathered about feature extraction, because algorithms in machine learning are generally sensitive to rising dimensionality of input data. In this thesis, we will look at this problem from the network’s perspective, trying to make the network itself less susceptible to the omnipresent dimensionality problem.

The final aspect of input preparation mentioned above is data rescaling. The background for this aspect is that many learning algorithms use a chosen metric to calculate a scalar distance between pairs of input vectors. On the other hand, the input vector components often have diverse origins and interpretations, and sometimes differ widely in their typical scales. Using these vectors in a typical distance metric, like the Euclidean metric, generally does not produce meaningful output.

Typically, the input vectors are rescaled to center each component around zero with a mean deviation of one. This step ensures that the distance measure reacts approximately equally to changes in any component of the feature vectors. The mean and the deviation of each component are calculated over all input vectors, and subsequently used to rescale the input data.

The scenario depicted in this thesis disallows such an a-priori rescaling of input data, because learning shall take place continuously over an indeterminate amount of time. Because we still want to perform component equalization in the manner just described, we implement a new metric with a built-in component rescaling vector. This vector is adjusted using a slow rolling average filter, which performs essentially the same mean and deviation calculation as in the a-priori method above. The following section gives the details of this input preparation technique.

6.6 Adaptive Metrics for Input Rescaling

Consider the following modification of the Euclidean distance formula.

$$D^2(\underline{a}, \underline{b}) := \frac{1}{n} \sum_{i=1}^n \frac{(a_i - b_i)^2}{d_i} \quad (6.1)$$

Each component's square difference is divided by a distinct d_i . Were this equal to the square deviation σ_i^2 over all existing input vectors' i components, then this division would produce the same result as an a-priori rescaling of the input vectors to mean zero and deviation one ($\bar{\xi}_i = 0, \sigma_i^2 = 1$).

The additional factor $1/n$ makes the average distance value delivered by this metric independent of the dimensionality of the input vectors. This factor is irrelevant for individual learning scenarios, but because some parameters for the GNG, the ITM, and other vector quantization networks are formulated in terms of typical distance values, a distance metric with a universally constant scaling is of great practical value.

Note that the Euclidean metric can be formulated as a scalar product: $D^2(\underline{a}, \underline{b}) = (\underline{a} - \underline{b})^2$. Because the scalar product formulation is more practical and more general, we rather redefine the scalar product to use the above metric:

$$\underline{a} \cdot \underline{b} := \frac{1}{n} \sum_{i=1}^n \frac{a_i b_i}{d_i} \quad (6.2)$$

To calculate the estimates d_i for the square deviations, we employ several rolling average filters¹ which are adjusted with the input vectors ξ . One set of filters, c_i , approaches the

¹A rolling average filter is a simple discrete low-pass finite impulse response (FIR) filter, which approaches a fixed target value exponentially with a decay constant defined by the amount of influence of the previous filter value (Γ in the examples given here).

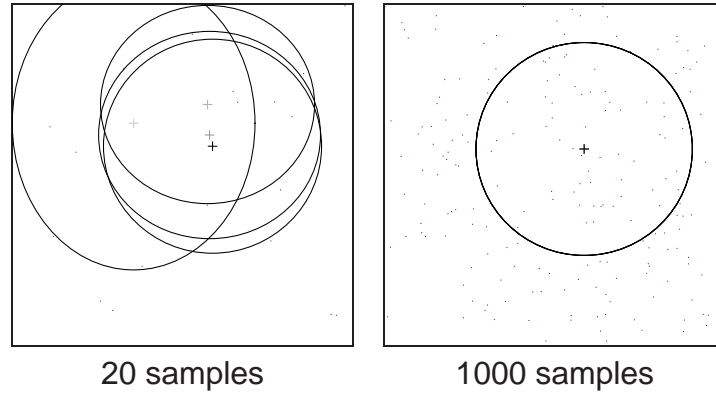


Figure 6.3: **Mean and Deviation Estimation:** The presentation of only 20 input samples results in the convergence of the c_i and d_i estimates shown in the left panel. At this point Γ is almost at its final level, and a much slower correction takes place. The estimates exhibit almost no further change even after 1000 input samples, as shown in the right panel.

mean of each component $\bar{\xi}_i$, the other set of filters, d_i , approaches the deviation of each component $\overline{(\xi_i - \bar{\xi}_i)^2}$. The amount of influence of each new input vector is determined by Γ , resulting in the following set of adaptation formulae.

$$\begin{aligned} \Delta_i &\leftarrow \xi_i - c_i \\ c_i &\leftarrow c_i + \Gamma \cdot \Delta_i \end{aligned} \quad (6.3)$$

$$\begin{aligned} \delta_i &\leftarrow \Delta_i^2 - d_i \\ d_i &\leftarrow d_i + \Gamma \cdot \delta_i \end{aligned} \quad (6.4)$$

The factor Γ introduces a typical time scale into the system. A large value will make the d_i fluctuate wildly, rendering the metric from equation 6.1 on the preceding page useless. A small value of Γ will produce the desired result in the long run, but at the beginning of the metrics adaptation much depends on the initial values of c_i and d_i . To obtain the desired behavior, we modify the value of Γ itself, starting with 1, then lowering it down to a desired target value of γ using a decay constant of λ .

$$\Gamma \leftarrow \Gamma + \lambda(\gamma - \Gamma) \quad (6.5)$$

This method makes the initialization of the c_i and the d_i less critical. It can be interpreted as an alternative to initialization. In settings where the a-priori knowledge of the input data is sufficient, we may supply that knowledge by properly initializing the mean and deviation estimates, and by setting Γ to an initial value close to γ . If no a-priori knowledge can be supplied in this way, Γ can be initialized to a higher value to reflect the lack of confidence in the initial c_i and d_i .

A large value of Γ gives the first few samples more influence on the intermediate estimates, and therefore those samples should be chosen with care. As can be seen in figure 6.3, as

few as 20 well-chosen samples suffice to kick-start the adaptive metrics system into almost perfect operation. The choice of initial samples is recommended, but it is not vital. The rolling average filters converge to the same estimates regardless of the behavior during the initial phase. But since even a mediocre, automated initial sample choice can significantly improve the transitional phase, this method can be viewed as a favorable tradeoff.

6.7 Expansive Adaptation

When mapping an input space using a vector quantization network, like an SOM or a GNG, one of the notable effects is that nodes stand off the edges by about half the diameter of a typical Voronoi cell. This is the natural equilibrium state in which as many samples fall to each side of a single node. In fact, there are generally several stable equilibrium states, with edge node positions influencing all other node positions². Figure 6.4 shows this effect for the GNG and the SOM.

The border left by the nodes is, naturally, optimal in terms of entropy maximization as performed by the SOM. Therefore, any measure taken to make the nodes move toward the border results in a slight deterioration of the mapping quality. But since our main interest lies in using topological maps literally as road maps, total coverage of the input space is more relevant than strict entropy maximization.

The method proposed here involves an additional weight vector manipulation of the node n closest to the stimulus ξ . If the winner node has more than zero, but less than average³ number of neighbors, the weight vector is modified according to the following formula.

$$\begin{aligned} \underline{g} &\leftarrow \frac{1}{\#N(n)} \sum_{i \in N(n)} \underline{w}_i \\ \underline{w}_n &\leftarrow \underline{w}_n + \epsilon \eta (\underline{w}_n - \underline{g}), \end{aligned} \quad (6.6)$$

where $N(n)$ is the set of neighbors of the nearest node n , and \underline{g} is the center of gravity of that node's neighbors' weight vectors. The adaptation step moves nodes with less-than-average number of neighbors farther away from their neighborhood.

On one hand this produces approximately the desired coverage of the input space, but on the other hand it introduces a strong distortion of the map at the edge. In two dimensions the distortion is limited, but in higher-dimensional spaces this may become a problem. Further tests and a thorough analysis of this technique must be made to pinpoint its limitations in more detail. We anticipate, though, that the algorithm is robust given moderate

²For the special case of a linear SOM, Kohonen has presented a solution to the equilibrium equations, which exhibits an intricate periodic spacing of the nodes. For higher dimensional grids, the solutions are expected to be much more complex.

³The average number of neighbors equals two times the total number of edges over the total number of nodes.

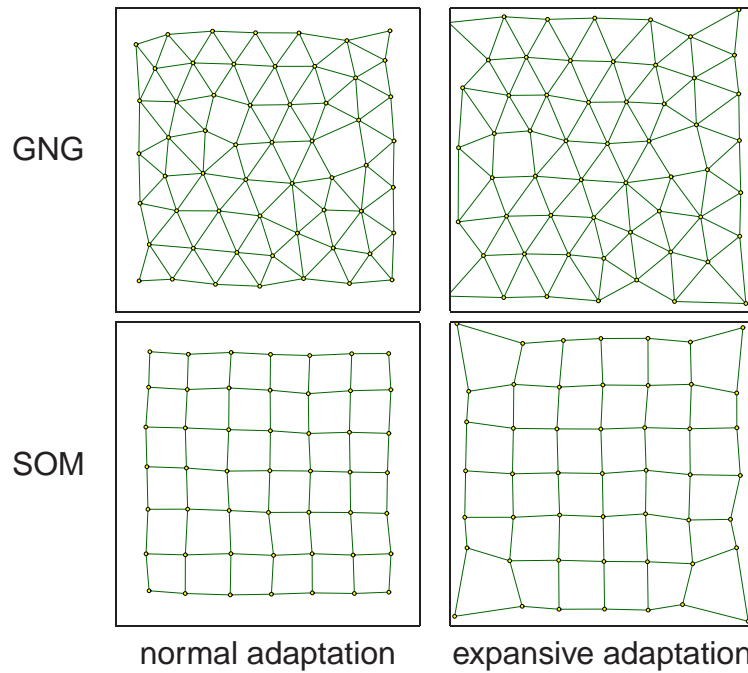


Figure 6.4: **Expansive Adaptation:** The panels show typical equilibrium states of a GNG and an SOM on the left hand side. Both models adapt the vectors in the same manner and therefore result in a map with a border about half the typical diameter of a Voronoi cell wide. The addition of a simple expansion scheme yields the maps on the right hand side. Nodes in exposed positions, especially in the corners, move farther away from their neighbors, producing a distortion of the map with better border coverage.

values of η , because it affects only at most half of the nodes, and because the expansion motion can always be compensated and damped by the normal adaptation rule. Convergence should therefore still be guaranteed.

Chapter 7

The Instantaneous Topological Map

Our search for a neural network type capable of representing arbitrary topologies has turned up the Growing Neural Gas as the most promising model. But aside from this topological flexibility, the network must also be able to adapt well even with a strong serial correlation in the stimuli. Such correlations are found in robotics and in control systems in general, where stimuli are most naturally generated along trajectories in the input space.

In this chapter, we will first introduce the standard GNG algorithm by Fritzke, which we will enhance with a more flexible node creation mechanism. This network type performs acceptably well in a random walk testing scenario, but the performance can be improved drastically with the introduction of a different set of learning rules which form the Instantaneous Topological Map.

7.1 Improving the GNG for Correlated Stimuli

Our main interest is in situations where exploration of the state or feature space occurs along continuous trajectories, possibly with some moderate amount of superimposed noise. As our data model to mimic that situation we consider a (discrete) random walk with small step size d , given by

$$\begin{aligned}\vec{x}(t+1) &= \vec{x}(t) + \vec{p}(d, \alpha(t)) \quad \text{and} \\ \alpha(t+1) &= \alpha(t) + \eta,\end{aligned}\tag{7.1}$$

where $\vec{p}(d, \alpha)$ is the polar coordinate representation of a step of length d in the angular direction α , and η is a random variable. The step length d remains constant while the angle α changes by uniformly distributed random amounts η . Workspace limits are implemented by simply forbidding steps that lead outside of the allowed area (see figure 7.1 on page 69).

Collecting a large number of samples in this way produces an approximately even distribution of stimuli in the workspace area. The problem can therefore be made equivalent to an even distribution of stimuli by raising the typical timescale of the network's reactions, which in most cases can be achieved by lowering the learning rate. But since we wish to make the most of the incoming samples we need to perform *fast* adaptation, and therefore this simple trick is not an option here.

Before demonstrating the consequence of serial correlation in input stimuli, we wish to briefly recall the main ingredients of the GNG in order to provide the background for the following discussion and for the design of the ITM.

The basic GNG algorithm works on a set of nodes i , each represented by a weight vector w_i and an accumulated error e_i , and a set of edges j with an age value a_j . The adaptation with a new stimulus ξ consists of four distinct steps [8].

- 1. Matching:** Find the node n nearest to the stimulus ξ and the second-nearest node s .
- 2. Reference vector adaptation:** Given adaptation rates ϵ_1 and ϵ_2 , adapt the nearest node and its topological neighbors as follows:

$$\Delta w_n = \epsilon_1 (\xi - w_n), \quad \Delta w_i = \epsilon_2 (\xi - w_i) \quad \forall \quad i \in N(n), \quad (7.2)$$

where $N(n)$ denotes the set of neighbors of n .

- 3. Edge update:** (i) Create an edge connecting n and s if it does not already exist. Set that edge's age to zero. (ii) Increment the age of all other edges emanating from n and delete any whose age surpasses a given a_{\max} . When deleting an edge, check the other referenced node for emanating edges; if there are none, remove that node as well.
- 4. Node update:** (i) Increment the error measure of the nearest node:

$$\Delta e_n = \|\xi - w_n\|^2. \quad (7.3)$$

(ii) Add a new node every λ adaptation steps by finding the node q with maximum accumulated error and its neighbor r with maximum accumulated error:

$$q = \arg \max_i e_i, \quad r = \arg \max_{j \in N(q)} e_j. \quad (7.4)$$

Make a new unit s with $w_s = \frac{1}{2}(w_q + w_r)$ and initialize its error with e_q . Decrease the errors of q , r , and s by a given factor α . (iii) Multiply the errors of all nodes with a decay factor d , so that they cannot grow indefinitely.

The familiar matching and reference vector adaptation steps are the heritage of Kohonen's SOM. The vector adaptation differs slightly from the SOM algorithm in that in the SOM *every* vector in the network is changed using a Gaussian centered around n as an influence

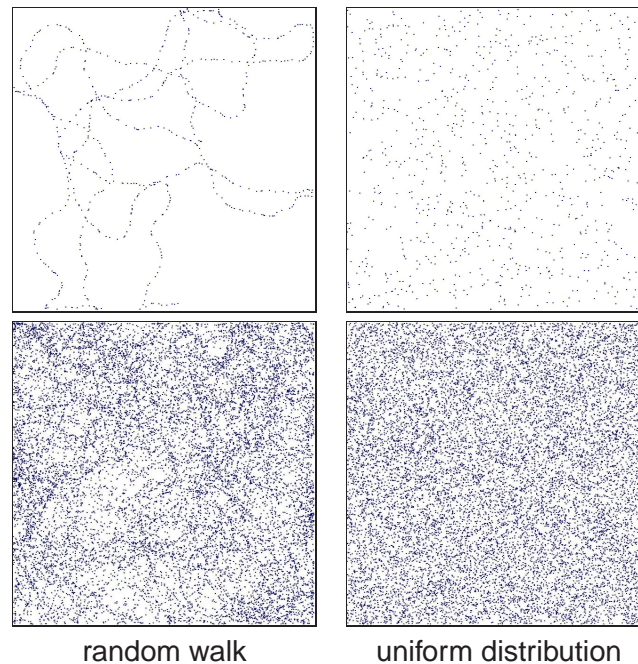


Figure 7.1: **Random Walk Example:** The figure to the left shows a typical random walk sequence produced by equation 7.1 on page 67 limited to a square play-field. An approximately uniform statistical stimulus distribution can be seen in the right panel. In the long term, the random walk sequence and the uniform distribution become nearly equivalent (lower panels).

function. Due to the GNG's topological complexity, this method would be computationally very expensive here. Therefore, the GNG adapts only n and its immediate topological neighbors. We will discuss the changing role of the vector adaptation in the ITM later on. Let us instead consider the topological adaptation steps in more detail.

The edge creation rule builds an edge of the Delaunay triangulation, given that the weight vectors of the nearest and second-nearest nodes are on opposite sides of the stimulus, because in this case, the nearest and second-nearest nodes always share a Voronoi cell border. An edge created in this way may become obsolete if new nodes are created or if the nodes move. The aging mechanism erases such edges eventually, but finding a suitable age limit can be difficult. The same limit may delete useful edges and still leave many useless ones untouched.

A noteworthy fact is that the construction of a Delaunay edge does not rely on the distribution of stimuli. Edges will be constructed in the fashion described even if stimuli touch only selected trajectories in input space. The triangulation is not guaranteed to be complete, but that is the case for both statistical and correlated series of stimuli: the Delaunay edges corresponding to shorter Voronoi borders are less likely to be constructed.

The node creation mechanism is less obvious and leaves more room for choice. The error accumulation provides a means of determining the optimum position for the creation of the

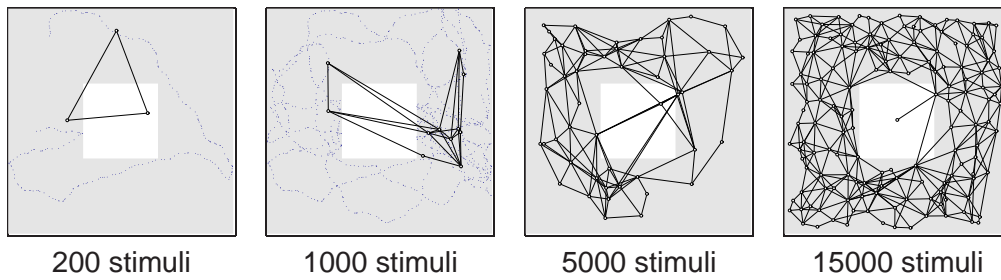


Figure 7.2: **Adaptation of a Standard GNG with Correlated Stimuli:** The network has been parameterized in such a way that the final result approximately matches that of the enhanced GNG and the ITM. The panels show intermediate stages of the training process. Especially during the starting phase, the standard GNG leaves large portions of the presented trajectory uncharted because of interference effects.

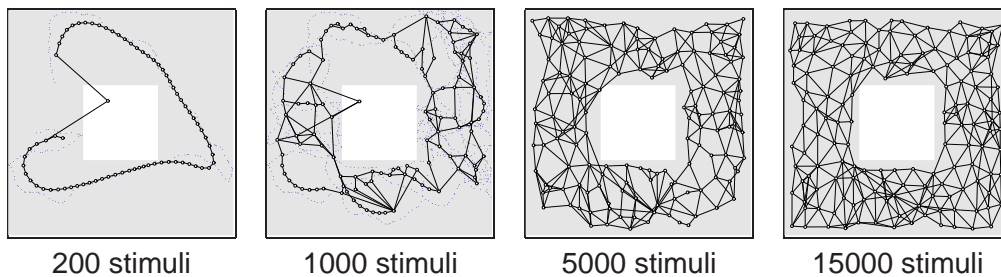


Figure 7.3: **Improved GNG Using an Error Threshold:** As an input, the same sequence as in figure 7.2 was used. In the startup phase, neurons are now created much faster to learn the trajectories traversed. As the error approaches the desired value, fewer new nodes are added.

next node. From a statistical point of view, this is a straightforward approach which leads to overall error minimization with a constant growth rate of the network. Notably, there is no provision for removing nodes except through the deletion of a node's last emanating edge.

When adapting to sequences of stimuli resembling trajectories, this node creation algorithm is not optimal, because topological disturbances can still be introduced by the fact that single nodes follow the trace of stimuli for long distances. To improve this behavior, we propose a modification of the node creation algorithm.

Our first approach to improving node creation works by defining a threshold value for the error, e_{\max} , with which the accumulated error measure of the nearest neuron, e_n , is compared. If it is larger, a new node is created between n and its neighboring node with highest error count. This small design change alone gives a dramatic improvement which can be appreciated by comparing the intermediate stages of the mapping task depicted in figures 7.2 and 7.3.

The modified algorithm proves to be more flexible than standard GNGs. Node creation now is a reaction to certain stimulus patterns, instead of being triggered by fixed external

clock cycles. The main advantage comes from switching from a global method, i.e., finding the node with highest accumulated error, to a local method. Designing the algorithm to only use the neighborhood of a node for adaptation keeps the computational effort low even for very large networks.

An added benefit of the threshold-driven node creation is its suitability for incremental learning. In the former approach, nodes are created at a constant rate, regardless of the stimulus pattern. In contrast, the threshold-driven approach automatically stops the creation of nodes if the input space is well covered by the network, and starts creating new nodes if stimuli appear in previously uncharted areas. This modification enables the GNG to react faster to its changing input and is therefore less sensitive to destructive interference.

7.2 The Instantaneous Topological Map (ITM)

There are still two disadvantages to keeping an error measure for each node and an age counter for each edge. (i) More parameters (error decay rate, error threshold, error distribution factor, maximal age) make optimizing a network more cumbersome. The parameter values are not very critical, but a wrong choice can still slow down the convergence considerably, or destroy it completely. Changes in the experimental setup almost always mean redesigning the parameterization of the GNG. (ii) Each slowly changing state variable (age and error count) introduced into the system produces some inertia, slowing down adaptation and defining a characteristic timescale which must be accounted for. The amount of time the network needs to react to changes in the input stimulus pattern depends very much on the choice of the corresponding decay factors.

We therefore propose a new network type which does not need any edge aging or error accumulation to generate its map. In fact, it does not even require node adaptation. We call it ITM, for “Instantaneous Topological Map”.

The ITM consists of a set of neurons i with weight vectors w_i , and a set of undirected edges, represented implicitly by specifying a set of node neighbors $N(i)$ for each node i .¹ The network starts out with only two connected nodes. The adaptation triggered by a new stimulus ξ consists of the following steps:

- 1. Matching:** Find the nearest node n and the second-nearest node s (with respect to a given distance measure $D(a, b)$, e.g., the Euclidean distance or the adaptive metrics introduced in section 6.6 on page 63).

$$\begin{aligned} n &= \arg \min_i D(\xi, w_i) \\ s &= \arg \min_{j, j \neq n} D(\xi, w_j) \end{aligned} \tag{7.5}$$

¹The $N(i)$ are further constrained by the requirement that neighborhood relations between a pair of nodes shall always be symmetric.

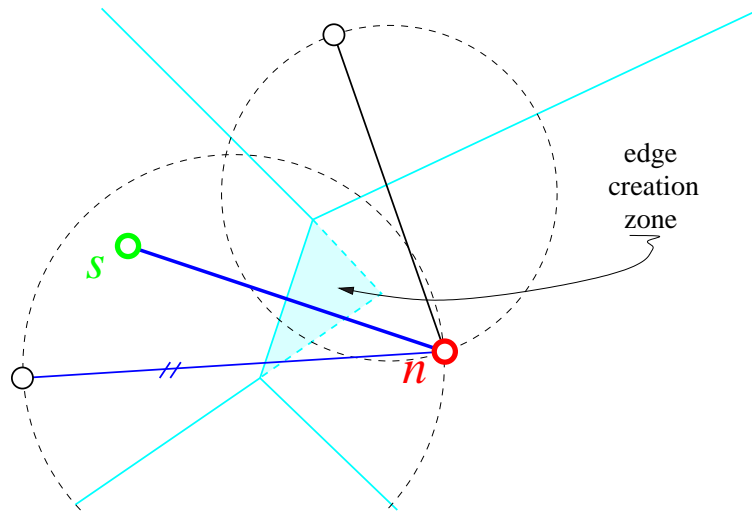


Figure 7.4: **Edge Update in the ITM:** Edge addition is triggered when a stimulus hits the grey region where the Voronoi cell of n intersects the Voronoi cell of s if n were not present. Removal of edges is triggered by the Thales sphere through n and one of its neighbors. If s lies inside that sphere, the corresponding edge is removed (the marked lower edge in the figure).

- 2. Reference vector adaptation:** Move the weight vector of the nearest node toward the stimulus by a small fraction ϵ . Below we will show that this step can even be omitted.

$$\Delta w_n = \epsilon (\xi - w_n) \quad (7.6)$$

- 3. Edge adaptation:** (i) Create an edge connecting n and s if it does not already exist. (ii) For each member m of $N(n)$ check if w_s lies inside the Thales sphere through w_n and w_m . If that is the case, remove the edge connecting n and m . When deleting an edge, check m for emanating edges; if there are none, remove that node as well (see figure 7.4). The testing formula is

$$\forall m \in N(n) \quad : \quad \text{if } (w_n - w_s) \cdot (w_m - w_s) < 0 \quad \text{then remove edge } \overline{nm} \quad (7.7)$$

- 4. Node adaptation:** (i) If the stimulus ξ lies outside the Thales sphere through w_n and w_s , and outside a sphere around w_n with a given radius e_{\max} , create a new node y with $w_y = \xi$. Connect nodes y and n .

$$\text{if } (w_n - \xi) \cdot (w_s - \xi) > 0 \quad \text{and} \quad D(\xi, w_n) > e_{\max} \quad \text{then create node at } \xi. \quad (7.8)$$

- (ii) If w_n and w_s are closer than $\frac{1}{2}e_{\max}$, remove s (see figure 7.5 on the facing page)².

²If using adaptive metrics, the scalar product in equations 7.7 and 7.8 must be replaced by the modified version as described in section 6.6 on page 63.

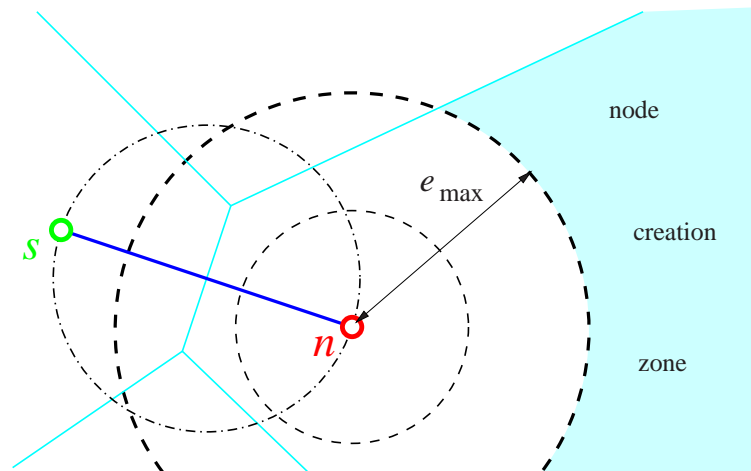


Figure 7.5: **Node Update in the ITM:** Node addition and removal in the ITM is guided by the Thales sphere through the nearest two nodes, n and s , and spheres through n of radius e_{\max} and $\frac{1}{2}e_{\max}$.

In terms of **computational expense**, the matching step is the only step that scales with the number of neurons. Edge adaptation scales with the average number of neighbors, which is related to the local intrinsic dimensionality of the input data. All other operations are independent of the number of neurons involved. This means that the algorithm executes fast even for large networks. The GNG with threshold-driven node creation is equally efficient, but the standard GNG requires searching for the optimal graph location each time a new node is created, which makes the algorithm slightly more costly.

Note that the search operations using $N(n)$ do not depend on the number of components of the feature vectors, but only on the “true” underlying dimensionality of the data. A two-dimensional distribution embedded in a higher-dimensional feature space will only produce four to six emanating edges per node on average, regardless of the feature vector size. The ITM shares this property with Fritzke’s GNG, who presents examples for this behavior in [8].

Our experience with the algorithm indicates that **reference vector adaptation** (step 2) can even be omitted because nodes are created and deleted swiftly if the node distribution is found to be too sparse or too dense. The former learning rate ϵ , which was essential to adjust the network to fit the input data, has now assumed the role of a smoothing parameter. Choosing small values of ϵ makes the nodes assemble slowly in a tidy arrangement with distances between nodes approximately equal. The relaxation time of this process does not affect the network’s overall performance (see figure 7.6 on the following page).

Edge creation produces a valid Delaunay edge, as stated before. This edge is then used to verify the other edges emanating from the nearest node. Only those edges are kept which cross the corresponding Voronoi cell border. This eliminates all non-Delaunay edges and few Delaunay edges, mainly those belonging to the convex hull. The advantage of this

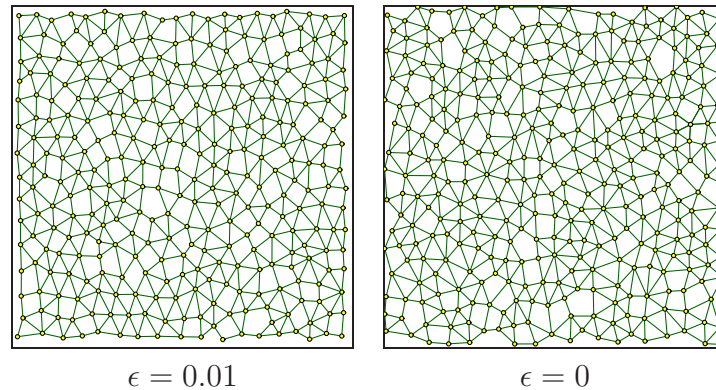


Figure 7.6: **Reference Vector Adaptation in the ITM:** In this experiment, a random walk trajectory inside a square region was used to stimulate an ITM. The left panel shows the result with $\epsilon = 0.01$, and the right panel the result with no reference vector adaptation at all. The difference between the two can barely be perceived, which demonstrates the changing role of the adaptation step; in the ITM, ϵ is an optional smoothness term which can slightly improve the node positions.

method compared to former edge deletion techniques is that it destroys all non-Delaunay edges and that it does not rely on parameter tuning to do so. An exhaustive Delaunay test which detects even small eccentric Voronoi borders between connected nodes would be computationally much more expensive, as the amount of calculations needed would scale with the dimensionality of the data *and* the total number of nodes.

Node creation avoids putting new nodes inside the Thales sphere through nearest and second-nearest node, because doing this would render useless the connection just made. If the stimulus lies farther away from the nearest node than a given threshold, a new node is created at the position of the stimulus. The threshold, e_{\max} , therefore has the meaning of a desired mapping resolution. This method is substantially different from providing a learning rate, as nodes are created at a maximum speed of one per stimulus if necessary, in which case the network stores the input data in weight vectors, and their closest neighbors in its graph. Because nodes can still move by a small amount in this algorithm, a criterion is provided to remove nodes that are too close to each other. The threshold used is derived from e_{\max} .

Configuring an ITM network is exceedingly easy, since only at most two parameters need to be found: the desired resolution e_{\max} , and, optionally, the smoothing parameter ϵ (the former learning rate).

7.3 Results

We use the random walk sequence of stimuli generated by equation 7.1 on page 67 to measure and compare the performance of the three network models just described. This

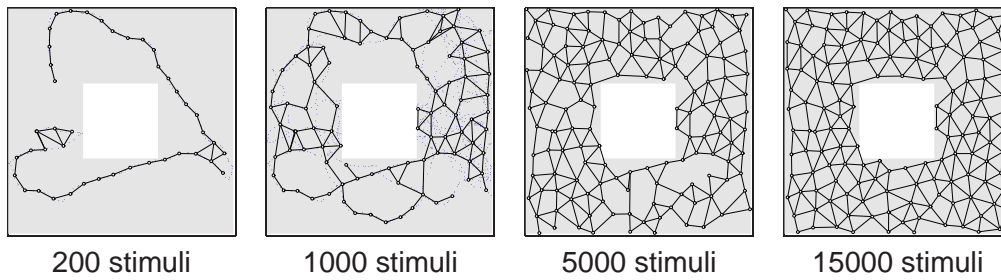


Figure 7.7: **Adaptation Phases of the ITM:** The ITM network generates a map from the same trajectory-like stimulus sequence as in figure 7.2 on page 70. The adaptation rate ϵ is not critical in this method, it can safely be set to zero.

sequence is the simplest model of an autonomous robot driving randomly through a room with a square obstacle in the middle. The objective is to map this room using a neural network.

Figures 7.2 and 7.3 on page 70, and 7.7 each show four phases in the adaptation of the standard GNG, the enhanced GNG (error triggered node generation), and the ITM, respectively. The network parameters are chosen so that each network arrives at approximately the same number of nodes at 15000 samples; in this way the intermediate phases can more easily be compared.

During the experiments, the number of nodes, the normalized root mean square error (i.e., the averaged value of $D(w_n, \xi)$), and the number of excess edges and missing edges with respect to the Delaunay triangulation were recorded (see figures 7.8 on the following page and 7.9 on page 77).

The ITM's normalized root mean square error (NRMSE) stays almost constant during training. This derives from its immediately creating nodes to achieve a desired precision. The slower error decay of the enhanced GNG originates from the inertia introduced by the error accumulators and the learning rate. The standard GNG is designed to slowly improve its mean error by adding nodes at regular intervals.

The edge creation and deletion algorithm's performance shows up when comparing the network's graph to the Delaunay triangulation. The edge aging mechanism of both GNG models is responsible for the high number of surplus edges, about ten to twenty percent, while the extremely strict immediate removal rule of the ITM lowers this number to almost zero. The small advantage of the enhanced GNG over the standard version can be explained with the better optimization of node placement. Each newly placed node makes some existing edges obsolete, and since standard GNG node creation frequency stays equally high throughout the experiment, the proportion of creating and deleting obsolete edges is less favorable.

The number of edges missing to complete the Delaunay triangulation cannot drop to zero in our experiment because of the obstacle in the middle of the imaginary room. Edge

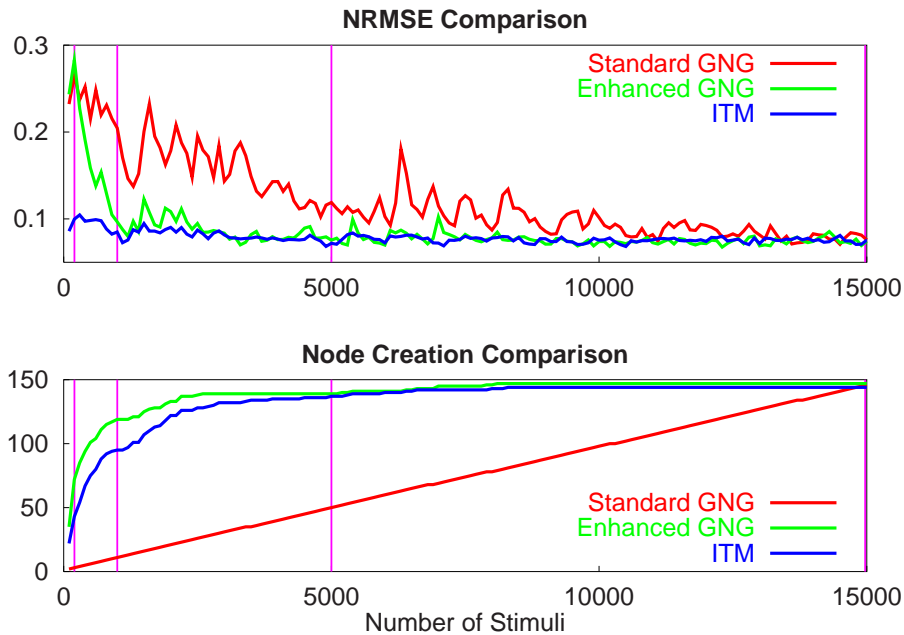


Figure 7.8: **Error Comparison of Three Network Models:** The graphs show measurements made on the three models discussed. The random walk stimulus pattern used can be seen in the figures 7.2 and 7.3 on page 70, and 7.7 on the page before, which show snapshots after 200, 1000, 5000, and 15000 samples (vertical lines in the graphs above). Although it creates nodes more slowly than the enhanced GNG, the ITM achieves the desired error from the start because it can create new nodes outside the current scope of the network.

creation functions by the same principle in all network models, so they perform almost equally well in this respect, with a slight disadvantage for the ITM. This is because of the simple but very strict immediate edge deletion technique used, which sometimes erases even valid Delaunay edges.

7.4 Statistical Distributions

Although we introduced and validated the ITM network model on the basis of trajectory-like series of stimuli, the ITM still performs very well in settings with statistically uncorrelated stimulus distribution. In these settings, too, the ITM can outperform the other network models in terms of convergence speed, because it has no inner state variables that can introduce inertial effects. Each adaptation step can produce and remove nodes and edges immediately, with no dependency on the network's past history.

As can be seen in the experimental example (see table 7.1 on page 78), the convergence behavior of the ITM does not degrade when using a statistical stimulus distribution. The intermediate stages show significant differences between the GNG and the ITM, because

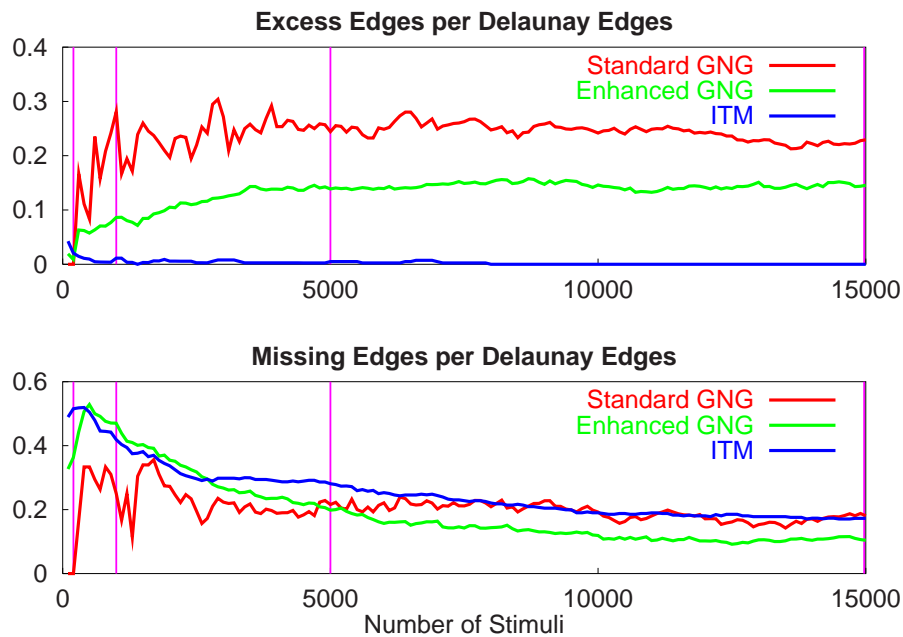


Figure 7.9: **Graph Comparison of Three Network Models:** The graphs were continuously compared to the Delaunay triangulation to gauge the quality of the connections created by the three network models discussed. The performance advantage of the ITM becomes apparent in the low number of excess edges which would carry no useful information for path finding. The underlying experiment is the same as in figure 7.8 on the preceding page.

the node placement principles are substantially different. The resulting map is nevertheless almost the same for the GNG and the ITM.

7.5 Architectural Comparison

The ITM is in some respects complementary to the SOM. While the SOM has rigid topology and relies on learning rate and smoothness parameter annealing to adjust the nodes' positions and map the underlying topology, the ITM has rigid node positions and generates the topology with adaptation rules. While the SOM does not depend on changing topology to produce useful mappings, but can benefit from such additions (as in the GNG), the ITM does not depend on changing the nodes' positions, but can benefit from a small smoothness term ϵ (the former learning rate).

One aspect of using the precision parameter, e_{\max} , instead of a SOM-type learning rate, is that nodes are always approximately equally spaced. The ITM model is not suitable for applications where the network's node density needs to be a function of statistical stimulus density. This famous property of the GNG and the SOM is rooted in the adaptation of the nearest node and its topological neighbors; therefore, it cannot be replicated in the ITM (see figure 7.10 on the next page).


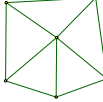
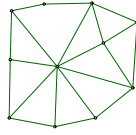
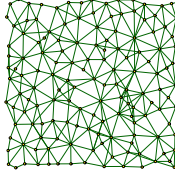
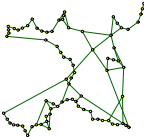
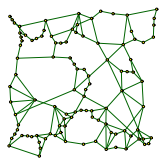
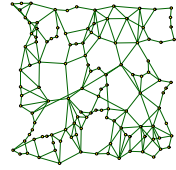
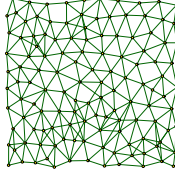
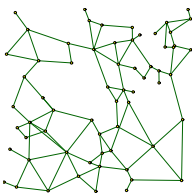
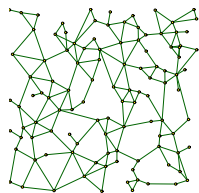
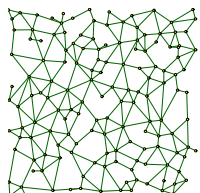
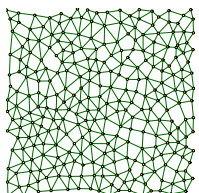
network type	200 stimuli	500 stimuli	1000 stimuli	15000 stimuli
GNG				
eGNG				
ITM				

Table 7.1: **Statistical Input Sequence Evaluation:** The table shows the performance of three networks given uniformly distributed input vectors in a square area. Although it has been specially designed for correlated stimuli, the ITM performs well when compared to the GNG or the GNG with enhanced node generation (eGNG). Especially in the intermediate stages the ITM is able to deliver a well-structured map of the stimuli it has seen so far, which contrasts to the relatively slowly emerging maps as formed by the other models.

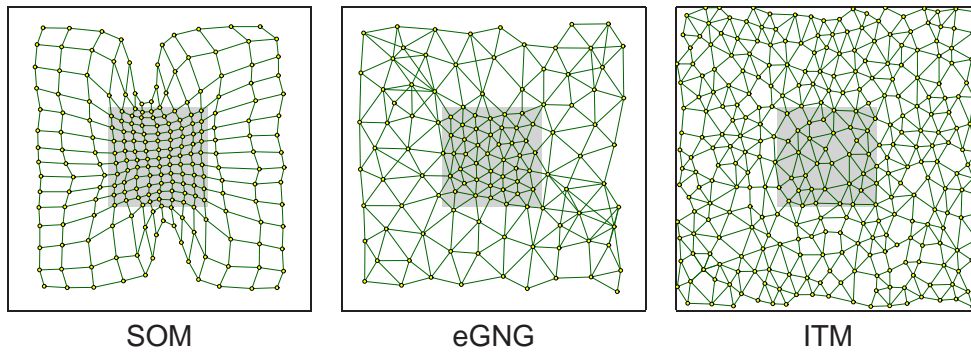


Figure 7.10: **Non-Uniform Stimulus Density:** The panels show the result of stimulation with an input distribution which has a central region with ten times the stimulus density of the outer area. The SOM and the GNG accumulate nodes in this central area, while the ITM remains completely oblivious of density variations. This also demonstrates that the SOM and the eGNG perform entropy maximization, while the ITM strictly minimizes the error.

GNG

parameter	symbol	value
nearest neighbor adaptation	ϵ_1	0.1
second nearest neighbor adaptation	ϵ_2	0.01
maximal edge age	a_{\max}	300
node creation interval	λ	1000
error distribution factor	α	0.5
error decay factor	d	0.999

ITM

parameter	symbol	value
resolution	e_{\max}	0.36
smoothness term	ϵ	0

Table 7.2: **Network Parameterization for the Dimension Test:** The GNG parameterization is very close to the defaults commonly used by Fritzke, the maximal edge age being the key parameter. The ITM parameterization disables reference vector adaptation, and sets a resolution which triggers the generation of sufficiently many nodes.

7.6 Dimensionality of Input Data

The experiments presented in this chapter involve two-dimensional input vectors with different topological structures for the clarity of the presentation. For higher-dimensional input data, the ITM can be expected to perform similarly well as the GNG, from which it is derived. It can also be expected to retain its fast and reliable mapping properties, and its advantage in terms of computational expense can even rise, because the strict edge deletion mechanism produces minimal neighborhood sets and consequently shorter testing loops for the edge update.

Our experiments compare the mapping of a standard GNG, which is known to produce faithful topological maps, and an ITM for different dimensional input stimuli in terms of the quality and completeness of the mapping. The stimuli are evenly distributed over an N -dimensional hypercube, and the networks are configured to generate enough nodes to obtain some in the center of the hypercube.

With rising dimension, the ratio of nodes on a hyper-face of the cube to the total number of nodes continuously rises, which makes it necessary to raise the number of nodes as well. In our experiments, the GNG was therefore configured to generate 4^N nodes, to make sure at least some nodes build the maximal number of neighborhood relations. The ITM, in contrast, does not need special configuration, because its resolution parameter automatically generates the required number of nodes, regardless of the dimension.

Table 7.2 shows the precise parameterization of the two networks used in the test. The parameters were left unchanged for all dimension values, except for the maximal number

dim	network type	# of nodes	# of edges	max # of neighbors	avg # of neighbors	total # of stimuli
2	GNG	16	35	7	4.4	14100
	ITM	21	47	7	4.5	6600
3	GNG	64	286	17	8.9	64100
	ITM	58	228	15	7.9	49000
4	GNG	256	2062	29	16.1	281400
	ITM	164	1140	29	13.9	283400
5	GNG	1024	12995	49	25.4	1086900
	ITM	419	4725	49	22.6	535000
6	GNG	4096	73694	75	36.0	4324400
	GNG ($a_{\max} = 700$)	4096	106266	100	51.9	1630900
	ITM	1099	19975	96	36.4	1940600

Table 7.3: **Dimension Tests on the ITM and the GNG:** The networks were given an evenly distributed input sequence in an N -dimensional unit hypercube. Although the number of nodes generated by the ITM is generally lower than the pre-defined number of nodes for the GNG, 4^N , the maximal number of neighbors indicates that both networks identify the dimensionality of the input space correctly. The average number of neighbors is an indicator for the amount of edge nodes which have lower neighbor counts than inner nodes. Since this factor is directly related to the total number of nodes, the network with higher final node count also yields a higher average neighbor count.

of nodes for the GNG. The adaptation was stopped if the network did not change substantially for at least 15000 steps. Some key figures that characterize the final network status were then recorded: the number of nodes, the number of edges, which together yield the average number of neighbors, and the maximal number of neighbors to a single node (see table 7.3).

In the table, data with N ranging from 2 upto 6 has been used. Higher dimensional input data yields rapidly growing networks, which has led to technical limitations above $N = 6$. In real-world applications, the size of the feature vectors is often much larger, but the intrinsic dimensionality is almost always much lower than the feature vector size. Therefore, the measurements have some significance for practical problems, although the dimension range tested is somewhat limited.

The GNG experiment with six-dimensional data shows the limitations of the edge aging algorithm. While the GNG with maximal edge age of 300 yields a maximal neighborhood count of 75, the ITM shows 96 neighbors at most. A further run with a maximal edge age of 700 raised the GNG's maximal neighborhood to 100, which confirms that edge aging is a rather fragile method of pruning unnecessary edges.

The measurements indicate that the ITM performs at least as well as the GNG when identifying the underlying dimensionality of the input data. Its greatest advantage is that it cannot be mis-configured to produce erroneous neighborhood relations. The GNG, in

contrast, requires more experience for the correct choice of its parameters. Especially too high or too low settings of a_{\max} , for example, can significantly alter the neighborhood relations and thus indicate different underlying dimensionality for the same data. These artifacts cannot be easily detected if the origin of the data is not known. The ITM can therefore be considered a much more reliable dimensionality analysis tool than the GNG.

7.7 Conclusions

The Instantaneous Topological Map produces reliable charts of trajectories without the need for special preparation of input samples. There are no delays in the construction of the map. These factors make the ITM especially useful in robotic control applications. A robot exploring its surroundings can store the topological data in an ITM, which then functions as an associative memory device. The robot can use this map literally to get from one location, represented by one node, to another: following the shortest path in the ITM's graph leads it to the target, automatically avoiding obstacles.

Many control processes involve finding an effective way of setting input values in order to reach a target output value with minimum effort. Using an ITM to map the state space while a simple controller is operating can turn up a more efficient pathway leading to the target position. Nodes along that pathway correspond to a series of target settings for the controller. Feeding that series instead of the final target values to the controller can lead to better overall performance.

The ITM was specifically designed for this family of problems, and we will follow this trail further in the following chapter. But it turns out that the ITM's remarkable properties may make it attractive for other areas, too.

One such area is data mining, a discipline which specializes in the retrieval of intelligible information from large unstructured data sets. One aspect of data mining is dimension reduction, which is closely related to the problem of feature vector extraction in neuro-informatics. Many projection algorithms have been devised to reduce the number of dimensions of a data set, mostly for visualization purposes.

The problem of dimensionality identification, on the other hand, has not been given much attention, although knowledge of the true underlying dimension of data delivers valuable information. Clustering techniques can help to identify topological properties of a data set, but the interpretation of cluster graphs alone cannot reliably deliver dimension information. The ITM is, to our knowledge, the first algorithm which locally identifies the intrinsic dimension of a data set by building a graph structure with reliable neighborhood information. The number of neighbors to a node can be correlated with the dimensionality of the data set at that position. This information is local up to a given mapping resolution, which defines the granularity of the map.

One should always bear in mind, though, that reliable dimension analysis requires many samples, as can be appreciated in table 7.3 on page 80 in the number of stimuli necessary to reach the saturation. The number of required samples also grows with the network size, which sets a natural limit for the granularity of the analysis. But in spite of these drawbacks, which originate from the very nature of the problem, the ITM promises to become a useful data mining tool in its own right.

Chapter 8

Path Finding and Obstacle Avoidance with the ITM

The ITM provides us with a means to map the state space of a robotic manipulator or another control system directly from the trajectory it traverses in a simple and efficient way. It is a small step to put this map to good use as a trajectory generator for the layered control system developed before.

Designing the feature vector of the ITM to contain portions (or all) of the parameter set of the low-level controller enables us to use the ITM in the same way as the state machine, with each node representing a state, and each edge representing a possible transition from one state to the other.

In this chapter, we will use a simple and efficient algorithm from graph theory for finding optimal pathways through the ITM's graph. Feeding a controller with the reference vectors of the intermediate nodes provides a simple trajectory generator which automatically performs obstacle avoidance for those obstacles present during training, because there are simply no traversable nodes in those areas.

8.1 Graph Distance Labeling

To perform efficient path finding, we will attach a “via vector” index to each node, which is the index of a neighboring node which takes us closer to the target node. To this end, each node must know its graph distance to the target. An algorithm which calculates these distances efficiently is readily available. In a slight modification, we implement it as follows.

Given a set M of nodes n , each with a set of neighbors $N(n)$, where each element of $N(n)$ is itself member of M , we define a target node t , and a graph distance d_n and a via v_n for each node in the graph¹.

Initialization: Initialize all via indices to point to nowhere, and the graph distances of all nodes except the target node to infinity. The target node's distance is initialized to zero.

$$\begin{aligned} d_n &= \infty & \forall n \in M \mid n \neq t \\ v_n &= \text{invalid} & \forall n \in M \\ d_t &= 0 \end{aligned} \quad (8.1)$$

Iterative distance construction: Scan through the nodes with a graph distance value less than infinity, and check each neighboring node. If a neighbor's graph distance is greater than the originating node's plus the length of the edge, then update the neighbor's graph distance and its via, and set a change signal. Repeat this step until a loop produces no change signal.

$$\begin{aligned} &\forall n \in M \mid d_n < \infty, \quad \forall j \in N(n) \quad : \\ &\text{if } d_n + l_{n,j} < d_j \quad \text{then} \\ &\quad d_j \leftarrow d_n + l_{n,j}, \\ &\quad v_j \leftarrow n \\ &\quad \text{set change signal.} \end{aligned} \quad (8.2)$$

This simple algorithm produces the desired distance labels and optimal pathways from any node to the target node t (see figure 8.1 on the next page).

One notable detail about the algorithm is that it does not require the via indicators to operate. We introduce these to optimize path generation. Also, the graph does not necessarily contain connections from or to all nodes. The distance of every node without connection to the target remains infinity, and the via indicator points nowhere. This information is useful to identify nodes belonging to different clusters (see figure 8.2 on the facing page).

8.2 Trajectory Generation

Using the labeled graph to generate a trajectory is a simple task given the controller infrastructure presented in the first part of this thesis. The state machine formalism allows us to expand the state graph with a "template" state, which takes the weight vector of a network node and converts it into a valid set of controller parameters. One of its transition criteria is fulfilled as soon as the controller reaches its target (a via point in the path representing the trajectory). The transition leads to a state which selects the next node on the trajectory, and the loop closes (see figure 3.1 on page 24, where these additional states can be seen in the upper portion).

¹The neighborhood relations $N(n)$ need not be symmetrical for this algorithm to operate, but we will employ it on the ITM, which does have symmetrical neighborhood relations.

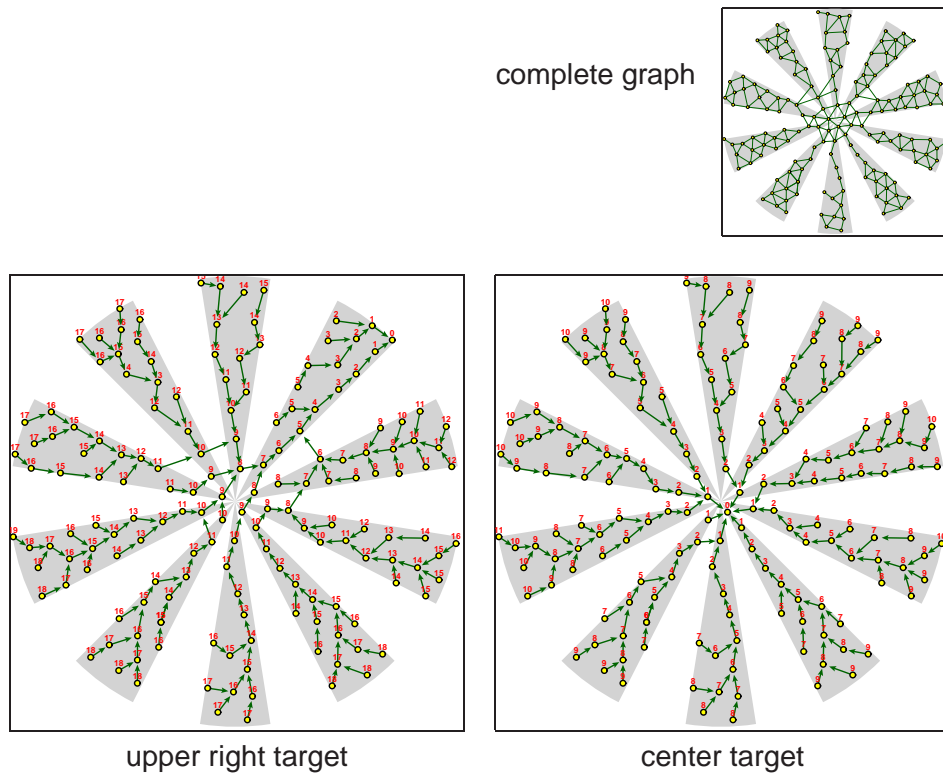


Figure 8.1: **Graph Distance and Path Generation:** The small insert shows an ITM created by stimulating with a random walk pattern inside a star shaped region. The lower panels visualize the result of the labeling algorithm for two different target nodes t . The node labels allow us to immediately find an optimal route from any node to the target node.

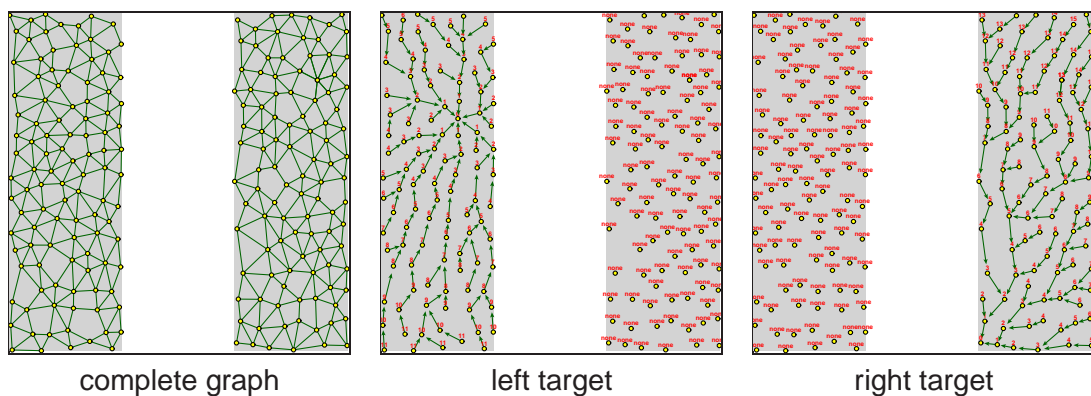


Figure 8.2: **Cluster Identification:** If no path exists from a node to the target node, the length indicates infinity (“none” in the graphs), and the via vector is invalid. This allows us to identify clusters in the graph by selecting a target node known to belong to that cluster and then applying the path finding algorithm.

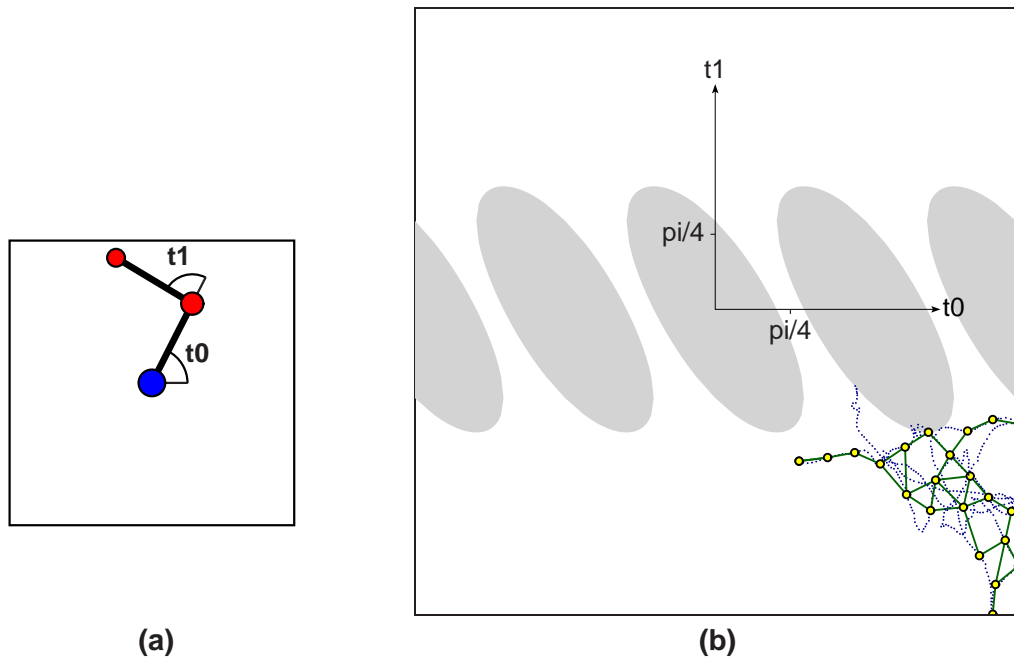


Figure 8.3: **Two-Dimensional Robot Simulation:** A two-joint robot arm (a) with unit segment lengths is confined by four walls 1.6 units from the robot base. The walls prohibit a large portion of the possible configurations (b), producing a complicated two-dimensional topology, which can be mapped, e.g., using an ITM. The four walls translate into ellipses in joint angle space, which repeat indefinitely along the θ_0 axis (labeled t_0 in the graph). Since θ_0 is limited to $[-\pi \dots \pi]$, we only see three ellipses and two half-ellipses.

8.3 Experimental Validation

As a toy problem, the prolific two-dimensional two-joint robot arm is simulated in real time, including centrifugal and coriolis forces, to make the stimulus patterns generated more realistic. The joint angles, θ_0 and θ_1 , are both limited to the $[-\pi \dots \pi]$ range by simulated bumpers, and the end effector is confined by four soft walls. The square of walls in Cartesian coordinates translates into elliptical regions in joint angle coordinates, which result a two-dimensional topology with three holes in joint angle space (see figure 8.3). We choose the segments of the robot to be 1 unit long, and the walls to form a square with each side measuring 3.2 units. In this way, we obtain a topology in joint angle space which leaves only four narrow paths for a configuration change², because the robot has to move into a corner to stretch out completely.

A simplified version of the controller architecture is used to maneuver the arm to specific

²When a robot can achieve the same end effector position from several different sets of joint angles, it is said to have several possible “configurations”. The two-joint robot arm presented here generally has two configurations for each end effector position, one with the elbow joint bent to the left ($\theta_1 > 0$), the other one with the elbow bent to the right ($\theta_1 < 0$). Because of the joint angle limits, the arm needs to stretch out completely to perform a configuration change.

joint angle configurations. As an example, figure 8.4 on the next page shows the graph formed by stimulating an ITM with a random walk pattern from the robot arm simulator, and the trajectory generated by a simple algorithm, which sets the controller target values to the weight vector of the node pointed to by the via index of the current nearest neighbor. In this example, the inertia of the robot arm sometimes hits a different node than the one targeted by the controller, but since the via indices are available for all nodes, this is not a major problem. The trajectory generator simply carries on, and finds its target node eventually.

This simple path following algorithm can be improved by tracking the manipulator motion and altering the edge weights ($l_{n,m}$) if some segments cannot be traversed with the respective controller settings. We implement this trajectory generator using the state machine, by adding three states, **start**, used by external applications to trigger the trajectory generator, **traveling**, used as a transitional state during the motion, **arrived**, to signal successful termination, and **stranded**, which signals that a path to the target node cannot be found. Details on the different states follow:

start: This state initializes an internal variable for the state **traveling**, the index i of the last node visited, to an invalid value. **start** always switches to **traveling**.

traveling: In violation of the strict Turing principle of the state machine, this state has an internal variable, i , and uses it to check several conditions of the trajectory traversal. The current best match node is denominated c .

1. If the current node's graph length is zero ($d_c = 0$), switch to **arrived**.
2. If the graph length has decreased ($d_c < d_i$), then check:
 - (a) If the node lies on the intended path ($c = v_i$), enforce the appropriate edge by decreasing its graph length $l_{i,c}$, but not below 1.
 - (b) If not ($c \neq v_i$), punish the appropriate edge by increasing its graph length l_{i,v_i} , but not above a given limit l_{\max} . If this limit is exceeded, remove the edge.

Set $i \leftarrow c$, choose the weight vector of v_c as new controller target, and re-enter **traveling**.
3. If the current node indicates no pathway to the target ($v_c = \text{invalid}$), switch to **stranded**.
4. If the current node c has not changed for more than a given amount of time t_{\max} , punish the appropriate edge length l_{c,v_c} as shown above, and re-enter **traveling**.

stranded, arrived: These are both terminal states, which can be used as success or failure signals.

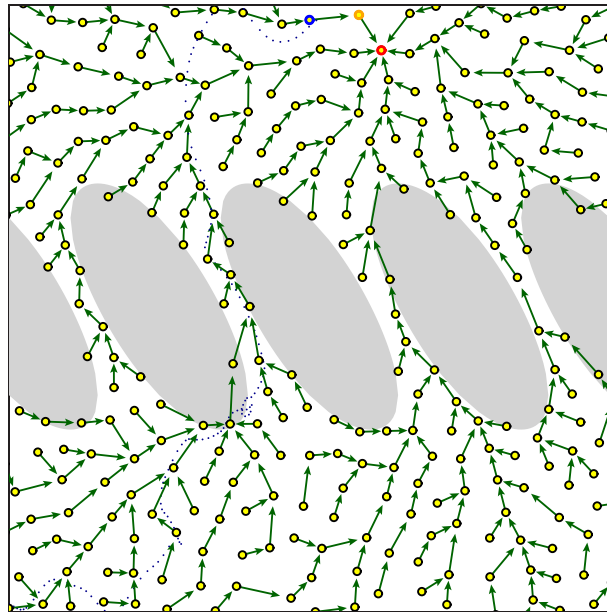


Figure 8.4: **Trajectory Generation Example:** Here, a simple trajectory generator maneuvers the manipulator to the target position by configuring the controller to move to the position of the node pointed to by the current node's via vector. The inertia of the manipulator forces it to leave the intended trajectory several times, but this simple mechanism recovers immediately because instead of one planned trajectory it knows pathways from all known configurations to the target, and therefore ultimately reaches its goal.

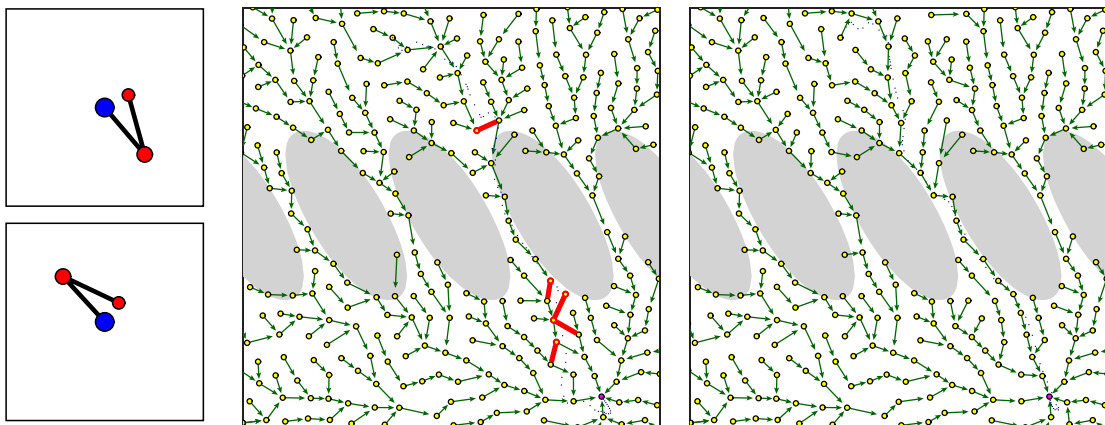


Figure 8.5: **Trajectory Reinforcement Experiment:** In the two-link robot scenario, we use the trajectory generation state machine to perform a configuration change. During the first run (middle frame), the dynamics of the robot throw it off the intended trajectory, as indicated by the thick lines, which show edge punishment events. After ten runs (right frame), the trajectory has been modified so that it can be traversed without problems.

This mechanism can change the route chosen by the graph labeling algorithm, and therefore has limited adaptive capabilities. After a few repetitions of the same motion, the manipulator improves the trajectory slightly (see figure 8.5 on the facing page).

Obviously, the algorithm described cannot cope with “orbiting trajectories”. This prolific problem in trajectory generation arises from the fact that due to acceleration limits, a manipulator usually needs to be slowed down in time before the end of the trajectory, or else it will oscillate around its target position for some time. Since the graph labeling procedure marks each node with its graph distance to the target, this deceleration scheme can easily be transported to the vector quantization state space representation.

Note that the trajectory generator operates with the edge lengths $l_{n,m}$, which we have introduced with the path labeling algorithm. These variables change slowly, and rise in number with the growth of the network. This results in a typical time scale for trajectory changes, which stands in contrast to the immediate adaptation behavior of the ITM. We accept this fact because of the drastic improvement we gain from edge labeling, and because an acceptable alternative has not yet emerged.

In this chapter, we have introduced a simple graph labeling algorithm which enables us to use our chosen method of state space representation, an ITM network, for generating trajectories from the current manipulator position to a target position defined by the index of the corresponding network node. The only ingredient still missing in the control system is a method for state space exploration. We will investigate this topic in the following chapter.

Chapter 9

An Active Exploration Engine

Building an input space representation with an ITM has already been performed in chapter 7 on page 67 based on random walk stimulus sequences. The experiments presented show that the network type chosen can adapt well with the serially correlated stimuli originating, for example, from a control system.

The random walk method can of course also be used with robotic manipulators, but in real-life control systems the amount of time taken for exploration is a critical issue, and as far as efficiency goes, random walk marks the bottom line. In the present chapter, we will therefore develop an active exploration mechanism based on several simple optimization heuristics, and evaluate its performance.

9.1 Basic Ingredients

Even without the aid of our mapping and trajectory generation mechanism, we could improve the basic random walk approach by replacing it with a “greedy” version. Instead of randomly varying the direction of travel in each time step, we keep the same direction until no further motion in that direction is possible, and then switch to another random direction. A similar method is known in function minimization, where the direction switching condition is met when the function value cannot be lowered further.

In our scenario, implementing the greedy algorithm involves setting target positions for the controller layer and reacting on sensory feedback to determine when to switch the direction. We will employ the state machine based trajectory generator to achieve this goal.

Using the trajectory generator and the current topological map enables us to further improve the algorithm. We restrict a random node choice to those nodes with less-than-average number of neighbors, and then start a series of greedy steps from that node. The

nodes chosen in this way are likely to belong to the topological edge of the map built so far, and are therefore suitable starting points for further exploration.

In the implementation of this scheme, we attach the exploration algorithm directly to the trajectory generation scheme presented in the preceding chapter. As additional static information, we need to store the current exploration direction p , a vector of random direction with a length of $1.2 \cdot e_{\max}$.

arrived: We add a rather complicated transition criterion to this terminal state, which always switches to **start** after choosing a new exploration target along a carefully chosen direction.

1. Build a tentative target position t by adding p to the current node's weight vector w_c .
2. Find the node n closest to t . If it is different from c , this indicates that the network is already present in the area to be explored, so a new direction has to be chosen by randomizing p and re-entering step 1. Repeat this only a limited number of times to avoid deadlocks.
3. Check the validity of the new target by stimulating the network with t and observing if this produced a new tentative node which can be used as a trajectory target. If no valid target t could be found even after iterating the above steps several times, choose a new edge node c at random among those with lower-than-average number of neighbors. Repeat the search at step 1. Limit the number of repetitions to avoid deadlocks.
4. As a last resort, choose a random edge node as described above and take it as a new target. This is a fallback solution, which should never be necessary under normal conditions.

This construction produces a new node in a previously uncharted area in almost all situations. The trajectory generator subsequently attempts to maneuver to that node's position and will either reach it and re-enter this target chooser or remove it, thereby entering the **stranded** state.

stranded: Entering this state in most cases means that a chosen direction has led to the edge of the allowed state space, and therefore a direction change (new random selection of p) is forced before entering the same procedure as for the **arrived** condition.

This algorithm has a strong tendency to explore the edge of the already charted area, and even sometimes neglects the inner regions of the map. In this respect, the explorer is fundamentally different from random walk and other methods which lack knowledge of the previous moves. We shall see this in the following comparison.

9.2 *Results*

To evaluate the explorer's performance, we experiment with the simulated two-link robot arm confined by a square of walls, as introduced in chapter 8. The random walk is implemented by randomizing the forces exerted on the joints, while the explorer uses the controller layer and trajectory generator also introduced in the aforementioned chapter.

Figure 9.1 on the following page shows a typical learning sequence with random walk and active exploration at equal numbers of stimuli presented.

In this experiment, the active exploration algorithm finds the crucial aspect of the underlying topology, a pathway to the second configuration, considerably faster than random walk. Both methods have to face the difficulty that the pathways from one configuration to the other are narrow, but the greedy steps of the active explorer are more likely to find one of these paths. Additionally, since the active explorer uses its map to find uncharted areas, it can also take advantage of the first pathway it finds, and use it to quickly build thorough maps of both configurations, while the random walk still relies on luck to flip from one half-space to the other.

The aggressive node placement of the explorer produces a map which is partly unreliable because some nodes lie outside the allowed areas and some edges cross forbidden areas. The trajectory generator must compensate this effect by removing those unusable edges when the need arises.

An alternative would be to mark such nodes as unreachable and leave them in the graph to make the search for new uncharted areas more efficient. But this method would conflict with the fact that forbidden areas may change in shape over time. The exploration algorithm must not introduce static information about reachable or unreachable areas, because the obstruction might be temporary.

The algorithm presented here does not introduce any new static variables into the system. Although the search for a new tentative node may take several iterations, it takes place in an instantaneous step compared to the trajectory generation and manipulator motion.

This exploration engine completes the ensemble of algorithms which work on a state space representation with an ITM vector quantization network. But although the system is completed with this step, there is some interplay of the individual elements which we will illuminate in the final chapter.

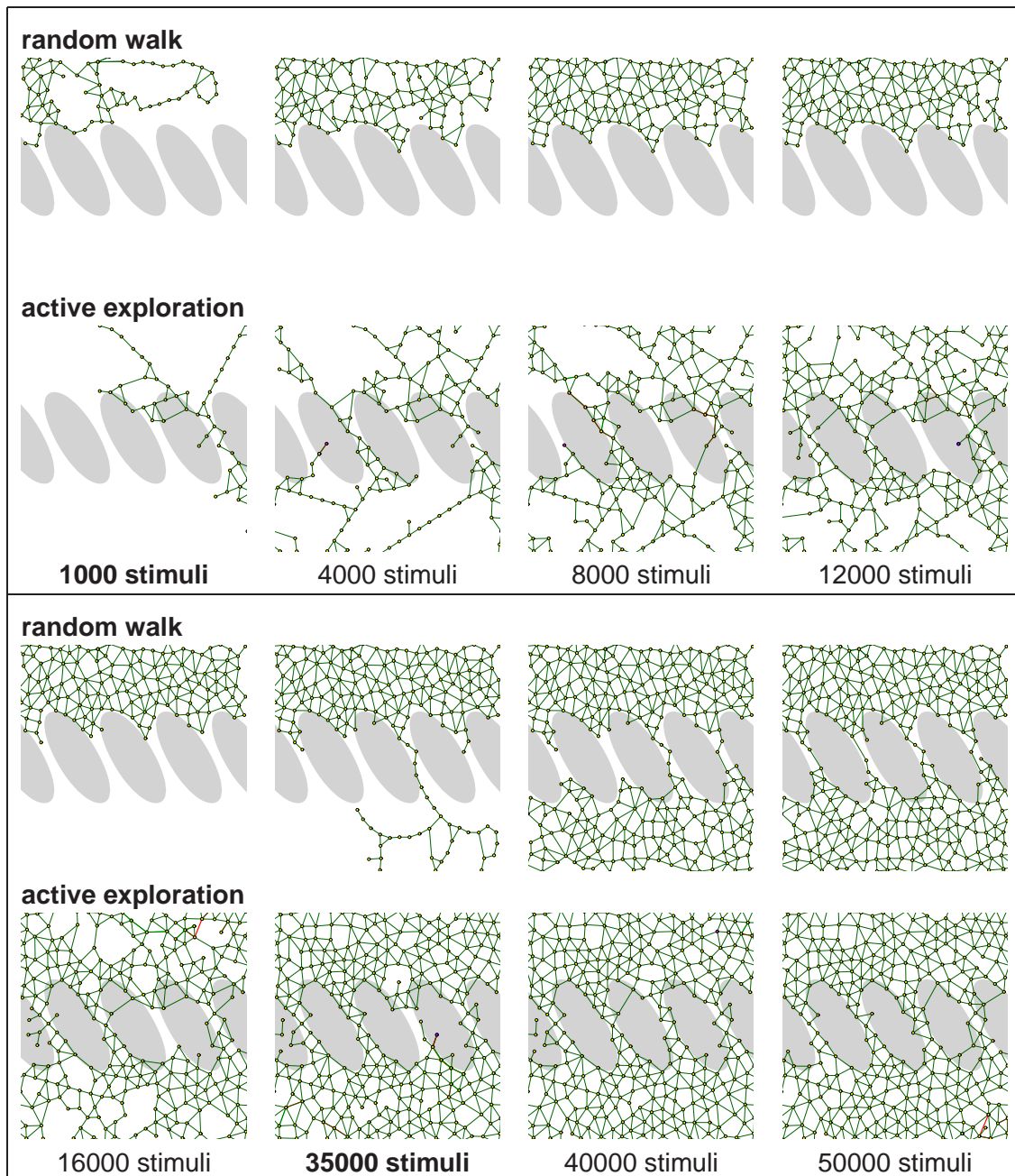


Figure 9.1: Active vs. Random Walk Exploration: The simulated two-dimensional robot problem introduced in the previous chapter is used to compare random walk exploration and the novel active exploration scheme. This exploration method discovers the second configuration after about 1000 stimuli, by crossing through one of the narrow channels where the arm can stretch out completely. The random walk algorithm, in contrast, is stuck in one configuration for about 35000 stimuli. The active explorer produces a network which reaches farther into the wall areas than the one delivered by random walk, because the explorer tentatively places nodes in uncharted areas, and subsequently removes them if they cannot be reached.

Chapter 10

Discussion and Outlook

The objective of the work presented in this thesis was to provide an architecture for a robotic control system suitable for human-machine interaction, and to use this architectural infrastructure to provide integrated solutions for higher-level tasks like trajectory generation, obstacle avoidance, and exploration.

An approach for state space representation with vector quantization networks has been presented, which is a fundamental ingredient of the higher-level tasks listed above. To this end, a novel type of neural network, the Instantaneous Topological Map, was developed, which in contrast to most other vector quantization approaches uses strict geometrical rules for adaptation. This network type is exceedingly easy to configure, fast both in terms of adaptation behavior and of computing expense, and capable of handling strong correlations in the stimuli.

As a proof of concept, the human-machine communication features were embedded in a larger research context, the SFB 360, and the state space mapping algorithms were extensively tested in artificial toy problems and in a simulated two-link robot arm scenario.

There are many ramifications of the research topics touched in this work, which could not all be mentioned before. Let us take the time now to investigate the consequences of the system design presented here, and the perspectives for future research work.

10.1 Uses Beyond Robotics

The controller infrastructure we have presented in the first part of this thesis has been especially developed for the TUM hand in our laboratory. Nevertheless, the architecture can be used in other settings as well. Even seemingly simple control problems may profit

from the methods shown, e.g. if they have several control outputs. In those cases, many pathways from one set of targets to the other are possible, and finding the optimal pathway could be achieved with either the state machine formalism or the topological mapping and trajectory generation method. And although examples originating from mechanics or kinematics are most easily imaginable, other control systems, e.g. chemical process control, can in principle possess a similar structure.

In producing distributed systems, one optimization issue is reducing the amount of data used in communication between the system's components. In this area of research, programming frameworks similar to the state machine shown here have already received much attention. Our motivation for the use of a state machine was simplicity and intuitiveness of the interface. The fact that this interface produces only a minimum of bandwidth is additionally beneficial.

10.2 Future Perspectives of the Control Architecture

External Trajectory Reinforcement

The trajectory planner introduced here is already capable of adapting to the dynamic properties of the underlying system by reinforcing those trajectories which are most easily followed by the controller layer. But the reinforcement is not restricted to inertial effects or obstacles that forbid a certain motion. External interference, e.g. by the human operator, can also influence this mechanism, providing a simple means of teaching motion patterns.

Teaching Motions with Labeled Nodes

Along with the prospect of labeling nodes in an intuitive fashion, this might lead to a versatile motion teaching and reproduction system. In such a system, a target node would be labeled with a certain name, and the trajectory generator would attempt to reach that position. The human operator would distort the trajectory as intended, and after some repetitions the system would be able to reproduce this motion.

One remarkable feature of this approach is that the individual trajectories share one state space representation, and therefore segments of individually taught motions can be automatically concatenated to build more complex motion patterns.

Dynamics Modeling with a Directional Graph

Even the simple dynamics of the two-link robot arm used for the validation of the neural layer suggest that the non-directional graph used in the ITM may be too limited for complex control systems. Some concepts for converting the ITM maps into directional graphs have been developed, and the most promising method involves symmetric edge creation and deletion, as already implemented in the ITM, but direction-dependent edge labeling for use in path finding and trajectory generation.

Travel Time Labeling

Especially for robotic control, the interpretation of the edge length $l_{n,m}$, introduced in chapter 8, as a traveling time from node to node may become a useful addition. In the present work, the lengths are abstract integer numbers which are used for path reinforcement. The literal meaning of travel times along with direction-dependent labeling can provide valuable additional information in the topological map. For example, this information could allow an improved trajectory generation mechanism to avoid orbiting around the target position by decelerating the manipulator in time.

Trajectory Comparison Metric

For trajectory planning, the graph distance and its possible interpretation as a travel time are most relevant. In comparing two trajectories and evaluating their similarity, a distance measure for trajectories in the graph can quickly become necessary. We suggest a preliminary representation of trajectories as a series of node positions. The Levenstein Distance Algorithm (LDA) could then be used to compare such series of nodes by performing pattern matching to find surplus, missing and differing node entries, and weighting these three classes independently to find a meaningful scalar trajectory distance.

This simple definition does not account for the identification of trajectories contained in a larger trajectory, but it suffices to evaluate a trajectory generator by comparing its result with a previously defined trajectory. Learning algorithms may take advantage of the metric as an error feedback or as a reinforcement critic. Therefore, we think that the well-designed definition of such a trajectory comparison metric will help to perfect related algorithms.

Tactile Object Recognition

The mapping procedures shown in this work perform obstacle identification indirectly, by only adding traversable state space areas to the map. For permanent obstacles, this is a

feasible procedure, but short-term disturbances and obstacles need special treatment. In this case, the predominant problem is the distinction of objects, possibly including the human operator's hand and other highly transitional interferences.

A trajectory recognition mechanism could be a first step to solve this identification problem. Identifying an object through a certain motion sequence can become a successful object recognition method. Especially in tactile sensing research, systematically tracing an object with a robotic finger has already been performed, but a system based on the control and exploration techniques presented here, which adapts to find suitable tracing motions to discriminate a set of objects, would require much less a-priori knowledge.

10.3 Future Perspectives for the ITM

Graph Visualization for the ITM

The ITM's capability of reliably identifying the underlying dimensionality of the input data can inspire the use of graph construction techniques in data visualization. But since the graph itself becomes complex in higher dimensions, there is no obvious method of intuitively visualizing the graph structure.

Helge Ritter has introduced a very practical and intuitive projection method for his HSOM, which "zooms in" on one node and its immediate neighbors, and contracts the distances of nodes farther away, so that the infinite horizon lies on a circle around the current focus. This projection technique may be modified to accommodate arbitrary graphs, so that an abstract visualization of the connected clusters formed by an ITM could be designed. This would make it easier for researchers to use the ITM both as a dimensionality identification and as a clustering method, as shown in figure 8.2 on page 85.

Associative Completion

One widespread application of vector quantization networks involves their use as associative memory devices. This is commonly done by masking some of the input vector's components when calculating the distances to find the best matching node. This node's weight vector can then be used to complete the input vector.

The same approach can become useful in the ITM in conjunction with the control architecture. In a setting where the controller output is only indirectly coupled to the state variables comprising the ITM's feature vector, the present architecture cannot maneuver along intended trajectories reliably. Adding the controller output as a feature component

can eliminate this shortcoming. During training, the additional information is fed into the ITM along with the normal feature vector, and during matching the controller output can be found by associative completion as described above.

One should bear in mind, though, that adding more entries to the feature vector can delay the adaptation process considerably, because the dimensionality of the feature space rises. The drastic consequences of this move can be appreciated in the dimension analysis in chapter 7. But since this analysis also indicates that the ITM stands good chances of mapping higher dimensional feature spaces faster than comparable algorithms, the inherent risk of raising the feature vector's dimensionality is relatively low.

Eliminating Quantization Effects

When using associative completion in a vector quantization network, the quantization error becomes especially apparent, because even the known components of the input vector “snap” to those of the closest weight vector stored in the network. Some network types which alleviate this problem with interpolation approaches have been mentioned in chapter 6.

Transferring the LLM algorithm to the ITM may prove beneficial, for instance in a setting where controller outputs must be delivered by the neural network, as described above. The linear maps attached to each node would require more training examples than the plain ITM, but the subsequent ability to produce output which reacts more sensitively on input vector changes, i.e., without visible quantization, could make the longer training phase worthwhile.

10.4 Closing Remarks

Previous robotics research was a strong inspiration for the present work. But observing the developments of the past years, a slight stagnation becomes apparent. Many tough problems have been successfully solved, while others remain out of our reach, and we may consider ourselves lucky if we at least find out why they are so elusive. A novice researcher might feel discouraged at the sight of both the remarkable achievements and the seemingly impossible unsolved problems.

With this work, we hope to prove that robotics research still has much to offer. Especially in joining neuro-informatics and robotics there are many architectural designs that have yet to be tested in the real world, and probably at least as many that have yet to be invented. And, as this thesis shows, the challenge lies not only in employing already

available algorithms in a new area, e.g. by transferring neural networks to robotics or other control tasks, but also in advancing new developments in *both* areas to find new solutions.

The neural networks community makes a considerable effort to test emerging methods in real-life settings. The standardization of typical neural networks problems has enabled us to compare many algorithms impartially. But this standardization can also narrow our view, keeping us from considering a special problem, although it might well stimulate the development of an algorithm which is generally useful. In our opinion, the ITM is an example of a general algorithm which emerged from a very special design goal. The departure from standard paradigms favored this development, which benefits both research fields involved.

This work took us on a journey through many fields, and many thoughts could only be touched very briefly. We hope that the reader may accept this open-endedness and understand it as an encouragement and as an invitation which robotics, human-machine interfacing, and neuro-informatics hold for all of us.

Bibliography

- [1] AMP Incorporated, Valley Forge, PA 19482. *Piezo Film Sensors Technical Manual*, Dec. 1993.
 - [2] D. H. Ballard, M. M. Hayhoe, and P. K. Pook. Deictic codes for the embodiment of cognition. Technical report, University of Rochester, 1995.
 - [3] G. Canepa, M. Campanella, and D. De Rossi. Slip detection by a tactile neural network. In *Proceedings of the ICIROS'94*, volume 1, pages 224–231, 1994.
 - [4] P. Dario. Tactile sensing: Technology and applications. *Sensors and Actuators*, A(25-27):251–256, 1991.
 - [5] P. Dario, A. Sabatini, B. Allotta, M. Bergamasco, and G. Buttazzo. A fingertip sensor with proximity, tactile and force sensing capabilities. In *Proceedings of the ICIROS'90*, pages 883–889, 1990.
 - [6] A. Fagg, N. Sitkoff, A. Barto, and J. Houk. Cerebellar learning for control of a two-link arm in muscle space. In *Proceedings of the ICRA'97*, volume 3, pages 2638–2644, 1997.
 - [7] B. Fritzke. Growing cell structures — a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.
 - [8] B. Fritzke. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 7:625–632, 1995.
 - [9] B. Fritzke. A self-organizing network that can follow non-stationary distributions. In *Proceedings of ICANN'97*, pages 613–618. Springer, 1997.
 - [10] B. Fritzke. *Vektorbasierte Neuronale Netze*. Shaker Verlag, 1998.
 - [11] K. S. Fu, R. C. Gonzales, and C. S. G. Lee. *Robotics, Control, Sensing, Vision, and Intelligence*. McGraw-Hill International Editions, 1987.
 - [12] G. Heidemann. *Ein flexibel einsetzbares Objekterkennungssystem auf der Basis neuronaler Netze*. PhD thesis, Technische Fakultät, Universität Bielefeld, 1998.
-

-
- [13] R. D. Howe and M. R. Cutkosky. Dynamic tactile sensing: Perception of fine surface features with stress rate sensing. *IEEE Transactions on Robotics and Automation*, 9(2):140–150, April 1993.
 - [14] W. Ilg, T. Mühlfriedel, and K. Berns. A hybrid learning architecture based on neural networks for adaptive control of a walking machine. In *Proceedings of the ICRA'97*, volume 3, pages 2626–2631, 1997.
 - [15] Interlink Electronics, Europe, Echternach, G.D. de Luxemburg. *The Force Sensing Resistor*, Feb. 1990.
 - [16] J. Jockusch. Taktile Sensorik für eine Roboterhand. Master's thesis, Technische Fakultät, Universität Bielefeld, 1996.
 - [17] J. Jockusch and H. Ritter. An instantaneous topological mapping model for correlated stimuli. In *Proceedings of the IJCNN'99*, 1999. paper #445.
 - [18] S. Jockusch. *Modellierung und Manipulation von Bild- und Grafikdaten mit neuronalen Netzen*. Dissertation, Technische Fakultät, Universität Bielefeld, 1995.
 - [19] J. Jockusch et. al. A tactile sensor system for a three-fingered robot manipulator. In *Proceedings of the ICRA'97*, volume 4, pages 3080–3086, 1997.
 - [20] S. Jung, T. C. Hsia, and R. G. Bonitz. On robust impedance force control of robot manipulators. In *Proceedings of the ICRA'97*, volume 3, pages 2057–2062, 1997.
 - [21] N. Jungclaus. *Integration verteilter Systeme zur Mensch-Maschine-Kommunikation*. Dissertation, Technische Fakultät, Universität Bielefeld, 1998.
 - [22] N. Jungclaus, R. Rae, and H. Ritter. An integrated system for advanced human-computer interaction. In *UCSB-Workshop on Signals and Images (SIPL)*, pages 93–97, 1998.
 - [23] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.
 - [24] K. Liano. Robust error measure for supervised neural network learning with outliers. *IEEE Transactions on Neural Networks*, 7(1):246–250, 1996.
 - [25] H. Liu, P. Meusel, and G. Hirzinger. A tactile sensing system for the DLR three-finger robot hand. In *Proceedings of the ISMCR'95*, pages 91–96, 1995.
 - [26] J. Lloyd and V. Hayward. *RCCL/RCI System Overview*. McGill Research Centre for Intelligent Machines, McGill University, Aug. 1988.
 - [27] T. M. Martinetz and K. J. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.
-

-
- [28] B. H. Mehler. Entwicklung eines taktilen ortsauflösenden Sensorsystems zur Unterstützung des Greifens mit Robotern. Master's thesis, Lehrstuhl für elektrische Meßtechnik, TU München, 1994.
- [29] R. Menzel, K. Woelfl, and F. Pfeiffer. The development of a hydraulic hand. In *2nd Conf. on Mechatronics and Robotics*, pages 225–238, 1993.
- [30] A. Meyering and H. Ritter. Learning 3D-shape perception with local linear maps. In *International Joint Conference on Neural Networks '92, Baltimore*, 1992.
- [31] W. T. Miller, III, R. S. Sutton, and P. J. Werbos, editors. *Neural Networks for Control*. MIT Press, 1990.
- [32] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
- [33] R. Rae, M. Fislage, and H. Ritter. Visuelle Aufmerksamkeitssteuerung zur Unterstützung gestikbasierter Mensch–Maschine Interaktion. *KI – Künstliche Intelligenz, Themenheft Aktive Sehsysteme*, 99(1):18–24, March 1999.
- [34] R. Rae and H. Ritter. 3d real-time tracking of points of interest based on zero-disparity filtering. In S. Posch and H. Ritter, editors, *Workshop Dynamische Perzeption*, Proceedings in Artificial Intelligence, pages 105–111, 1998.
- [35] R. Rae and H. Ritter. Recognition of human head orientation based on artificial neural nets. *IEEE Transactions on Neural Networks*, 9(2):257–265, March 1998.
- [36] S. Rankers. Steuerung einer hydraulisch betriebenen Roboterhand unter Echtzeitbedingungen. Master's thesis, Technische Fakultät, Universität Bielefeld, 1994.
- [37] R. P. N. Rao and D. H. Ballard. An active vision architecture based on iconic representations. Technical report, University of Rochester, 1995.
- [38] H. Ritter. Learning with the self-organizing map. *Artificial Neural Networks*, 1:379–384, 1991.
- [39] H. Ritter. Self-organizing maps in non-euclidean spaces. In *WSOM'99 Conference Proceedings*, 1999. (invited paper).
- [40] H. Ritter, T. M. Martinez, and K. J. Schulten. *Neural Computation and Self-Organizing Maps*. Addison-Wesley, 1992.
- [41] H. J. Ritter. Parametrized self-organizing maps. In *Proceedings of ICANN'93*, pages 568–575. Springer, 1993.
- [42] D. Selle. Realisierung eines Simulationssystems für eine mehrfingerige Roboterhand zur Untersuchung und Verbesserung der Antriebsregelung. Master's thesis, Technische Fakultät, Universität Bielefeld, 1994.
-

- [43] H. Shinoda, K. Matsumoto, and S. Ando. Acoustic resonant tensor cell for tactile sensing. In *Proceedings of the ICRA'97*, volume 4, pages 3087–3092, 1997.
 - [44] H. Shinoda, N. Morimoto, and S. Ando. Tactile sensing using tensor cell. In *Proceedings of the ICRA'95*, volume 1, pages 825–830, 1995.
 - [45] H. Shinoda, M. Uehara, and S. Ando. A tactile sensor using three-dimensional structure. In *Proceedings of the ICRA'93*, volume 1, pages 435–441, 1993.
 - [46] S. Sur and R. M. Murray. An experimental comparison of tradeoffs in using compliant manipulators for robotic grasping tasks. In *Proceedings of the ICRA'97*, volume 2, pages 1807–1814, 1997.
 - [47] M. E. Tremblay and M. R. Cutkosky. Estimating friction using incipient slip sensing during a manipulation task. In *Proceedings of the ICRA'93*, volume 1, pages 429–434, 1993.
 - [48] G. Wöhlke. The Karlsruhe dextrous hand: Grasp planning, programming and real-time control. In *Proceedings of the ICIROS'94*, volume 1, pages 352–359, 1994.
-