Visuo-motorische Koordination eines Industrieroboters

und

Vorhersage chaotischer Zeitserien

Zwei Anwendungen von selbstlernenden, neuronalen Algorithmen

Jörg Walter

Visuo-motorische Koordination eines Industrieroboters

und

Vorhersage chaotischer Zeitserien

Zwei Anwendungen von selbstlernenden, neuronalen Algorithmen

Diplomarbeit

von

Jörg Walter

Physik–Department
der
Technischen Universität München
Institut T30
Prof. K. Schulten, Ph.D.

Februar 1991

Inhaltsverzeichnis

Kapitel 1

Einleitung und Überblick

Die rasante Entwicklung des Computers hat dem alten Menschheitsdrang nach dem Verständnis von „Geist“ und „Intelligenz“, sowie der Funktionsweise unseres Gehirnes, neue Hoffnung gegeben. Der Computer hat in der Tat eine große Beschleunigung des wissenschaftlichen Fortschrittes ermöglicht, auch auf denjenigen Gebieten, die für das Verständnis des Gehirns wesentlich sind, zum Beispiel in der Neurophysiologie und in der Kognitionspsychologie. In diesem Zusammenhang hat sich ein interdisziplinäres Arbeitsgebiet etabliert, die *Neuroinformatik*.

Neben Neurobiologen, Psychologen, Medizinern, Informatikern waren Physiker maßgeblich an der Bildung von wichtigen Modellvorstellungen der Neuroinformatik beteiligt. Als einer der prominenten Beiträge sei das Hopfield-Modell [24, 25] des assoziativen Speichers genannt, das im Jahr 1982 mit der Einführung einer Energiefunktion das mathematische Arsenal der statistischen Mechanik für eine bestimmte Klasse neuronaler Netze erschloß. Ferner sei das Malsburg-Willshaw-Modell genannt [61], das die Formation einer geordneten Verschaltung zweier Neuronenschichten beschreibt. Ein dritter wichtiger Ansatz ist der *Backpropagation-Algorithmus*, der die Adaptation eines mehrschichtigen Systems aus formalen Neuronen anhand einer Sequenz von Lernbeispielen mit einem Gradientenabstiegsverfahren realisiert (siehe z. B. Rumelhart, et al., 1986 [52]). Einen guten Überblick über das Gebiet der neuronalen Netze gewährt zum Beispiel [48].

Anwendungen

Dank der Verfügbarkeit immer leistungsfähigerer Rechner besitzen die Erkenntnisse über die Funktionsweise des Gehirnes über den reinen Erkenntniswert hinaus in steigendem Maße den Wert der technischen Anwendbarkeit, z. B. als Ausgangspunkt für den Entwurf von massiv parallelen Rechnern, oder für die Entwicklung biologisch motivierter Algorithmen. Biologisch motiviert bedeutet nicht, daß die Algorithmen sich sklavisch am biologischen Vorbild orientieren, sondern biologisch soll heißen, daß die Algorithmen im Prinzip die Rechenprozesse im Gehirn, soweit sie uns bekannt sind, nachahmen, insbesondere sich durch Lernfähigkeit und Fehlertoleranz auszeichnen. Typischerweise verarbeiten derartige Algorithmen die Informationen durch das kollektive Zusammenwirken von vielen Rechenknoten, sei es auf einem Parallelrechner oder auch auf einem schnellen, konventionellen Rechner (in *von-Neumann-Architektur*, mittels sequenziellen Programme).

In den folgenden Kapiteln werden zwei solcher Anwendungen beschrieben, die im Rahmen dieser Arbeit entstanden sind.

Kohonennetze und „Neuronengase“

Die Untersuchungen in der vorliegenden Arbeit beruhen teilweise auf dem Modell der *selbstorganisierenden Merkmalskarten*, das von dem finnischen Physiker T. Kohonen [26, 27] vorgeschlagen wurde und eine Verallgemeinerung des Malsburg-Willshaw-Modell darstellt. Es beschreibt einen Mechanismus für die Bildung einer geordneten, sogenannten „topologischen“ Abbildung eines abstrakten Eingangsreizes auf ein periodisches Gitter von formalen *Neuronen*. Hierbei werden „Ähnlichkeit“ der Reizmerkmale in eine Nachbarschaftsrelation im Gitter umgewandelt.

Eine wesentliche Erweiterung erfuhr das Modell durch Ritter [50], indem die Nachbarschaftsrelationen von Neuronen in geschickter Weise für das schnelle und zuverlässige Erlernen von Ausgabesignalen genutzt werden kann. Nun ist die Zwischenabbildung in eine feste Gitterstruktur für bestimmte Eingangsreizverteilungen eine starke, manchmal unerwüschte Einschränkung, für die sich vor kurzem Abhilfe fand. Durch Wahl einer anderen, die „Nachbarschaft“ definierenden Funktion, entstand das sogenannte „Neuronengas-Modell“, welches sich den Vorteil des kooperativen Lernens von Neuronen zu Nutze macht, dabei jedoch die Zwischenabbildung auf eine vordefinierte Gitterstruktur umgeht [37]. In Kapitel **2** werden diese Modelle ausführlicher charakterisiert und mathematisch formuliert.

Zeitserienvorhersage

Bei der ersten Anwendung geht es um eines der zentralen Probleme der Wissenschaft: Gegeben Information über die Vergangenheit eines Systems — wie kann man die Zukunft vorhersagen? Der klassische Ansatz ist, ein erklärendes Modell zu suchen. Der Ausgangspunkt wird dabei das Wissen um die Gesetzmäßigkeiten sein, die das System beherrschen. Schließlich werden die Anfangsbedingungen gemessen und die zukünftige Zeitentwicklung aus dem Modell ermittelt.

Unglücklicherweise ist Information über die in einem System involvierten Prozesse manchmal *nicht* oder nur *unvollständig* vorhanden. In solchen Fällen muß man die Vorhersage auf empirische Regelmäßigkeiten stützen, die man (hoffentlich) durch Beobachtung des Systems erfährt. Bekannte Beispiele sind die Vorhersage des Wetters oder von Börsenkursen, oder auch die Vorausbestimmung einer Teilchentrajektorie in einem Kraftfeld, z. B. der Flugbahn eines Tennisballs.

Ein sehr anspruchsvolles Anwendungsbeispiel ist die Vorhersage von sogenannten „chaotischen" Zeitserien. Eine der großen Hoffnungen, die in die moderne Chaosforschung gesetzt werden, besteht darin, daß manches System, dessen Verhalten bisweilen völlig „zufällig (*random*)" erscheint, vielleicht doch vorhersagbar ist — zumindest über kurze Zeitspannen mit guter Genauigkeit. Man hofft, daß nicht die komplizierte Dynamik eines Systems mit unpraktikabel vielen, irreduziblen Freiheitsgraden vorliegt, sondern die chaotische Dynamik eines determinierten Systems mit nur wenigen Freiheitsgraden. Bis vor kurzem gab es keine Möglichkeit, einen solchen Fall für Vorhersagen in der Praxis auszunutzen.

In Kapitel **3** wird eine neue Methode beschrieben, die die Vorhersage von Zeitserien dieser Art ermöglicht. Sie wurde im Rahmen der vorliegenden Arbeit entwickelt und für das Vorhersageproblem einer Teilchentrajektorie in einem dreidimensionalen Potential sowie für eine Mackey-Glass Zeitserie in numerischen Simulationen erfolgreich angewendet. Letztere Zeitserien sind Lösungen einer chaotischen Differentialgleichung, die von Mackey und Glass für die Beschreibung von Blutzellkonzentrationen eingeführt wurden. Die Verwendung sowohl des erweiterten Kohonenmodells als auch des „Neuronengases" läßt einen interessanten Vergleich dieser beiden Algorithmen zu.

Visuo-motorische Roboterkoordination

Der zweite Hauptteil dieser Arbeit beschäftigte sich mit einem Thema aus der Robotik. Eine der eindrucksvollen Eigenschaften biologischer „Motor"-steuerungen ist ihre Fähigkeit, sich selbst zu kalibrieren und ihre Leistungsfähigkeit durch Erfahrung zu steigern. Konventionelle, ingenieursmäßig entwickelte Maschinensteuerungen werden typischerweise mittels externer Meßgeräte und Referenzen kalibriert. Durch verschiedene Einflüsse wie Vibration, Verschleiß und Alterung kann diese Eichung „driften" oder auch sprunghaft verändert werden. Insbesondere bei bewegten Strukturen, wie einem Roboter, können Kollisionen nicht immer ausgeschlossen werden. Das Resultat eines solchen Ereignisses ist, daß bis zum nächsten Kalibrierungstermin die Maschine bzw. der Roboter immer wieder denselben Fehler wiederholt. Hat sich die Struktur dabei in *unvorhergesehener* Weise geändert, so ist eine sehr teure Umarbeitung oder

Umprogrammierung der Steuerung nötig, um Restfehler nach einer Neukalibrierung zu beseitigen. Lernfähige Steuerungsalgorithmen können dieses Problem meistern. Allerdings brauchen sie *authentische* Rückkopplung über ihr tatsächliches Steuerverhalten, d. h. eine Rückkopplung, die nicht anfällig für falsche Kalibrierung ist. Die Natur gibt uns hier ein erfolgsversprechendes Vorbild: sie macht sich die *visuelle Rückkopplung* zu Nutze.

In Kapitel **4-7** wird gezeigt wie ein Roboter mit Hilfe eines Netzwerkalgorithmus lernt, sich durch zwei Kameras zu betrachten und mit dieser visuellen Rückmeldung sein Arm im Gesichtsfeld der Kameras zu positionieren. Der verwendete Roboterarm PUMA 562, ein in der industriellen Fertigung häufig eingesetzter Roboterarm, lernt hier – einem Neugeborenem gleich – durch Üben, d. h. durch Versuch und Irrtum, die Geometrie des Armes und die Geometrie der von den Kameras gesehenen Umwelt bei seinen Bewegungen richtig zu berücksichtigen.

Die visuo-motorische Koordination kann als das Erlernen einer abbildenden Funktion vom sensorischen Eingang (Videobildinformation) in den Raum der Gelenkwinkelstellungen betrachtet werden, welche hier durch ein geschicktes Rückkoppelverfahren erzielt wird. Kapitel **4** widmet sich der Erläuterung der Lernverfahrens.

Teil der Aufgabenstellung dieser Diplomarbeit war Planung, Beschaffung, Aufbau und Inbetriebnahme eines Roboterlabors. Damit ließ sich die Anwendbarkeit und Funktionstüchtigkeit der entwickelten Algorithmen in der Praxis erstmalig testen und somit verbundene Probleme und Eigenschaften für weiterführende Implementationen studieren. Kapitel **5** gibt eine detailiertere Beschreibung des entwickelten Systemkonzeptes und der einzelnen Komponenten. Diese gehören zu vier Hauptkomplexen: (*i*) der Roboter, (*ii*) der Zentralrechner, (*iii*) die Roboter Schnittstelle und (*iv*) das Bildverarbeitungssystem. Die Details der Implementation des visuo-motorischen Kontrollalgorithmus finden in Kapitel **6** ihre Beschreibung. Die Ergebnisse der Experimente werden in Kapitel **7** dargestellt und diskutiert.

Kapitel 2

Kohonennetz und „Neuronengas“

2.1 Biologische Motivation selbstorganisierender Karten

Der Sitz der intelligenten Leistungen des menschlichen Gehirns ist die Großhirnrinde, auch der *Neokortex* genannt. Anatomisch ist dies eine etwa 2-3 mm dünne, stark gefaltete Schicht aus Nervenzellgewebe von ungefähr einem fünftel Quadratmeter Abmessung. Diese Schicht bedeckt die entwicklungsgeschichtlich älteren Hirnteile, den Hirnstamm, das Klein- und das Zwischenhirn. Das menschliche Hirn besteht insgesamt aus schätzungsweise 10^{10} Neuronen, die jeweils mit $10^3 - 10^4$ anderen Neuronen verbunden sind. Ein Neuron strukuriert sich in seinen *Dentritenbaum*, den Zellkörper (*Soma*) und ein langes *Axon*, welche sich meist den Aufgaben *Informationseingabe*, *-verarbeitung* und *Ausgabe* zuordnen lassen. Die Information wird in Form von transienten Potentialänderungen (*Spikes*) entlang der Zellausläufer transportiert. Die Verbindungsstellen zu anderen Neuronen bilden *Synapsen*; Kontakte, die die Information mittels *Neurotransmitter* (spezifische chemische Botenstoffe) in eine Richtung übertragen und je nach Art der Synapse zu einer Erregung des Zielneurons beitragen (*exzitatorisch*) oder dieser abträglich sind (*inhibitorisch*; siehe z. B. [41]).

Im menschlichen Kortex sind heute etwa 80 Rindenfelder bekannt, die jeweils ein hochparalleles Modul für verschiedene Aufgaben darstellen. Es finden sich Areale, die für das Sehen, die Hör- oder Tastwahrnehmung (*visueller, auditiver und somatosensorischer Kortex*) zuständig und direkt mit Sinnesrezeptoren verbunden sind. Andere Regionen (*Assoziativfelder*) verknüpfen die Informationen, die aus anderen Regionen kommen. Wieder andere Felder sind für die Steuerung der Muskulatur verantwortlich

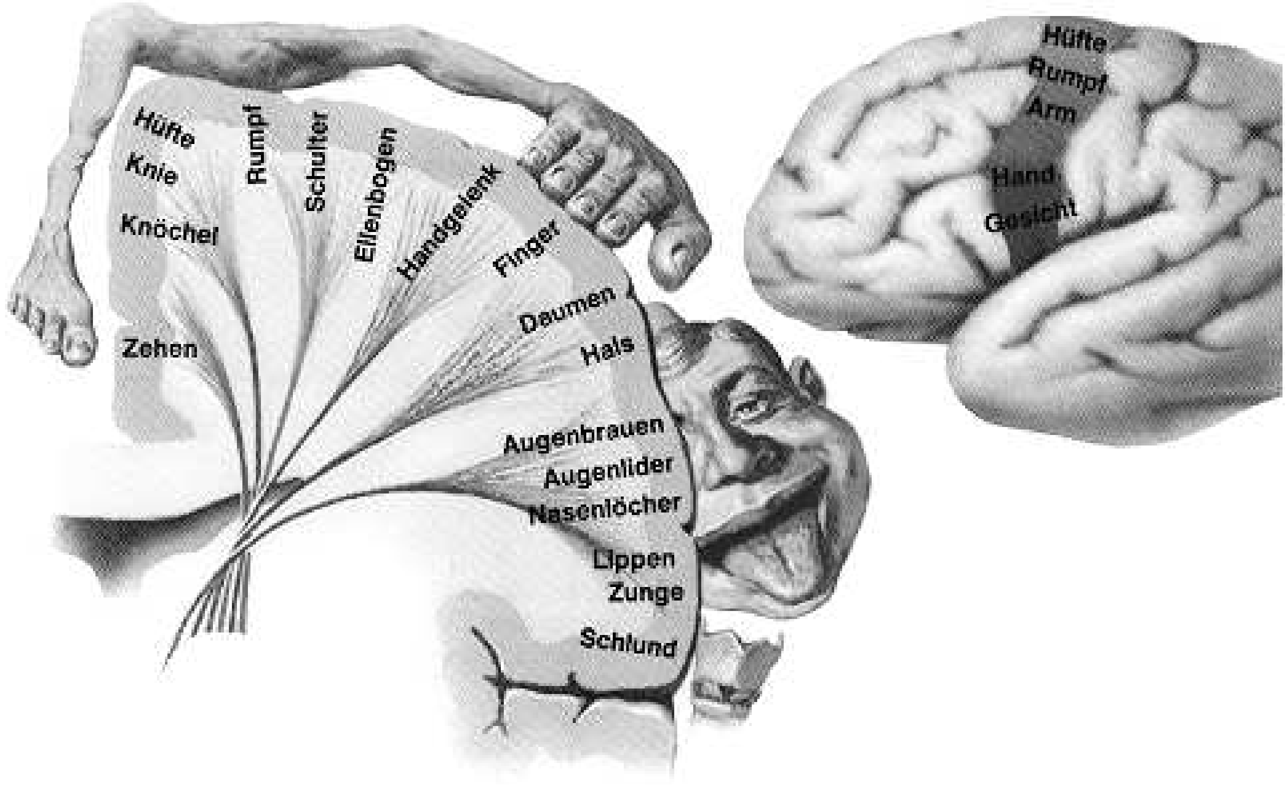


Abbildung 2.1: Somatotopische Gliederung motorischer Zentren im *Gyrus praecentralis* (*rechts*) in lateraler Ansicht der Großhirnrinde. *Links:* Schnitt durch das 4. primäre motorische Rindenfeld mit einem *Homunculus*. Der Homunculus illustriert die topologische Korrespondenz von Nervenzellen zu verschiedenen Körperteilen in denen sie Muskeln in Bewegung versetzen können (aus [41]).

(*Motorkortex*).

Trotz der Unterschiedlichkeit der Aufgabenstellungen dieser Areale treten doch überraschende Gemeinsamkeiten auf. Sie sind meist aus einem halben Dutzend Schichten aufgebaut und unterliegen weitgehend einem gemeinsamen *topologischen* Organisationsprinzip: benachbarte Neuronen eines Ausgangsfeldes sind nahezu immer mit wiederum benachbarten Neuronen im Zielareal verbunden. Dieses Prinzip findet sich auch in der Verbindungsstruktur der sensorischen Felder zu den Rezeptoren, zum Beispiel der „Sinnesoberflächen“ der Haut (Wärme- und Drucksensoren) oder der *Cochlea* (mittels Härchenzellen in der Gehörschnecke). Bild 2.1 zeigt ein Beispiel einer solchen topologischen Abbildung — auch *„topologische Karte“* genannt. Es zeigt im oberen Bildteil einen Homunkulus, der die strenge *somatotopische* Gliederung des primären motorischen Kortex andeutet, von dem aus Muskeln in verschiedenen Körperteilen in Bewegung versetzt werden können.

Der Vorteil einer solchen Strukturierung ist die Optimierung von Kommunikationsdistanzen zwischen Neuronen, die für ähnliche Signale zuständig sind. Dies stellt für ein massiv paralleles System, im Gegensatz zum konventionellen Computer mit einem, oder höchstens wenigen Rechenwerken, einen sehr bedeutenden Aspekt dar. Ferner zeugt der topologische Aufbau von einer wichtigen Abstraktionsleistung: unwesentliche Einzelheiten werden unterdrückt und die wichtigsten Signaleigenschaften oder Reizmerkmale werden entlang der Kartendimensionen abgebildet.

Ein Großteil der funktionellen Aufteilung des Kortex in Rindenfelder ist offensichtlich genetisch festgelegt, jedoch ist es unmöglich, die immense Anzahl der synaptischen Nervenverbindungen detailliert zu spezifizieren. Vielmehr muß ein zuverlässiger, selbstorganisierter Strukturbildungsmechanismus das gewünschte Resultat erzeugen.

Eine der Theorien für die Bildung von derartigen geordneten Verbindungen zwischen zwei Schichten wurde von v.d. Malsburg vorgeschlagen [55]. Sie beschreibt die *aktivitätsgetriebene* Bildung von synaptischen Verbindungen von Nervenfasern in einer Schicht (sensorische Oberfläche, z. B. Retina) mit Nervenfasern in einer Zielschicht (*Kortex*). Der Prozeß der topographischen Selbstorganisation der Karten beruht nach v. d. Malsburg auf folgenden Grundprinzipien: (*i*) Wettbewerb der synaptischen Verbindungen zwischen den beiden oben genannten Schichten, (*ii*) Wettbewerb um die Zuständigkeit der Neuronen bzgl. eines gegebenen Eingangssignals und (*iii*) Kooperation zwischen Neuronen, die im Kortex benachbart sind. Anhand von Computersimulationen konnte v. d. Malsburg qualitative Übereinstimmung zwischen Vorhersagen und experimentellen Befunden demonstrieren.

Es folgte eine Reihe von Weiterentwicklungen in verschiedene Richtungen; eine davon war ein Vorschlag von Kohonen. Im Unterschied zu den Arbeiten von v.d. Malsburg und Willshaw strebt Kohonens Modell weniger biologische Detailtreue an, sondern eher einen abtrakteren und allgemeineren Ansatz, der die beobachteten Organisationsphänomene wiederzugeben vermag (siehe z. B. [43, 42]). Dabei beschränkt sich die Beschreibung der kortikalen Vorgänge auf Zeitskalen, die nur die synaptischen Lernprozesse darstellen. Die kurzzeitigen Aktivitätsänderungen der Neuronen werden ignoriert. Im nächsten Abschnitt soll eine mathematische Beschreibung des Kohonenmodells vorgestellt werden.

2.2 Kohonens Modell selbstorganisierender Karten

Das Modell von Kohonen beschreibt die Entstehung einer topographischen Projektion eines Raumes von Eingangsreizen auf eine in der Regel zweidimensionale Schicht von formalen Neuronen[1]. Das Modell geht davon aus, daß sich als Folge einer zufälligen Sequenz von Eingangsreizen das Netzwerk so ändert, daß sich die topologische Abbildung — auch *topologische Merkmalskarte* genannt — in einer Anfangsphase ausbildet und zunehmend verfeinert. Dieser Strukturbildungsmechanismus bedarf keines Lehrers und keiner Rückkopplung eines Erfolgssignals (*reenforcement learning*) und gehört zur Klasse der sogenannten *unsupervised learning* Algorithmen.

Jedes dieser Neuronen — sie seien im folgenden mit dem Index $\mathbf{r}$ gekennzeichnet — ist über Synapsen mit denselben m Nervenfasern, die das zu verarbeitende Signal tragen, verknüpft. Anders formuliert, jeder dieser neuronalen Prozessorelemente $\mathbf{r}$ bekommt die gleichen m Werte x_j $(j = 1, 2, \ldots m)$ zugeführt, die einen Punkt $\mathbf{x}_{stim} = (x_1, x_2, \ldots x_m)^T$ (Stimulus) im m-dimensionalen Eingangsraum definieren. Pro Eingangsleitung $j \in \{1, 2, \ldots m\}$ besitzt jedes Neuron $\mathbf{r}$ ein reelles Gewicht $w_{\mathbf{r}j}$ (Synapsenstärke), die zusammen den *Gewichts-* oder *Referenzvektor* $\mathbf{w_r} = (w_1, w_2, \ldots, w_m)^T$ konstituieren. Für ein einlaufendes Signal $\mathbf{x}_{stim} = (x_1, x_2, \ldots, x_m)^T$ bestimmt jedes Prozessorelement seine *Stimulusdistanz*

$$SD_{\mathbf{r}} = \mathbf{D}(\mathbf{w_r}, \mathbf{x}). \tag{2.1}$$

Dabei beschreibt $\mathbf{D}(\mathbf{u}_1, \mathbf{u}_2)$ eine Distanzfunktion. $\mathbf{D}$ wird häufig nach der Euklidischen Metrik $\mathbf{D}(\mathbf{u}_1, \mathbf{u}_2) = \|\mathbf{u}_1 - \mathbf{u}_2\|$ (diese wird im folgenden verwendet) oder nach einer Raumwinkelmetrik $\mathbf{D}(\mathbf{u}_1, \mathbf{u}_2) = 1 - \mathbf{u}_1 \cdot \mathbf{u}_2 \left(\|\mathbf{u}_1\| \cdot \|\mathbf{u}_2\|\right)^{-1}$ gewählt.

Jedes Eingangssignal x_{stim} verursacht einen Wettbewerb der kleinsten Stimulusdistanzen. Dazu wird dasjenige Neuron $\mathbf{s}$ bestimmt, dessen Referenzvektor $\mathbf{w_s}$ am nächsten zum Stimulus x_{stim} liegt[2]:

$$SD_{\mathbf{s}} = \min_{\forall\, \mathbf{r}} SD_{\mathbf{r}} \tag{2.2}$$

Diese Auswahlregel beschreibt eine Abbildung ϑ_W eines Eingangsreizes $\mathbf{x}_{stim} \in X$ auf die Neuronenschicht

$$\vartheta_W \;:\; \mathbf{x}_{stim} \mapsto \mathbf{s} = \vartheta_W(\mathbf{x}_{stim})\ , \tag{2.3}$$

[1] Der Einfachheit halber im folgenden „Neuron“ genannt.

[2] Dazu gibt es mehrere Möglichkeiten: Zum Beispiel kann ein zentrales Prozessorelement die Stimulusdistanzen $SD_{\mathbf{r}}$ sortieren und den ermittelten „Sieger“ zurückübertragen, oder jedes Neuron vergleicht direkt sein Ergebnis mit dem aller anderen. Man kann auch eine sogenannte *Umfeldhemmung (laterale Inhibition)* verwenden: Mit Hilfe eines langreichweitig hemmenden Botenstoffes, der lokal durch einen kurzreichweitigen Aktivator kompensiert wird, kann man erreichen, daß am Ende genau ein einzelnes Erregungszentrum bestehen bleibt [48]. Eine weitere Möglichkeit: ein zentrales Prozessorelement sendet ein Schwellensignal, das von 0 ausgeht und dann langsam ansteigt. Das erste Neuron dessen Stimulusdistanz SD kleiner ist als das Schwellensignal, gewinnt das Rennen. Wichtig ist, daß es Methoden gibt, die das Neuron $\mathbf{s}$ in einer Zeitspanne finden, die unabhängig von der Anzahl der Neuronen in der Schicht ist [22].

der Index $\mathbf{w}$ soll dabei die Abhängigkeit von den Synapsenstärken $\mathbf{w}$ andeuten. ϑ_W unterteilt den Eingangsraum X in disjunkte Zellen, sogenannte *Voronoi-Polygone* oder auch *Tessellationszellen* [23]. Die Grenzen zwischen diesen Zellen sind durch die Hyperebenen definiert, die die Verbindungsgeraden zwischen benachbarten Referenzvektoren senkrecht halbieren, wie in Bild 2.2 illustriert. Soweit ist das Konzept identisch zu einem konventionellen Vektorquantisierungsverfahren. Den Unterschied bildet die folgende Lernregel:

$$\mathbf{w}_{\mathbf{r}}^{(new)} = \mathbf{w}_{\mathbf{r}}^{(old)} + \varepsilon h_{\mathbf{rs}} \left(\mathbf{x}_{stim} - \mathbf{w}_{\mathbf{r}}^{(old)} \right) . \tag{2.4}$$

Hierbei skaliert der Faktor $\varepsilon \in [0,1]$ die Größe des Adaptationsschrittes, moduliert durch die *Nachbarschaftsfunktion* $h_{\mathbf{rs}} = f(\|\mathbf{r}-\mathbf{s}\|)$. Die Funktion $h_{\mathbf{rs}}$ hat ihr Maximum für $\mathbf{r} = \mathbf{s}$ ($h_{\mathrm{ss}} = 1$) und fällt monoton für zunehmenden Gitterabstand $\|\mathbf{r}-\mathbf{s}\| \to \infty$ auf Null ab. In Bild 2.2 ist die Abbildung ϑ_W schematisch darge-

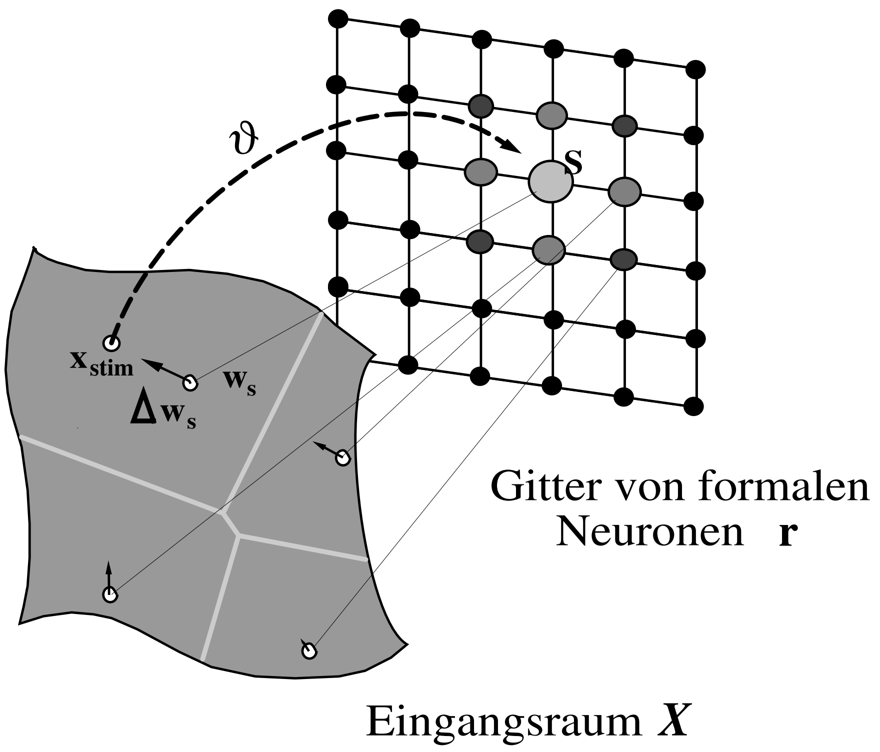


Abbildung 2.2: Der Lernschritt im Modell von Kohonen: der Stimulusvektor $\mathbf{x}_{stim} \in X$ selektiert das zuständige Neuron $\mathbf{s}$, in dessen Nachbarschaft alle Neuronen ihre Gewichtsvektoren $\mathbf{w}$ in Richtung $\mathbf{x}_{stim}$ verschieben. Die hellgrauen Linien zeigen die Aufteilung (*Tessellation*) von X in disjunkte Zellen (Voronoi-Polygone). Die Pfeillängen in X und verschiedene Graufärbung der Neuronen deuten an, daß das Ausmaß der Verschiebung mit wachsendem Abstand vom Zentrum $\mathbf{s}$ abnimmt.

stellt. Die Gewichte der Neuronen nahe am „Siegerneuron“ $\mathbf{s}$ und von $\mathbf{s}$ selbst werden merklich geändert, die entfernteren Neuronen hingegen, für die $h_{\mathbf{rs}}$ klein ist, verspüren kaum einen Effekt, wie durch die verschiedene Graufärbung und Pfeillängen in Abbildung 2.2 angedeutet wird. An dieser Stelle wird der topologische Informationsgehalt vermittelt: Die im Gitter benachbarte Neuronen erfahren ähnliche Adaptationen und werden schließlich für ähnliche Eingabemuster zuständig. Eine geeignete Wahl für $h_{\mathbf{rs}}$ ist z. B. die Gaußglocke

$$h_{\mathbf{rs}} = \exp(-\frac{\|\mathbf{r}-\mathbf{s}\|^2}{2\sigma^2}) \,. \tag{2.5}$$

Der Radius σ bestimmt dabei die Größe der Nachbarschaftregion auf dem Neuronengitter (siehe Bild 2.4). Anfänglich werden beide Parameter[3], ε und σ, relativ groß gewählt, um eine schnelle Grobanpassung des Netzwerkes zu erreichen. Mit zunehmender Zahl von Lernschritten t werden sie dann allmächlich verringert, so daß sich die Feinstruktur der Karte entwickeln kann.

Die Gleichung 2.4 „zieht“ den Referenzvektor $\mathbf{w_s}$ des Siegerneurons in Richtung des Stimulusvektors $\mathbf{x}_{stim}$, aber nicht nur den des Neurons $\mathbf{s}$, sondern auch die Referenzvektoren $\mathbf{w_r}$ von dessen Gitternachbarn (siehe auch Bild 2.2). Man kann sich das Gebilde als eine Art elastisches Netz vorstellen, das versucht, sich der Stimulusverteilung im Eingaberaum so gut wie möglich anzupassen. Die Topologie des Netzes ist gleich derjenigen der Neuronenschicht und die Knoten werden von den Referenzvektoren $\mathbf{w}$ beschrieben.

2.2.1 Interessante Eigenschaften

Wie im folgenden gezeigt wird, läßt sich die Adaptationsregel 2.4 auch als Minimalisierungsvorschrift für eine Kostenfunktion ableiten. Diese Vorschrift ist mit der Frage optimaler Datenrepräsentation und Datenkompression verknüpft, die in der Datenspeicherungstechnik und Übertragungstechnik auftritt. Die Idee ist, eine gegebene Menge oder Wahrscheinlichkeitsverteilung von Vektoren (bzw. von Daten oder Mustern) $\mathbf{x} \in X$ in N Klassen zu kategorisieren und dann alle Vektoren durch deren Klasse zu repräsentieren. Sobald Sender und Empfänger sich auf einen Satz von Klassen oder „Kodebuch“ geeinigt haben, wird nurmehr der entsprechende Klassenindex übertragen und nicht mehr der ganze Vektor. Diese Klassen werden meist durch einen Satz von *Prototypenvektoren* $W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_N\}$ definiert. Die Klassenzugehörigkeit eines gegebenen Musters $\mathbf{x}$ wird durch Finden des topologisch nächsten Prototypenvektors $\mathbf{w_s}$ gemäß der Auswahlregel 2.2 festgelegt. Eine „gute Repräsentation“ W des Eingangsraumes X läßt sich auf verschiedene Weise definieren. Oft wird die Minimalisierung des Erwartungswertes des quadratischen Rekonstruktionsfehlers,

[3]Typische Startwerte sind $\sigma(t=0) = O(0.5\text{-}0.9 \text{ Gitterabmessung})$, $\varepsilon(t=0) = O(0.5\text{-}0.9)$.

d. h. des Funktionals

$$E[W] = \frac{1}{2} \int \|\mathbf{x} - \mathbf{w}_{\vartheta(\mathbf{x})}\|^2 P(\mathbf{x})\, d^m\mathbf{x}, \tag{2.6}$$

gefordert, wobei $P(\mathbf{x})$ die Wahrscheinlichkeitsverteilung der Vektoren $\mathbf{x} \in X \subseteq \mathbf{R}^m$ beschreibt. Funktionale dieser Art werden auch Kostenfunktion genannt.

Nimmt man im obigen Fall zusätzlich eine „verrauschte“ Übertragung des Klassenindizes an, d. h. mit der Wahrscheinlichkeit $h_{\mathbf{rs}}$ wird der Index $\mathbf{s} = \vartheta_W(\mathbf{x})$ mit dem Index $\mathbf{r}$ als Folge des Übertragungsprozesses verwechselt, dann wird die Rekonstruktion des Vektors $\mathbf{x}$ nicht immer den nächsten Prototypenvektor $\mathbf{w_s}$ ergeben, sondern mit der Wahrscheinlichkeit $h_{\mathbf{rs}}$ fälschlicherweise $\mathbf{w_r}$. Folglich ist nun das Funktional

$$E'[W] = \frac{1}{2} \sum_{\mathbf{r},\mathbf{r}'} h_{\mathbf{r}\vartheta_W(\mathbf{x})} \cdot \int_{\vartheta_W(\mathbf{x})=\mathbf{r}'} \|\mathbf{x} - \mathbf{w_r}\|^2 P(\mathbf{x})\, d^m\mathbf{x}, \tag{2.7}$$

anstatt (2.6) zu minimieren [49]. Das Gradientenabstiegsverfahren liefert dazu eine einfache Methode, um ein lokales Minimum für $E'[W]$ zu finden, d. h. die Prototypenvektoren $\mathbf{w_r}$ werden entsprechend

$$\dot{\mathbf{w}}_\mathbf{r} = \sum_{\mathbf{r}'} h_{\mathbf{rr}'} \cdot \int_{\vartheta_W(\mathbf{x})=\mathbf{r}'} (\mathbf{x} - \mathbf{w_r})\, P(\mathbf{x})\, d^m\mathbf{x} \tag{2.8}$$

geändert. Für den Fall einer zufälligen Folge von Mustervektoren $\mathbf{x}$ ist dies äquivalent zur diskreten Kohonenlernregel Gleichung (2.4).

In einigen Spezialfällen kann man weitergehende Aussagen über das Konvergenzverhalten des Kohonen-Algorithmus machen. Um den Gleichgewichtsfall zu studieren, wird (2.7) zu Null gesetzt, was zu einem nicht-trivialen Gleichungssystem führt. Insbesondere im eindimensionalen Fall ($m = 1$) kann man dann die Konvergenz zu einer topographischen Karte beweisen [13], ferner kann man durch Übergang vom Neuronengitter zu einer Kontinuumsnäherung die Dichteverteilung der Referenzvektoren ableiten. Hierzu wird die Summe $\sum_{\mathbf{r}'}$ durch das Integral $\int \rho(\mathbf{r}) d\mathbf{r}$ über die Dichteverteilung $\rho(\mathbf{r})$ der Referenzvektoren im Eingaberaum X ersetzt. Für eine eindimensionale Kette von Neuronen mit einer hutförmigen Nachbarschaftsfunktion, die c Nachbarn berücksichtigt ($h_{\mathbf{rs}} = \{1 \text{ für } \|\mathbf{r}-\mathbf{s}\| \leq c, 0 \text{ sonst}\}$), wird in [51] gezeigt, daß sich die asymptotische Dichte der Referenzvektoren $\rho(\mathbf{x})$ dem Potenzgesetz

$$\rho(\mathbf{x}) \propto P(\mathbf{x})^\alpha, \qquad \text{mit} \quad \alpha = \frac{2}{3} - \frac{1}{3c^2 + 3(c+1)^2} \tag{2.9}$$

in Abhängigkeit der Eingangsvektorendichte $P(\mathbf{x})$ unterwirft. Da immer $\alpha < 1$ gilt, hat der Kohonen-Algorithmus die Tendenz Regionen des Eingangsraumes X mit großer Auftretenshäufigkeit unterzurepräsentieren und umgekehrt Regionen mit kleiner Häufigkeit stärker zu repräsentieren.

2.3 Die Erweiterung des Kohonen Modells

In den vorangegangenen Abschnitten wurde die Fähigkeit des Kohonen-Algorithmus beschrieben, auf der Basis einer Zufallsfolge von sensorischen Reizen eine topologische Karte auf einem Gitter von formalen Neuronen zu etablieren. Nun möchte man sich nicht damit begnügen, die Hauptmerkmale einer Signalverteilung optimiert zu repräsentieren, sondern man möchte auch eine spezifische Reaktion oder einen spezifischen Ausgabewert mit einem Eingangsreiz verknüpfen können.

Zu diesem Zweck wird an jedem Neuron der Schicht ein zusätzlicher *Ausgabewert* gespeichert (Ritter und Schulten, 1986 [50]). Bei diesem Ausgabewert $\mathbf{a}$ kann es sich etwa um eine skalare Größe, einen Vektor oder eine um eine lineare Abbildung, d. h. einen Tensor, handeln.

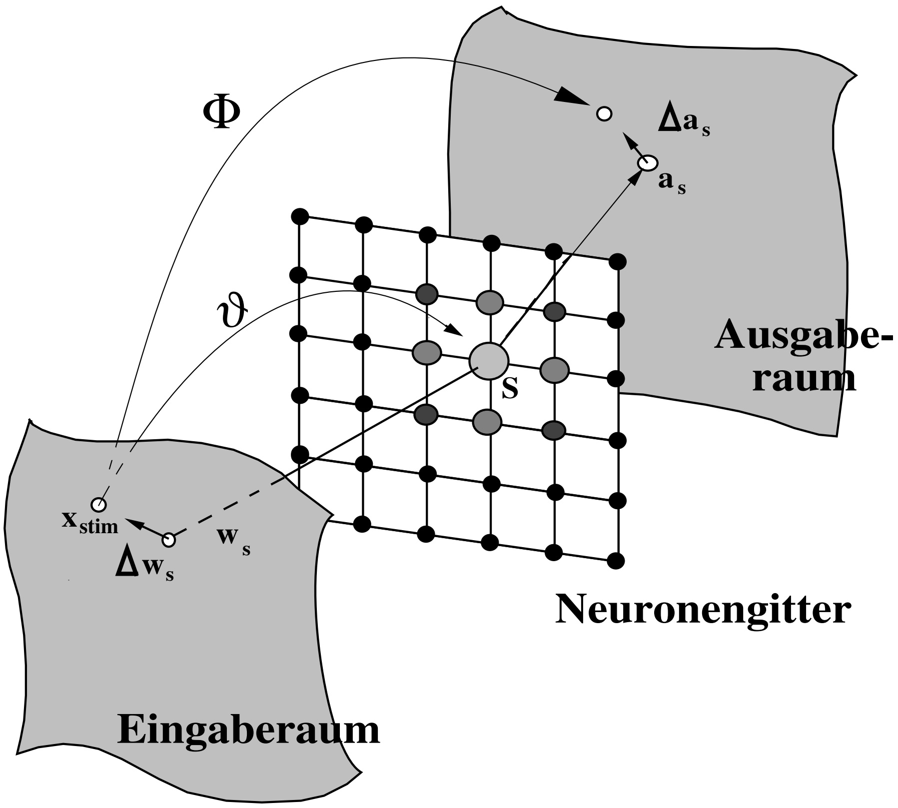


Abbildung 2.3: Jedes Neuron im planaren Gitter speichert zwei Daten: (i) den Referenzvektor $\mathbf{w}$, der die Zuständigkeitsverhältnisse regelt, und (ii) den Ausgabewert (hier den Vektor) $\mathbf{a}$. Der Stimulusvektor $\mathbf{x}_{stim}$ wird auf das Neuron ϑ_w im Gitter abgebildet (Φ). Das Neuron $\mathbf{s}$ und (in abgeschwächter Form) dessen Nachbarn (grau markiert) nehmen an der Adaptation teil: (i) Referenzvektoren $\mathbf{w}$ in Richtung $\mathbf{x}_{stim}$, (ii) Ausgabevektoren $\mathbf{a}$ gemäß eines geeigneten Fehlerkorrekturverfahrens.

Damit wird das Erlernen einer Abbildung

$$\Phi_{\vartheta,\mathbf{a}} : X \mapsto Y \subseteq \mathbf{R}^n, \qquad \mathbf{x} \to \mathbf{y} = \mathbf{a}_{soll} = \Phi_{\vartheta,\mathbf{a}}(\mathbf{x}) \tag{2.10}$$

des Eingangsraumes X in einen Ausgaberaum, den Bildraum Y, möglich. Abbildung 2.3 illustriert dies wieder anhand von flächenhaft dargestellten Ein- und Ausgaberäumen.

Das Lernen der Abbildung (2.10) kann durch den folgenden Lernalgorithmus für eine vektorelle Ausgabegröße bewirkt werden:

0. *„Initialisierung“:* Start mit geeigneten Anfangswerten für die Referenzvektoren $\mathbf{w_r}$ und Ausgabewerte $\mathbf{a_r}$. In Abwesenheit irgendwelchen a-priori Wissens können die Werte auch zufällig gewählt werden.

1. *„Stimuluswahl“:* Wähle zufällig einen Stimulus $\mathbf{x}$ gemäß der Wahrscheinlichkeitsdichte $P(\mathbf{x})$. Der zugehöhrige korrekte Ausgabewert sei $\mathbf{a}_{soll}(\mathbf{x})$.

2. *„Response“:* Bestimme den Gitterplatz $\mathbf{s} = \vartheta_W(\mathbf{x})$ („Erregungszentrum“) aus der Auswahlbedingung (2.2)

$$\|\mathbf{w_s} - \mathbf{x}\| = \min_{\forall\, \mathbf{r}} \|\mathbf{w_r} - \mathbf{x}\|, \tag{2.11}$$

   und gebe dessen gespeicherten Wert $\mathbf{a_s}$ aus (*winner takes all network*).

3. *„Adaptationsschritt (i)“:* Führe einen Lernschritt gemäß (2.4)

$$\mathbf{w_r}^{(new)} = \mathbf{w_r}^{(old)} + \varepsilon h_{\mathbf{rs}} \left(\mathbf{x} - \mathbf{w_r}^{(old)}\right).$$

   für die Referenzvektoren $\mathbf{w_r}$ aus.

4. *„Adaptationsschritt (ii)“:* Führe einen Lernschritt

$$\mathbf{a_r}^{(new)} = \mathbf{a_r}^{(old)} + \varepsilon' h'_{\mathbf{rs}} \Delta \mathbf{a_r}, \tag{2.12}$$

$$\text{mit} \quad \Delta \mathbf{a_r} = \mathbf{a}_{soll} - \mathbf{a_r}^{(old)} \tag{2.13}$$

   für die Ausgabevektoren $\mathbf{a_r}$ aus und gehe nach 1.

Schritte 1-3 bilden Kohonens ursprünglichen Algorithmus zur Bildung der topologieerhaltenden Karte auf dem Neuronengitter. Der neu hinzugekommene Lernschritt 4 adaptiert die Ausgabewerte, um die gewünschte Abbildung von X nach Y zu etablieren. Dies erfolgt völlig analog zum Lernschritt für die Referenzvektoren, gegebenenfalls mit eigener Lernschrittweite ε' und der Nachbarschaftsfunktion $h'_{\mathbf{rs}}$ (vgl. ε und $h_{\mathbf{rs}}$).

Der Lernschritt 4 kann in einer anderen Form dargestellt werden, wenn die zu lernende Ausgabefunktion linear vom Eingangssignals $\mathbf{x}$ abhängt, nämlich

$$\mathbf{y} = \mathbf{a_s} \cdot \mathbf{x}\,, \qquad \mathbf{a_r} \in \mathbf{R}^n \times \mathbf{R}^m \tag{2.14}$$

wobei $\mathbf{a}_r$ zu einer Matrix erweitert wird. In diesem Falle ändert sich Gleichung(2.13) zu einer Fehlerkorrekturregel vom Widrow–Hoff Typ [60]

$$\Delta \mathbf{a_r} = (\mathbf{y} - \mathbf{a_r} \cdot \mathbf{x}) \cdot \frac{\mathbf{x}}{\|\mathbf{x}\|^2}. \tag{2.15}$$

Die Fehlerkorrektur ist so gewählt, daß sie den mittleren quadratischen Ausgabefehler bzw. Kostenfunktion der Form[4]

$$E_{cost}[\mathbf{a_r}] = \frac{1}{2} \left\langle \xi^2(\mathbf{x}_{t'}, \mathbf{y}_{t'}, \mathbf{a}_r) \right\rangle_{t'} = \frac{1}{2} \left\langle \|\mathbf{y} - \mathbf{x} \cdot \mathbf{a}_s\|^2 \right\rangle_{t'} \quad , \tag{2.16}$$

mittels Gradientenabstiegverfahren minimiert. Diese stochastische Minimalisierungsmethode läßt sich aus der Verallgemeinerung des eindimensionalen Newton'schen Nullstellenverfahrens leicht ableiten. Differenzierbarkkeit angenommen, ist E_{cost} an den Stellen extremal, für die gilt:

$$\frac{dE_{cost}}{d\mathbf{a}} = \xi \frac{d\xi}{d\mathbf{a}} \stackrel{!}{=} 0 \; . \tag{2.17}$$

Betrachten wir vereinfachend eine skalare Funktion $\xi(a)$, deren Nullstelle gesucht ist (siehe Abbildung 2.4b). Man startet mit einem Schätzwert a_ν, der nun iterativ um Δa verschoben wird, bis man die Nullstelle gefunden hat. Wie man aus der

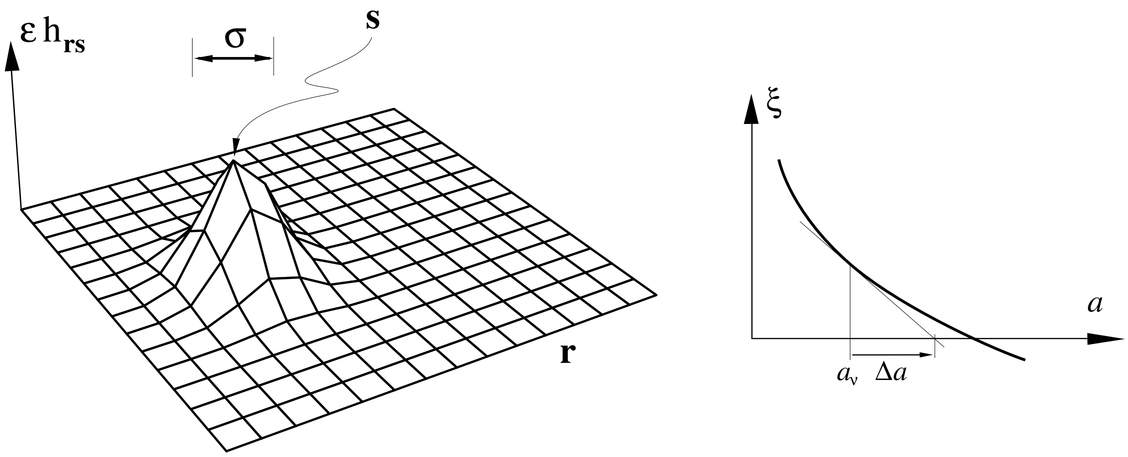


Abbildung 2.4: *links:* Gaußförmige Modulationsfunktion der Lernschrittweite $\varepsilon h_{\mathbf{rs}}$ in Abhängigkeit von der Gitterposition $\mathbf{r}$ und $\mathbf{s}$.
rechts: Methode des steilsten Gradientenabstieges im Eindimensionalen.

Graphik leicht ersieht, ergibt sich Δa zu

$$\Delta a \frac{d\xi}{da}|_{a_\nu} = -\xi(a_\nu) \; . \tag{2.18}$$

[4]Zeitmittel $\langle x \rangle_{t'} := \frac{1}{l} \sum_{i=1}^{l} x(t_i)$.

Verallgemeinert man dieses Verfahren für nichtskalare $\mathbf{a}$ zum Gradientenabstiegsverfahren, so kann man Gleichung (2.18) um den Gradienten $\frac{d\xi}{d\mathbf{a}}|_{\mathbf{a}_\nu}$ erweitern, um schließlich nach $\Delta\mathbf{a}$ aufzulösen:

$$\Delta\mathbf{a} = -\xi(\mathbf{a}_\nu)\, \frac{\left.\frac{d\xi}{d\mathbf{a}}\right|_{\mathbf{a}_\nu}}{\left\| \left.\frac{d\xi}{d\mathbf{a}}\right|_{\mathbf{a}_\nu} \right\|^2} \,. \tag{2.19}$$

Mit $\xi(\mathbf{a}_\nu) = \xi\left(\mathbf{x}(t), \mathbf{y}(t), \mathbf{a}\right)$ ergibt sich Gleichung (2.15). Diese Art der Adaptationsregel wird uns später wieder begegnen.

Das erweiterte Kohonenverfahren ist ein *Hybridalgorithmus* im Sinne einer Vereinigung eines *unsupervised* $[s = \vartheta_W(\mathbf{x})]$ und eines *supervised learning*-Schemas $[\Phi(\mathbf{s}),$ *siehe* (2.10)]. Man kann das im Verlaufe der Lernphase entstandene Gebilde auch als eine Art Filterbank für die Eingangssignale betrachten, oder als Wertetabelle für die Funktion Φ. Der besondere Vorteil des geschilderten Lernverfahrens liegt in der Adaptivität der Tabellenstruktur. Weder die Struktur noch die Zuordnung der Tabellenwerte ist von Anfang an starr vorgegeben, sie entwickelt sich statt dessen im Laufe der Lernphase. Regionen in X, die häufig benötigt werden, erhalten automatisch eine höhere Dichte an Tabelleneinträgen und ermöglichen damit eine bessere Auflösung der Eingabe-Ausgaberelation. Selten oder nie benutze Tabelleneinträge werden „umgewidmet“, was auch in der späteren „Arbeitsphase“noch möglich ist, falls ε auf einem Wert größer als Null gehalten wird (für $\varepsilon = \varepsilon' = 0$ „friert“ das Netzwerk ein).

Dank des topologieerhaltenden Charakters der Kohonen Karte werden im Gitter benachtbart liegenden Neuronen ähnliche Wertepaare $(\mathbf{x}, \mathbf{a}(\mathbf{x}))$ zugewiesen. Im Falle eines stetigen Zusammenhangs zwischen Eingabe- und Ausgabewerten müssen benachbarte Speicherplätze auch ähnliche Ausgabewerte $\mathbf{a}_{soll}$ lernen. Somit wird die Teilname einer ganzen Neuronengruppe an einem Lernschritt aus zwei Gründen zu einem wichtigen Vorteil:

- *Beschleunigung des Lernvorganges.* Gibt man den Funktionen $h_{\mathbf{rs}}$ und $h'_{\mathbf{rs}}$ anfänglich eine große Reichweite (σ, σ', siehe Bild 2.4), so nehmen an einem Adaptationschritt viele Neuronen teil, auch wenn der Lerninhalt für die meisten nur ungefähr „richtig“ ist. Schon nach bedeutend weniger Trainingsschritten als Speicherplätze vorhanden sind wird eine vollständige Tabelle erzeugt, die oft schon eine brauchbare Näherung der korrekten Ein-Ausgaberelation ist. Für weitere Verbesserungen werden die Nachbarschaftsreichweiten σ und σ' graduell verringern, so daß die Neuronen zunehmend “individualisieren“ und effektiv nur noch wenige einzelne Neuronen an diesem Schritt teilnehmen.

- *Robustheit.* Das Gradientenabstiegsverfahren kann im Falle ungünstiger Anfangsbedingungen in lokalen Nebenminima steckenbleiben. Wie in Abbildung

2.5 anhand eines „Honigtopfanalogons“ qualitativ illustriert wird, kann das Konvergenzverhalten des gesamten Lernalgorithmus durch Nachbarschaftskooperation verbessert werden (siehe auch Seite 52 und [48]).

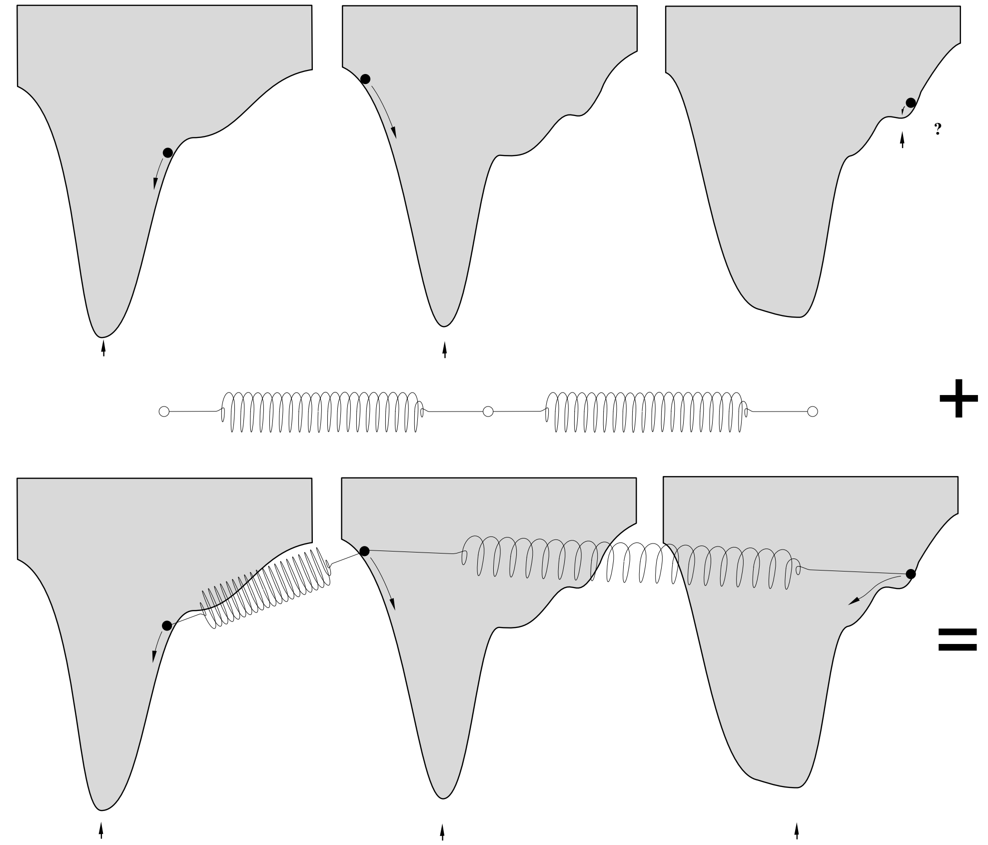


Abbildung 2.5: Verbesserung des Konvergenzverhaltens durch Kooperation dreier benachbarter Neuronen in Analogie zu Kugeln in Honig. *Oben:* Die Kostenfunktionen $E_{cost}[a_{\mathbf{r}_1}]$, $E_{cost}[a_{\mathbf{r}_2}]$ und $E_{cost}[a_{\mathbf{r}_3}]$ in Abhängigkeit eines zu lernenden Parameters $a_{\mathbf{r}_i}$ (i=1,2,3) seien als Funktion der potentiellen Energie für die Kugeln im Honigtopf nachgebildet. Nach zufälligem Einwurf (Initialisierung) laufen die Kugeln nach unten, *oben rechts:* die Kugel bleibt in einer Mulde stecken und erreicht nicht mehr das globale Minimum. *Mitte:* Verbindet man nun die Kugeln mit Federn (Ruhelänge gleich dem Topfabstand) können sich die Kugeln gegenseitig helfen, ins jeweilige globale Minimum zu gelangen (*unten*). Dies ist genau dann profitabel, wenn die Potentiallandschaften auch ähnlich, d. h. die Neuronen „benachbart“ genug sind. Günstig ist es, die Kopplungsstärke (Federkonstante) allmählich zu verringern, damit sich die Kugeln am Ende nicht vom jeweiligen tiefsten Punkt ablenken können (Pfeile, siehe insb. Topfboden rechts unten).

2.4 Das „Neuronengas“

Um optimale Ergebnisse mit dem erweiterten Kohonen-Algorithmus zu erzielen, muß die Topologie des verwendeten Gitters derjenigen der Mannigfaltigkeit X der Eingangssignale entsprechen, andernfalls wird anfänglich das Gitter stark verzerrt und verhindert die gewünschte Adaptation an die Mannigfaltigkeit X solange die Nachbarschaftsfunktionen $h_{\mathbf{rs}}$ und $h'_{\mathbf{rs}}$ Gitterregionen einschließen, die nicht für ähnliche Ein- und Ausgaberelationen zuständig sind. Damit hängt die Adaptationsleistung von der a-priori Kenntnis der topologischen Struktur der Mannigfaltigkeit X ab. In manchen Fällen ist diese schwer zu ermitteln, in anderen Fällen ist die Topologie stark heterogen, was eine optimale Ausnutzung der Speicherplätze vereitelt.

Der in [37] vorgeschlagene „Neuronengas“ Algorithmus[5] umgeht diese Probleme elegant. Dieses Modell sei im folgenden charakterisiert und anschliesend mit dem erweiterten Kohonenmodell verglichen.

2.4.1 Charakterisierung und Lernregeln

Das „Neuronengas“ Modell verwendet die gleichen N formalen Neuronen wie das oben dargestellte erweiterte Kohonenmodell, ohne sie jedoch in ein Gitter zu betten. Im folgenden werden Neuronen nicht mehr mit einem Gitterindex $\mathbf{r}$ (z. B. $\mathbf{r} \in \mathbf{R}^2$) sondern mit einem Index $\mu \in \mathbf{N}$ gekennzeichnet. Der wesentliche Unterschied ist die Art und Weise, wie die Nachbarschaftsbeziehungen zwischen den Neuronen festgestellt werden: sie werden im Falle des „Neuronengasmodell“ dynamisch auf der Basis der Nachbarschaftsverhältnisse im Eingangsraum bestimmt, im Gegensatz zum Kohonenmodell, in dem dies aufgrund der Nachbarschaftsverhältnisse im starren Neuronengitter geschieht. Insbesondere der „Rang“ [siehe 2.20)] des Abstandes des Referenzvektors $\mathbf{w}_\mu$ zum Eingangssignal $\mathbf{x}$ entscheidet, in welchem Ausmaß das Neuron μ am momentanen Signalverarbeitungsschritt teilnimmt.
Die Lernschrittfolge 2-4 auf Seite 13 ändert sich, wieder in der Reihenfolge des Algorithmus dargestellt, wie folgt:

2. Ausgabe (*„Response“:*)
   Für jedes neue Eingangssignal wird die Rangfolge $(\mu_0, \mu_1, \ldots, \mu_{N-1})$ der Neuronen nach wachsendem Abstand zum Vektor $\mathbf{x}$ ermittelt. Die Reihenfolge ist durch
   $$\|\mathbf{w}_{\mu_0} - \mathbf{x}\| < \|\mathbf{w}_{\mu_1} - \mathbf{x}\| < \cdots < \|\mathbf{w}_{\mu_{N-1}} - \mathbf{x}\|. \tag{2.20}$$
   definiert. Entsprechend einem *winner takes all network* wird der Wert $\mathbf{a}_{\mu_0}$ ausgegeben (was $\mathbf{a}_\mathbf{s}$ entspricht). Eine prinzipielle Alternative ist das Netzwerk als "*winner takes most network*" zu entwerfen, d. h. eine ganze Gruppe von Neuronen zu berücksichtigen, deren Referenzvektoren nahe bei $\mathbf{x}$ liegen. Diese kann man durch Mittelung über alle Ausgabewerte $\mathbf{a}_\mu$ erreichen, gewichtet mit

[5] Die Benennung „Neuronengas“ betont eine Analogie zu frei beweglichen Teilchen eines Gases.

einer Funktion $g^{mix}(\mu_k)$, die vom „Nachbarschaftsrang“ der Vektoren $\mathbf{w}_{\mu_k}$ zum Eingabesignal $\mathbf{x}$ abhängt.

Die Wechselwirkungsfunktion $g^{mix}(\mu_k) = g^{mix}(k)$ ist abhängig von der *Anzahl*[6] der Neuronen k, deren Referenzvektoren *näher* an $\mathbf{x}$ liegen *als* $\mathbf{w}_{\mu_k}$. Sie hat ihr Maximum bei $g^{mix}(k=0) = 1$ and fällt mit steigendem k auf null ab; eine geeignete Wahl ist

$$g^{mix}(k) = \exp\left(-\frac{k}{\lambda^{mix}}\right). \tag{2.21}$$

Damit resultiert die Mittelung für die Netzwerksausgabe $\mathbf{a}_{out}$ zu

$$\mathbf{a}_{out}(\mathbf{x}) = \mathcal{S}^{-1} \cdot \sum_k g^{mix}(k) \cdot \mathbf{a}_{\mu_k}\,, \qquad \text{mit } \mathcal{S} = \sum_k g^{mix}(k). \tag{2.22}$$

Der Beitrag des „Sieger-Neuron“ μ_0, dessen $\mathbf{w}_{\mu_0}$ am nächsten zu $\mathbf{x}$ liegt, wird am größten; das Neuron μ_1, dessen $\mathbf{w}_{\mu_1}$ am zweitnächsten zu $\mathbf{x}$ liegt, trägt am zweitmeisten bei, und so fort. Dieses Konzept der k-rangabhängigen Beteiligung des Neurons μ_k am aktuellen Versuch wird in den Adaptationsregeln übernommen.

3. *„Adaptationsschritt (i)“:* Führe einen Lernschritt gemäß (2.4)

$$\mathbf{w}_{\mu_k}^{(new)} = \mathbf{w}_{\mu_k}^{(old)} + \varepsilon g(k)\left(\mathbf{x} - \mathbf{w}_{\mu_k}^{(old)}\right). \tag{2.23}$$

für die Referenzvektoren $\mathbf{w}_{\mathbf{r}}$ aus.

4. *„Adaptationsschritt (ii)“:* Führe einen Lernschritt

$$\mathbf{a}_{\mu_k}^{(new)} = \mathbf{a}_{\mu_k}^{(old)} + \varepsilon' g'(k) \Delta \mathbf{a}_{\mu_k} \tag{2.24}$$

für die Ausgabevektoren $\mathbf{a}_\mu$ aus und gehe nach 1.

ε und ε' sind die schon bekannten Lernschrittweitenparameter aus Gleichung (2.4) und (2.12) und $g(k)$ und $g'(k)$ sind, analog zu g^{mix}, Funktionen des Nachbarschaftsranges k. Die Nachbarschaftsfunktionen $g(k)$ und $g'(k)$ haben den völlig analogen Effekt wie die Nachbarschaftsfunktionen $h_{\mathbf{rs}}$ und $h'_{\mathbf{rs}}$ im vorherigen Modell. Sie sorgen dafür, daß eine ganze Gruppe von Neuronen in der „Nachbarschaftsregion“ des „nächsten“ Neurons μ_0 am Adaptationschritt teilnimmt. Die Nachbarschaftsregion enthält nur solche Neuronen, die ähnliche Wertepaare lernen müssen und daher von

[6] In einer Parallelimplementation läßt sich k durch einfaches Abzählen erreichen. In einer sequenziellen Programmierung ist ein optimiertes Sortierverfahren (z. B. *Quicksort*) vorzuziehen, hier wird k identisch mit dem Rang, den das Neuron μ in der Liste $\{(||\mathbf{w}_i - \mathbf{x}||, i)\}$ einnimmt (für große N, siehe auch Seite 45).

denselben Lernschritten mutmaßlich profitieren können.
Eine geeinete Wahl ist analog zu (2.21)

$$g(k) = \exp\left(-\frac{k}{\lambda}\right) \quad \text{und} \quad g'(k) = \exp\left(-\frac{k}{\lambda'}\right). \tag{2.25}$$

λ und λ' übernehmen hier die Aufgabe von σ und σ' und skalieren die Größe der Nachbarschaftregionen. Auch sie werden anfänglich groß gewählt, um schnelles, grobes Lernen vieler Neuronen zu ermöglichen. Sie werden im Verlaufe der Lernphase langsam verkleinert, um eine Individualisierung der Neuronen und damit eine Detaillierung der Abbildung zu erreichen. λ^{mix} sollte stets kleiner als die mittlere Anzahl von „nächsten“ Nachbarn (siehe unten) gewählt werden. Indem die Reihenfolge der nächsten Nachbarzellen auf die Mittelung der Ausgabewerte Einfluß nimmt, erzielt man eine zusätzliche Verfeinerung der Tessellation im Vergleich zum *winner takes all network* (welches dem Limes $\lambda^{mix} \to 0$ entspricht).

2.5 Vergleich des erweiterten Kohonenmodell mit dem erweiterten „Neuronengasmodell“

Sowohl das erweiterte Kohonenmodell, als auch das erweiterte „Neuronengasmodell“ machen sich den Vorteil eines gleichzeitigen Lernens von vielen Neuronen zu eigen, der das Lernverhalten sowohl in Hinblick auf Konvergenzgeschwindigkeit als auch in Hinblick Robustheit verbessert.

Das Kohonenmodell benötigt eine genaue Festlegung der topologischen Struktur eines Gitters, in das die Neuronen eingebettet sind, und aus dessen gegenseitiger Lagebeziehung eine Nachbarschaft definiert wird. Es kann die Hauptmerkmalsdimensionen einer Eingangssignalverteilung entlang der Kartendimensionen abbilden (*feature mapping*), was einer wichtigen Abstraktionsleitung entspricht.

Das „Neuronengasmodell“ bedarf keinerlei Vorstrukturierung der Neuronen. Die Neuronen vermögen sich an beliebige Eingangsraummannigfaltigkeiten anzupassen. Dies soll an einem Simulationsbeispiel in Referenz [37] erläutert werden, das eine stark heterogene Eingangsraumtopologie aufweist. Die Mannigfaltigkeit der Eingangssignale X besteht aus der Vereinigungsmenge einer schlingenförmigen Stuktur (ein-dimensional), einem Rechteck (zwei-dimensional) und einem Quader (dreidimensional). Abbildung 2.6 zeigt eine Sequenz von vier Netzwerkzuständen: zu Anfang, nach $t = 5000$, $t = 10000$ und $t = 30000$ Lernschritten. Die Punkte markieren die Referenzvektoren $\mathbf{w}_\mu$, und die Linien verbinden jeweils nächste Nachbarn

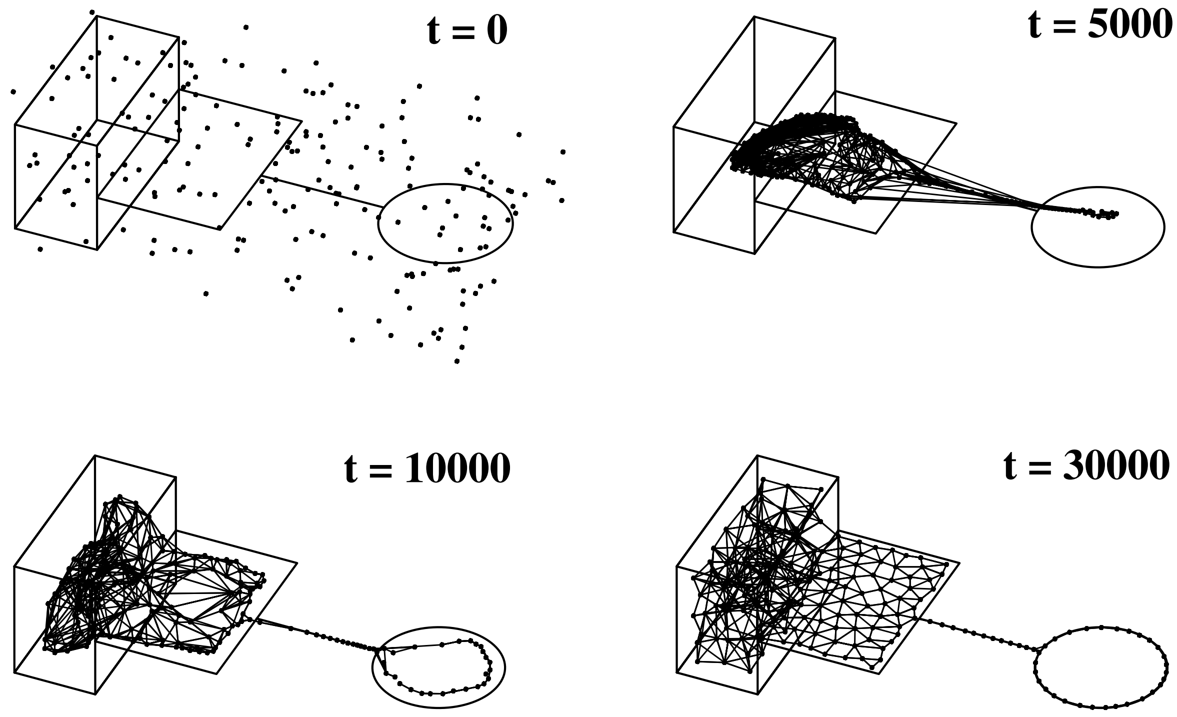


Abbildung 2.6: Das „Neuronengas“ Netzwerk quantisiert den Eingangssignalraum X, der hier eine stark heterogene Topologie aufweist. X besteht aus der Vereinigungsmenge einer schlingenförmigen Stuktur (1D), einem Rechteck (2D) und einem Quader (3D). Vier Momentaufnahmen des Netzwerkes von 200 Neuronen sind dargestellt: der Anfangszustand und die Zustände nach $t = 5000$, $t = 10000$ und $t = 30000$ Lernschritten. Die Punkte markieren die Referenzvektoren $\mathbf{w}_\mu$ und die Linien verbinden jeweils „nächste Nachbarn“. Seine „nächste Nachbarn“ bestimmt jedes Neuron für sich, indem es eine Liste von Verbindungen zu allen anderen Neuronen führt. In dieser Liste wird das „Datum“ des letzten „wichtigen Kontaktes“ geführt. Ist es älter als eine bestimmte Verfallzeit, wird dieser Nachbar nicht mehr als „nächster Nachbar“ betrachtet. Bei jedem Lernschritt entsteht genau ein „wichtiger Kontakt“ zwischen dem (*winner*-)Neuron $\mathbf{w}_{\mu_0}$ und dem Neuron $\mathbf{w}_{\mu_1}$ (siehe Seite 17). Am Ende der Lernphase reflektieren die Verbindungslinen die topologische Struktur von X und die mittlere Anzahl z von nächsten Nachbarn korrespondiert zur lokalen topologischen Dimension ($z[1D] = 2, z[2D] = 6, z[3D] = 8$; nach [37]).

(s.u.). Die Nachbarschaftsstruktur enthält die wesentliche Information über die Topologie der Mannigfaltigkeit X. Sie kann durch ein Zusatzverfahren explizit sichbar gemacht, und damit kann die Topologie von X erlernt werden Diese Information wird im Gegensatz zum Kohonenmodell nicht automatisch erzeugt. (siehe Bildtext und [37]). In manchen Anwendungen, wie z. B. Wegeplanung und Hindernisvermeidung [19], ist das Wissen über Nachbarschaftsverhältnisse der Referenzvektoren $\mathbf{w_r}$ von Bedeutung. Mit dieser Zusatzinformation läßt sich, ausgehend von einem Ausgabewert, auf das Urbild $\mathbf{w}$ und dessen Umgebung im Eingangsraum X rückschließen, um eventuell nach Variationen zu suchen.

Im Falle einer Zufallsinitialisierung der Referenzvektoren $\mathbf{w}$ besteht keinerlei Korrelationen zwischen benachbarte Neuronenindex μ bzw. $\mathbf{r}$ und deren $\mathbf{w}_\mu$ bzw. $\mathbf{w}_\mathbf{r}$. Die Neuronen im Kohonenmodell brauchen zu Beginn der Lernphase einige Zeit, um ihre topologische Ordnung herzustellen. Dieses Ordnen muß sorgfältig, d. h. langsam genug, geschehen, um eine vollständige „Entfaltung“ des Netzes zu erreichen und damit sogenannte „topologische Defekte“ auszuschließen (z. B. Falten und Knoten im Netz). Für das „Neuronengas“ besteht dazu kein Bedarf, die Zufallsinitialisierung der Referenzvektoren $\mathbf{w}_\mu$ *bestimmt* die anfängliche Nachbarschaftsrelationen zwischen Neuronen μ im Gas, welche dann direkt der Mannigfaltigkeit X angepaßt wird. Dies gibt dem „Neuronengas“ einen Zeitvorsprung vor dem Kohonenmodell.

Kapitel 3

Teil I: Vorhersage von Zeitserien

3.1 Einleitung und Überblick

In diesem Kapitel wollen wir uns mit der Vorhersage von Zeitserien beschäftigen. Zeitserien entstehen zum Beispiel durch stroboskopartiges Messen einer Observablen $x(t)$ eines dynamischen Systems. Wir nehmen an, die dem System zugrundeliegende Dynamik sei nicht oder nur unvollstandig bekannt. Anstatt ein Modell zu suchen, welches die Dynamik erklärt und dadurch eine Vorausberechnung ermöglicht, kann man versuchen, Vorhersagen auf empirische Regelmäßigkeiten des Systems zu stützen.

Diese Vorgehensweise macht man sich oft im täglichen Leben zu eigen, meistens unbewußt, etwa wenn ein Tennisspieler den Auftreffort eines anfliegenden Balls vorbestimmt, oder wenn ein Markler Börsentrendmuster ausnutzt, um künftige Kursentwicklungen abzuschätzen. Durch Betrachtung der momentanen Situation gestüzt auf Erfahrungen, aber ohne genaue Vorstellung der Ursachen will man die zukünftige Systementwicklung vorhersagen. Es ergeben sich zwei grundsätzliche Fragen: (i) Wie kann man die wichtigen Charakteristika einer Zeitserie so kodieren, daß eine Situation wiedererkennbar wird, wenn sich ein ähnlicher Systemzustand wiederholt? (ii) Wie kodiert man Erfahrung, um daraus eine Vorhersage rekonstruieren zu können? Im Folgenden werden wir den Systemzustand durch den momentanen Wert der Observablen und deren bisherigen Trends charakterisieren. Abschnitt 3.2 wird sich mit der Repräsentation dieser Situationen in sogenannte Zustandräumen auseinandersetzen und einige gebräuchliche Repräsentationen aufzeigen. In Abschnitt 3.3 werden einige wichtige Vorhersageverfahren kurz vorgestellt, eingeschlossen zwei neuronale Netzwerkverfahren.

In Abschnitt 3.4 wird eine im Rahmen dieser Arbeit entwickelte Methode [57] vorgestellt, die sich die beiden im vorigen Kapitel erläuterten neuronalen Netzwerke zu Nutze macht. Nach der Erläuterung des verwendeten Algorithmus sollen die numerischen Simulationsergebnisse für zwei Probleme vorgestellt werden: Das erste Problem beinhaltet die Vorhersage der Bewegung eines Teilchens in einem nichtlinearen 3-dimensionalen Potential. Das zweite Problem, besonders anspruchsvoll für einen Vorhersagealgorithmus, beinhaltet die Vorhersage einer chaotischen Zeitserie, der sogenannten *Mackey–Glass* Zeitserie. Eine Vorhersage dieser Zeitserie wurde auch von anderen neuronalen Netzwerkverfahren unternommen. In Abschnitt 3.3 werden die Verfahren erläutert und in Abschnitt 3.4 mit den von uns erzielten Resultaten verglichen.

Der nun folgende Abschnitt soll zunächst die umgangssprachlich oft vage bestimmten Begriffe Chaos und Zufall genauer erläutern.

3.1.1 Chaos und Zufall

Die Erforschung des Phänomens Chaos hat in den letzten Jahrzehnten unsere Vorstellung von Zufall und Determinismus stark verändert [8]. Als determiniert werden Systeme bezeichnet, deren zeitliche Entwicklung aus der Kenntnis der Anfangsbedingungen vollständig vorhersagbar ist. Ein Zufallssystem ist hingegen durch prinzipielle Unvorhersagbarkeit für größere Zeitspannen definiert. Die einzige wirkliche Quelle von Zufall, die wir kennen, ist die quantenmechanischen Unschärferelation.

Die Zufälligkeit der Dynamik eines Münzwurfes oder einer ins Roulett eingeworfenen Kugel beispielsweise, ist determiniert und besitzt nur wenige Freiheitsgrade. Ihre „Zufälligkeit“ besteht in der empfindlichen Abhängigkeit von den Anfangsbedingungen: Eine winzige Änderung der anfänglichen Bedingungen verstärkt sich exponentiell in der Zeit. Dies macht eine Vorhersage schwierig bzw. unmöglich. Wenn diese empfindliche Abhängikeit von den Anfangsbedingungen konsistent auftritt, spricht man von *determiniertem Chaos*. Determiniertes Chaos läßt sich in mathematischer Weise durch einen positiven Lyapunovexponenten charakterisieren. Der Lyapunov-Exponenten ist definiert als

$$\lambda_{lya} = \lim_{t\to\infty} \frac{1}{t} \int_t dt \ln |\dot{x}(t)| \ , \tag{3.1}$$

wobei $x(t)$ die Observable des System darstellt. Die Bedeutung des Lyapunov-Exponenten läßt sich an einem Ensemble von gleichen Testsystemen illustrieren. Sie mögen alle mit sehr ähnlichen Anfangsbedingungen (ε–Kugel) beginnen, deren Zustandsraumvektoren ein kleines Testvolumen im Zustandsraum ausfüllen. λ_{lya} beschreibt dann die gemittlte zeitliche (exponentielle) Entwicklung dieses Testvolumens: Ein negativer Lyapunov-Exponent beschreibt eine Kontraktion des Phasenraumvolumens, z. B. durch Dämpfung, ein positiver Lyapunov-Exponent λ_{lya} beschreibt ein exponentielles Anwachsen des Phasenraumvolumens. Exponentielles Anwachsen des Phasenraumvolumens, zusammen mit einer Faltung des Phasenraumes in sich selbst führt zu determiniertem Chaos.

Im Falle eines determinierten Chaos werden inhärente Unschärfen, Meßunschärfen und Rauschquellen exponentiell verstärkt und deren Auswirkungen können in endlich kurzer Zeit makroskopische Größenordungen erreichen. Diese Eigenschaft kann auch zur Idendifikation von chaotischen/nichtchaotischen Systemen mittels Vorhersagesystemen genutzt werden [53]. Die Vorhersagequalität bei nichtchaotischen Systemen hängt nur von genügender Kenntnis der Anfangsbedingungen ab, und der Vorhersagefehler wächst nur langsam mit der vorhergesagten Zeitspanne. Chaotische Systeme führen dagegen zu einem im Mittel exponentiellen Anwachsen des Fehlers.

Dissipative Systeme haben oft die Eigenschaft, daß ungestörte Trajektorien x_t in eine kleine Untermenge des Zustandraumes, *Attraktor* genannt, münden. Dies bedeutet eine zum Teil drastische Reduktion der Freiheitsgrade. Zum Beispiel eine Flüssigkeit mit (praktisch) unendlich vielen Freiheitsgraden kann unter bestimmten Umständen makroskopische Strömungsmuster ausbilden, die einem niedrig-dimensionalen Attraktor entsprechen. Ein berühmtes Beispiel bieten die 3-dimenisonalen Lorenzgleichungen, die ein turbulentes, konvektives System beschreibt.

3.2 Konstruktion des Zustandsraumes

Wir wollen uns nun dem Problem der Vorhersage der zeitlichen Entwicklung einer dynamischen Variablen $x(t)$ zuwenden. Wir nehmen an, daß die Dynamik von $x(t)$ durch eine kleine Anzahl von Freiheitsgraden d bestimmt wird[1]. Zunächst wird $x(t)$ durch „stroboskopisches“ Aufzeichnen mit einer konstanten Frequenz $1/\Delta t$ in eine Zeitsequenz $\{x(t_i)\}$ mit $t_i = i\Delta t$ $(i \in \mathbf{N})$ umgeformt. Aus dieser Zeitreihe betrachten wir die Datenpunkte in einem Zeitfenster der Breite m und konstruieren einen Zustandsvektor

$$\mathbf{x}_{state}(t) := \begin{pmatrix} x(t) \\ x(t-\Delta t) \\ x(t-2\Delta t) \\ \vdots \\ x(t-(m-1)\Delta t) \end{pmatrix} \in X \subseteq \mathbf{R}^m. \tag{3.2}$$

Man spricht dabei auch von *Einbettung* in einen Zusandsraum X_{state}.

[1] X kann auch die Dynamik eine Projektion eines höherdimensionalen Zustandsraumes auf den Unterraum des Attraktor der fraktalen Dimension d darstellen.

Nun stellt sich die Frage nach der optimalen Dimension m. Wird m zu klein gewählt, bleibt die Determiniertheit unsichtbar, und die Trajektorie erscheint zufällig. Eine untere Schranke ist $m \geq d$. Glücklicherweise kennt man auch eine obere Schranke für m. Ist d die Dimension der Mannigfaltigkeit, die den Attraktor enthält, so genügt $m = 2d + 1$, wie Takens zeigen konnte [54][2]. In der Praxis wird Δt und m oft durch Ausprobieren bestimmt, indem man mit einem kleinen, dann steigendem m anfängt, bis gute Ergebnisse gefunden werden. Etwas systematischere Prozeduren finden sich aufgrund informationstheoretischer Überlegungen, z. B. bei Fraser und Swinney [17] oder Broomhead und King [4], sowie Cremers und Hübler [7].

Die fraktale Dimension r des Attraktors kann mit verschiedenen Methoden bestimmt werden. Das klassische Skalierverfahren geht auf Grassberger und Procaccia zurück [20]. Dazu bestimmt man die minimale Anzahl n der Hyperkuben der Kantenlänge ε, die man braucht, um die Attraktor-(punkt)-menge M vollständig einzuschließen. Die Dimension r von M bestimmt sich dann zu

$$r = \lim_{\varepsilon \to 0} \frac{\ln\ n}{\ln\ 1/\varepsilon} \quad . \tag{3.3}$$

Um zum Beispiel eine Gerade der Länge L zu bedecken, benötigt man $n(\varepsilon) = \mathrm{L}/\varepsilon$ Kuben woraus $r = 1$ folgt. Für ein Quadrat der Seitenlänge L folgt entsprechend $n(\varepsilon) = \mathrm{L}^2/\varepsilon^2$ und $r = 2$. Nichtganzzahlige Dimensionen ergeben sich z. B. für Eiskristalle oder Küstenlinien, die Details auf allen Längenskalen zeigen.

Neben der Einbettung (3.2) gibt es viele andere Möglichkeiten, den Zustandsraum zu konstruieren, zum Beispiel durch Einbettung von Zeitableitungen

$$\mathbf{x}_{state'}(t) := \begin{pmatrix} x(t) \\ \dot{x}\ (t) \\ \ddot{x}\ (t) \\ \vdots \\ \frac{d^{(m-1)}x}{dt^{(m-1)}}(t) \end{pmatrix} \in \mathbf{R}^m, \tag{3.4}$$

die eine Phasenraumrepräsentation ergibt. Sie empfielt sich nicht für rauschbehaftete Daten, da hier Rauschen verstärkt wird. Eine Abwandlung dieser Repräsentation wurde für die in dieser Arbeit dargestellten Simulationen verwendet.

Die bisher vorgestellten Zustandsraummodelle beseichnet man als *autoregressive* Modelle, da sie den Zustandsraum direkt von den Daten konstruierten, was für deterministische Systeme am nächsten liegt. Dennoch sollen der Vollständigkeit halber auch noch sogenannte *moving average* Modelle erwähnen werden, die die Differenz von $x(t)$ und anderweitig vorhergesagten Werte von $\tilde{x}(t)$ einbetten (siehe Priestley [46], und Deppisch et al. für eine interessante rekursive Fehlereinbettungen [11]).

[2]Streng gültig nur für Systeme ohne äußeres Rauschen.

3.3 Vorhersage

Der nächste Schritt nach Festlegung des Zustandsraumes ist die Anpassung eines Modelles an die Vorhersage zugünftiger x-Werte. Dafür gibt es verschiedene Ansätze, die im Folgenden vorgestellt werden sollen.

Nach Voraussetzung läßt sich die Dynamik des Systems in der Form

$$x(t+T) = f_T^{true}\left(\mathbf{x}_{state}(t)\right) \tag{3.5}$$

beschreiben, dabei ist $\mathbf{x}_{state}$ der momentane Zustand und $x(t+T)$ der gesuchte x Wert. Gleichwertig kann man diese Extrapolation in der Zeit (3.5) auch als Abbildung *(mapping)* des Zustandsraumes in sich selbst interpretieren:

$$\mathbf{x}(t+T) = F_T^{true}\left(\mathbf{x}_{state}(t)\right). \tag{3.6}$$

Das Problem ist nun $x(t+T)$ vorherzusagen. Langfristigere Vorhersagen können bequem durch Iteration

$$\tilde{\mathbf{x}}(t+T) = F_T\left(\tilde{\mathbf{x}}_{state}(t)\right). \tag{3.7}$$

erzeugt werden, wobei "˜ " vorhergesagte Werte kennzeichnen soll. Möchte man kontinuierliche Zeitsysteme nicht verlassen, bleibt das Differential

$$\frac{d\mathbf{x}}{dt}(t) = f_T^{diff}\left(\mathbf{x}_{state}(t)\right). \tag{3.8}$$

anzunähern und zu integrieren.

3.3.1 Lineare Prädiktion

Einer der meist benutzten Vorhersagemethoden basiert auf adaptiver linearer *Prädiktion* (Vorhersage) [47]. Der vorhergesagte Wert $\tilde{\mathbf{x}}(t+1)$ zum Zeitschritt $t+1$ ist hier eine Linearkombination der m letzten Trajekorienpunkte

$$\tilde{x}(t+1) = \sum_{i=0}^{m-1} a_i x(t-i) + a_m. \tag{3.9}$$

(Ohne Einschränkung der Allgemeinheit setzen wir $\Delta t = T = 1$). Die Koeffizienten a_i sind ursprüglich unbekannt und werden durch ein lineares Regressionsverfahren oder eine adaptive statistische Iterationssmethode so bestimmt, daß der Wert der Kostenfunktion des quadratischen Prädiktionsfehlers

$$E_{cost} = \left\langle \xi^2 \right\rangle_{t'} = \left\langle \left(x(t+1) - \tilde{x}(t+1)\right)^2 \right\rangle_{t'}. \tag{3.10}$$

minimiert wird. Dazu eignet sich wieder die Methode des steilsten Gradienten (siehe p. 14).

Läßt sich das System durch ein lineares Modell gut beschreiben, wird der Satz von Koeffizienten a_i allmählich konvergieren und eine gute Approximation der Dynamik liefern. Viele interessante Phänomene genügen jedoch dieser Linearitätsbedingung nicht und eine genaue Vorhersage mit einem unveränderlichen Satz von a_i ist nicht möglich. Dies gilt zum Beispiel für chaotische Dynamik, wobei auch nichtlineare Terme in (3.9) aufgenommen werden müssen. Unter den vielen sich bietenden Möglichkeiten seien einige wichtige kurz erläutert.

3.3.2 Polynomiale globale Darstellung von f_T

Polynomiale globale Repräsentationen sind beliebt, da ihre Parameter mit *„least-square-fit“* Methoden angepasst werden können, um (3.10) zu minimieren. Die allgemeinste Form ist ein m-dimensionales Polynom d-ten Grades

$$f_T(\mathbf{x}_{state}) = Q_d(\mathbf{x}_{state}) = \sum_{i_1=0,\ldots,i_m=0}^{\sum_{j=1}^{m} i_j \leq d} a_{i_1,\ldots,i_m} \cdot x_1^{i_1} \cdot x_2^{i_2} \cdots x_m^{i_m}. \tag{3.11}$$

Diese Methode wurde von Gabor [18] erstmalig für autoregressive Modelle vorgeschlagen (siehe auch [7, 9]). Der Vorteil liegt in der garantierten Konvergenz (Theorem von Weierstraß [5]) von f_T solange d und m wächst. Der Nachteil: Die Anzahl der freien Parameter $a_{i_1,\ldots,i_m}$ ist $\binom{d+m}{m} = \frac{(d+m)!}{d!m!}$ und ihr *Anfitten* wird unpraktikabel, wenn d und m groß wird. Ferner steigt das Polynom fernab des Anfitbereiches mit der Potenz d an und leistet daher dort keine gute Extrapolation.

3.3.3 Rational polynomiale Darstellung (global)

Um letzteren Nachteil zu umgehen, läßt sich auch ein Bruch zweier Polynome Q_d/Q'_d als Repäsentation für f_T verwenden [5]. Das sich daraus ergebende Minimierungsproblem $Q'_d \cdot f_T^{true} - Q_d = 0$ ist linear in a, und daher mit Standardverfahren (*least square fit*) lösbar. Der Limes für $\|x_i\| \to \infty$ ist hier begrenzt.

3.3.4 Radiale Basisfunktionen

Radiale Basis Funktionen beschreiben f_T mit einem Satz von N radial um ihre Bases $\mathbf{w}_i$ abfallende, lokalisierte Funktionen ϕ_i. Diese Funktionen wurden zuerst von

Casdagli [5] für nichtlineare Modellierung vorgeschlagen (siehe auch Powell [45]). In vereinfachter Form ist f_T eine normierte Summe

$$f_T(\mathbf{x}_{state}) = \frac{\sum_{i=1}^{N} a_i \cdot \phi_i\left(\|\mathbf{x}_{state} - \mathbf{w}_i\|\right)}{\sum_{i=1}^{N} \phi_i\left(\|\mathbf{x}_{state} - \mathbf{w}_i\|\right)}, \tag{3.12}$$

mit Radialfunktionen ϕ_i, die nur vom Euklidischen Abstand von den Basen $\mathbf{w}_i$ abhängen; ein additives Polynom wird meist weggelassen. Für $\phi_i(r) = (r^2 + c_i^2)^{-\beta}$ mit $\beta > -1$ und $\beta \neq 0$ läßt sich die Existenz einer eindeutigen Lösung beweisen [5]. Moody und Darken [40] bevorzugten eine Gaußfunktion $\phi_i(r) = \exp(-(\frac{r}{c_i})^2)$ oder ein Fermifunktion $\phi_i(r) = \frac{1}{1+\exp(r/c_i - c_i')}$ aufgrund ihres stärkeren Abfalles für große r und der daraus resultierenden besseren Lokalität. Radiale Basis Funktionen bieten eine globale Interpolationstechnik für hochdimensionale Daten und zeichnen sich durch gute Lokalisierungseigenschaften aus.

3.3.5 Neuronale *feed-forward*-Netwerke

Lapedes und Farber [30] konnten die Nützlichkeit von neuronalen *feed-forward*-Netzwerken demonstrieren. Sie verwendeten ein mehrschichtiges Netz mit zwei verborgenen Ebenen *(hidden layers)* und sigmoiden Neuronen. Die Funktion f_T ergibt sich aus dem Zustandvektor $\mathbf{x}_{state} = \{x_1, x_2, \dots x_m\}$ (*input layer*) zu:

$$\begin{aligned} f_T(\mathbf{x}_{state}) &= \textstyle\sum_k a_k z_k - a_{0out} \\ \text{mit} \quad z_k &= \tanh\left(\textstyle\sum_j a_j y_j - a_{0h2}\right) \\ \text{mit} \quad y_k &= \tanh\left(\textstyle\sum_i a_i x_i - a_{0h1}\right), \end{aligned} \tag{3.13}$$

wobei y_j und z_k die Ausgangswerte der *hidden units* bezeichnen. Die Parameter $a_i, a_j, a_k, a_{0h1}, a_{0h2}$ und a_{0out} (*synaptischen Gewichte*) werden durch den sogenannten *Backpropagation*-Algorithmus [30, 52] bestimmt, der im wesentlichen ein Gradientenabstiegsverfahren darstellt, das geschickt die Kettenregel der Differentialrechnung ausnutzt.

3.3.6 Lokale Approximation

Die Idee der lokalen Approximation besteht darin, $f_T(\mathbf{x})$ lokal an der Stelle $\mathbf{x}_{state}$ zu approximieren. Das trivialste Beispiel wäre die „lokale Approximation 1. Ordnung“ oder „Nächste Nachbar-Approximation“: dies würde darin bestehen, aus den bisher gespeicherten Datensatz den nächsten Nachbarn (ähnlichsten) herauszusuchen und den Schätzwert $\tilde{x}(t+T)$ in übereinstimmung mit dem $x(t_{before}+T)$ zu wählen, was der Nachbar (aufgezeichnetes Ereignis) damals zum Zeitpunkt t_{before} als nächsten

Schritt tat. Dies läßt sich verbessern, indem man mehrere Nachbarn mitberücksichtigt und ihr Beitrag zu $\tilde{x}(t+T)$ entsprechend ihrem Abstand gewichtet. Nimmt man $s = m + 1$ nächste Nachbarn die den Simplex aufspannen, der $\mathbf{x}_{state}$ enthält, dann wird die Prozedur gleichwertig zur linearen Interpolation. Farmer und Sidorowich verwendeten diese „lokal-lineare“, oder „Approximation 2. Ordnung“ und berichten eine zusätzliche Verbesserung durch quadratische Interpolation [15]. Das System lernt hier einfach durch Abspeichern der rauschfreien Meßdaten, die Vorhersagen beinhalteten dann einen rechenintensiven Interpolationsalgorithmus in einer zu bestimmenden Untermenge der gespeicherten Datenpunkte. Dieser Umstand benachteiligt das Verfahren für zeitkritische Echtzeitanwendungen.

3.4 Vorhersage mit selbstorganisierenden Netzwerken

Im Folgenden wollen wir einen in dieser Arbeit entwickelten Vorhersagealgorithmus vorstellen (siehe auch [57]). Er läßt sich ebenfalls als lokales Approximationsschema klassifizieren und benutzt an Stützstellen gespeicherte, adaptive lineare Abbildungen. Nach der Erläuterung des verwendeten Algorithmus sollen die numerischen Simulationsergebnisse für zwei Probleme vorgestellt werden.

Das erste Problem beinhaltet die Vorhersage der Bewegung eines Teilchens in einem dreidimensionalen Potential der allgemeinen Form

$$V(x, y, z) = \Omega_x x^c + \Omega_y y^c + \Omega_z z^c, \quad \Omega_i = const, \quad c > 0, \text{ gerade.} \tag{3.14}$$

Diese Wahl enthält zwei interessante Extremfälle: $c = 2$ beschreibt einen harmonischen dreidimensionalen Oszillator, der mit einem einzigen Satz von Prädiktorkoeffizienten a_i beschrieben werden kann. Für $c \to \infty$ resultiert eine freie Bewegung in einem Kubus $-1 \leq x, y, z \leq 1$ mit ideal reflektierenden Wänden. Alle anderen Werte interpolieren zwischen diesen beiden Extremen.

Wir wollen zum Zweck der Vorhersage den Zustandsraum in disjunkte Zellen diskretisieren und für jede derartige Tesselationszelle eine lokal gültige lineare Abbildung f_T lernen (s. Seite 8). Ist die Raumaufteilung fein genug, wird die Linearitätsbedingung für jede Zelle eine gute Näherung darstellen und lokale Prädiktorvektoren $\mathbf{a}$ können iterativ gefunden werden. Die notwendige Zustandsraumdiskretisierung und das Finden von passenden Parametern $\mathbf{a}$ kann sehr effektiv durch selbstorganisierende, topologische Karten bewerkstelligt werden.

Für das nichtlineare, drei-dimensionale Potential (3.14) sind die Bewegungen in x, y und z separabel und werden von drei unabhängigen Prädiktionsmodulen

M_x, M_y, M_z parallel bearbeitet. Die Trajektorie wird auch hier (der Einfachheit halber) bei ganzen Zeitschritten ($\Delta t = 1$) diskretisiert und hintereinander folgende Datenpunkte zu Zustandsvektoren $\mathbf{x}_{hist}$, y_{hist} und z_{hist} („*history vector*"), kombiniert. Im folgenden wollen wir die Diskussion auf die x - Richtung beschränken, die anderen beiden Module arbeiten in gleicher Weise. Jedes Modul sieht „seine" Zustandsraumrepräsentation und macht — im allgemeinsten Falle — die Vorhersage für den nächsten Zeitschritt: (für M_x)

$$\tilde{x}_{t+1} = \sum_{i=0}^{m-2} a_{i,\mathrm{s}} x_{t-i} \; + a_{m-1,\mathrm{s}} \;=\; \mathbf{a}_\mathrm{s}^T \cdot \mathbf{x}_{hist} \tag{3.15}$$

mit dem Zustandsvektor

$$\mathbf{x}_{hist} := \begin{pmatrix} x_t \\ x_{t-1} \\ \vdots \\ x_{t-m+2} \\ 1 \end{pmatrix} \in \mathbf{R}^m. \tag{3.16}$$

Die Konstruktion mit einer konstanten $\mathbf{x}_{hist}$-Komponente (unterste Komponente in [3.16]) vereinfacht die mathematische Form. Für viele Systemdynamiken ist die Beteiligung eines konstanten Terms $a_{m-1,\mathbf{r}}$ redundant und der Ansatz

$$\tilde{x}_{t+1} = \sum_{i=0}^{m-1} a_{i,\mathrm{s}} x_{t-i} = \mathbf{a}_\mathrm{s}^T \cdot \mathbf{x}_{hist} \qquad \text{mit} \qquad \mathbf{x}_{hist} := \begin{pmatrix} x_t \\ x_{t-1} \\ \vdots \\ x_{t-m+1} \end{pmatrix} \in \mathbf{R}^m \tag{3.17}$$

genügt, was zur Reduktion der Freiheitsgrade genutzt werden kann.

Der wesentliche Punkt ist, daß der Koeffizientenvektor $\mathbf{a}_\mathrm{s}$ nicht konstant ist, sondern stattdessen von der Bewegung des Teilchens abhängt und zu jedem Zeitschritt ein Satz $\{a_{i,\mathrm{s}}\}$ mit dem Index $\mathbf{s}$ ausgewählt wird. Beides, die Auswahl als auch das adaptive Lernen der passenden Vorhersagevektoren $\mathbf{a}$ kann mit selbstorganisierenden Netzwerken vorteilhaft realisiert werden. Zunächst sei dies für den Fall des modifizierten Kohonennetzes demonstriert.

Betrachten wir dazu nochmal das planare Gitter von Neuronen in Abbildung 2.3 (Seite 12). Im vorliegenden Fall muß jedes dieser Neuronen zwei Daten speichern. Zum einen den Referenzvektor $\mathbf{w}$ der Dimension n und als Ausgabewert den Prädiktorvektor $\mathbf{a} \in \mathbf{R}^m$. Der Eingangsvektor $\mathbf{x}_{stim}$ beschreibt den momentanen Zustand des Systems. $\mathbf{x}_{stim}$ gehört nicht notwendigerweise derselben Repräsentation an wie $\mathbf{x}_{hist}$, was dann eine Verallgemeinerung von Gleichung (2.14) darstellt, die eine identische Wahl von $\mathbf{x}_{stim}$ und $\mathbf{x}_{hist}$ beinhaltet (für Repräsentation 3.16 natürlich nicht sinnvoll). Für das Problem (3.14) verwendeten wir, in Anlehnung an den physikalischen Phasenraum, den Vektor

$$\mathbf{x}_{stim} := \begin{pmatrix} x_t \\ \mathcal{K}_1(x_t - x_{t-1}) \\ \vdots \\ \mathcal{K}_{n-1}(x_t - x_{t-n+1}) \end{pmatrix} \in \mathbf{R}^n, \qquad \mathcal{K}_i = const. \tag{3.18}$$

und für $\mathbf{x}_{hist}$ die Repräsentation (3.17). Dabei sind die $\mathcal{K}_i$ Skalierungskonstanten, die alle Komponenten ungefähr in dieselbe Größenordnung bringen. Das Gitter operiert als ein *winner takes all network*, was andeutet, daß jeweils *ein einziges* Neuron für die Ausgabe zuständig ist ($g^{mix}(\mu_k) = \delta_{\mu_0 \mu_k}$). Es wird gemäß der Regel (2.11) ausgewählt, was den Eingaberaum, d. h. Zustandsraum, in Diskretisiertungszellen teilt, die durch die Menge von Systemzuständen gegeben sind, die dieselbe Zelle selektieren (s. Seite 13). Der Algorithmus wird in Abbildung 3.1 illustriert.

Weitere Vorhersagen lassen sich durch iteratives Anwenden der Prozedur mit geschätzten Zustandswerten erreichen, die entsprechende Komponenten in $\mathbf{x}_{stim}$ und $\mathbf{x}_{hist}$ ersetzen. Dies wird ebenfalls in Abbildung 3.1 illustriert.

In der Lernphase adaptieren die anfänglich zufällig gewählten Werte von $\mathbf{a}_\mathbf{r}$ und $\mathbf{w}_\mathbf{r}(\mathbf{r} \in M)$. Nach „Bekanntwerden“ eines neuen Wertes x_{t+1} werden, bevor ein neuer Prädiktionsschritt erfolgt, zwei Adaptationsschritte für alle Neuronen gemäß den Gleichungen (2.4, 2.12) durchgeführt. Die Minimierung der Kostenfunktion (vgl. 2.16)

$$E_{cost}[\mathbf{a}_\mathbf{r}] = \left\langle (x_{t'+1} - \mathbf{a} \cdot \mathbf{x}_{hist,t'})^2 \right\rangle_{t'} = \left\langle (x_{t'+1} - \tilde{x}_{t'+1})^2 \right\rangle_{t'}, \tag{3.19}$$

führt in diesem Fall zu

$$\Delta \mathbf{a}_\mathbf{r} = \left(x_{t+1} - \mathbf{a}_\mathbf{r}^T \cdot \mathbf{x}_{hist} \right) \frac{\mathbf{x}_{hist}}{\|\mathbf{x}_{hist}\|^2}. \tag{3.20}$$

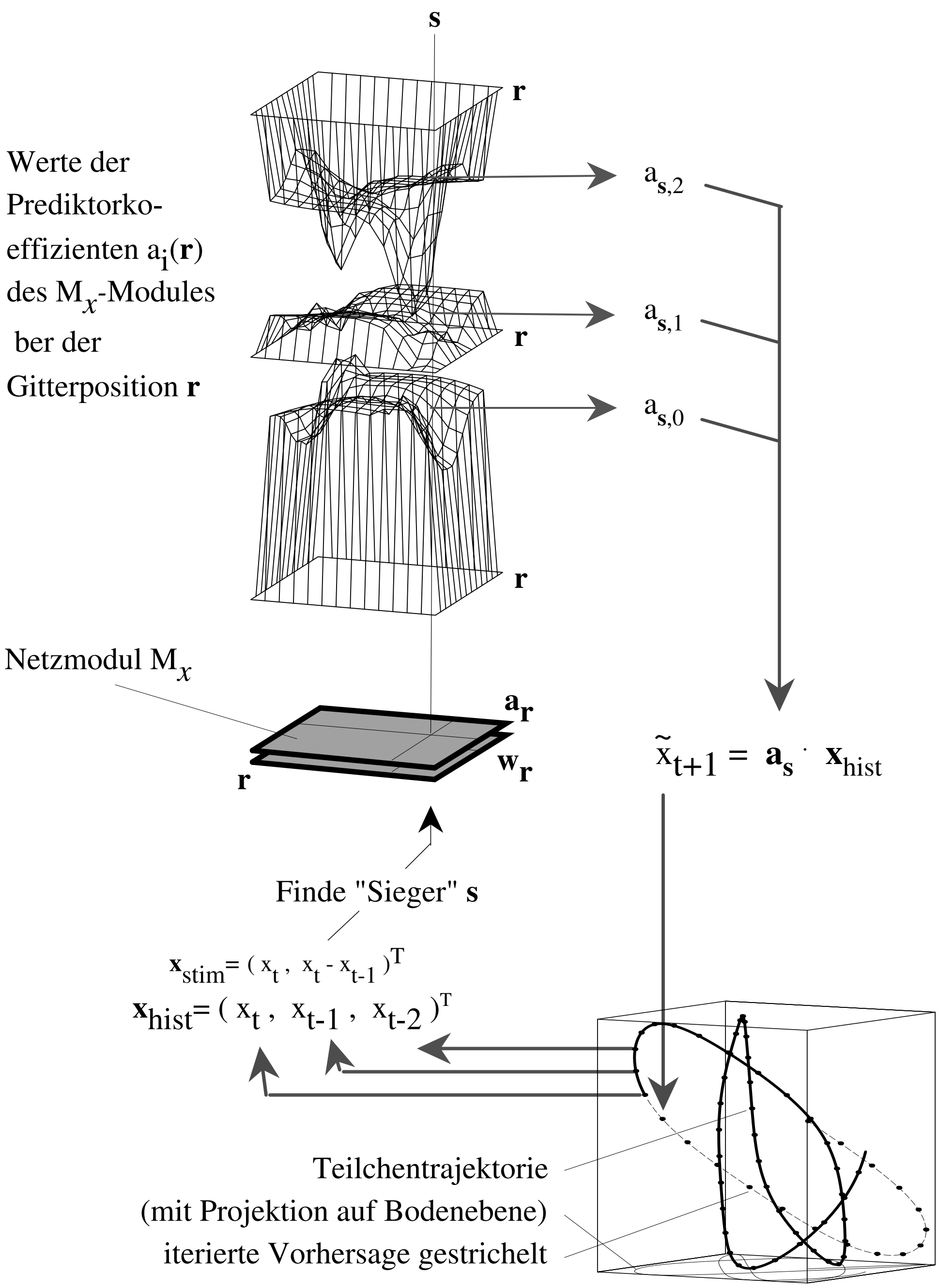


Abbildung 3.1: Der rekursive Vorhersagemechanismus am Beispiel einer Teilchentrajektorie. Aus den aktuellen letzten Datenpunkten der diskretisierten Trajekorie (*rechts unten: durchgezogene Linie*) werden die Zustandsvektoren $\mathbf{x}_{hist} \in \mathbf{R}^m$ und $\mathbf{x}_{stim} \in \mathbf{R}^n$ (hier m=3, n=2) kombiniert. Das zuständige *winner* Neuron **s** liefert die Koeffizenten $a_{\mathbf{s},i}$ für die Vorhersage des nächsten Zeitschrittes $t+1$. Die Prädiktion läßt sich rekursiv fortsetzen, indem man die geschätzen Trajektorienpunkte zur Grundlage der nächsten Vorhersagen macht (*gestrichelte Line*). (*Links oben:*) Man beachte die glatte Oberglächen der Koeffizientengebirge $a_{0,\mathbf{r}}$, $a_{1,\mathbf{r}}$ und $a_{2,\mathbf{r}}$, die die topologieerhaltende Eigenschaft der selbstorganisierenden Karten wiederspiegeln.

3.5 Simulationsergebnisse für die Teilchentrajekorie

In diesem Abschnitt sollen die Simulationsergebnisse des Algorithmus für das Problem (3.14) der Teilchentrajektorie im 3D Potential mit $c = 2$, $c = 4$, $c = 8$ und $c = 16$ vorgestellt werden (s. Seite 30). Aus Gründen der Vergleichbarkeit wurden für alle vier Fälle die gleichen Parameter gewählt: ein periodisches, quadratisches Gitter der Kantenlänge $L = 12$, also insgesamt 144 Neuronen mit einem zwei-dimensionalen Eingangsraum ($n = 2$, $\mathcal{K}_i = 1$) und einem linearen Prädiktionsverfahren, das die letzten drei Trajektorienpunkte (i.e. $m = 3$) berücksichtigt. Während der ersten

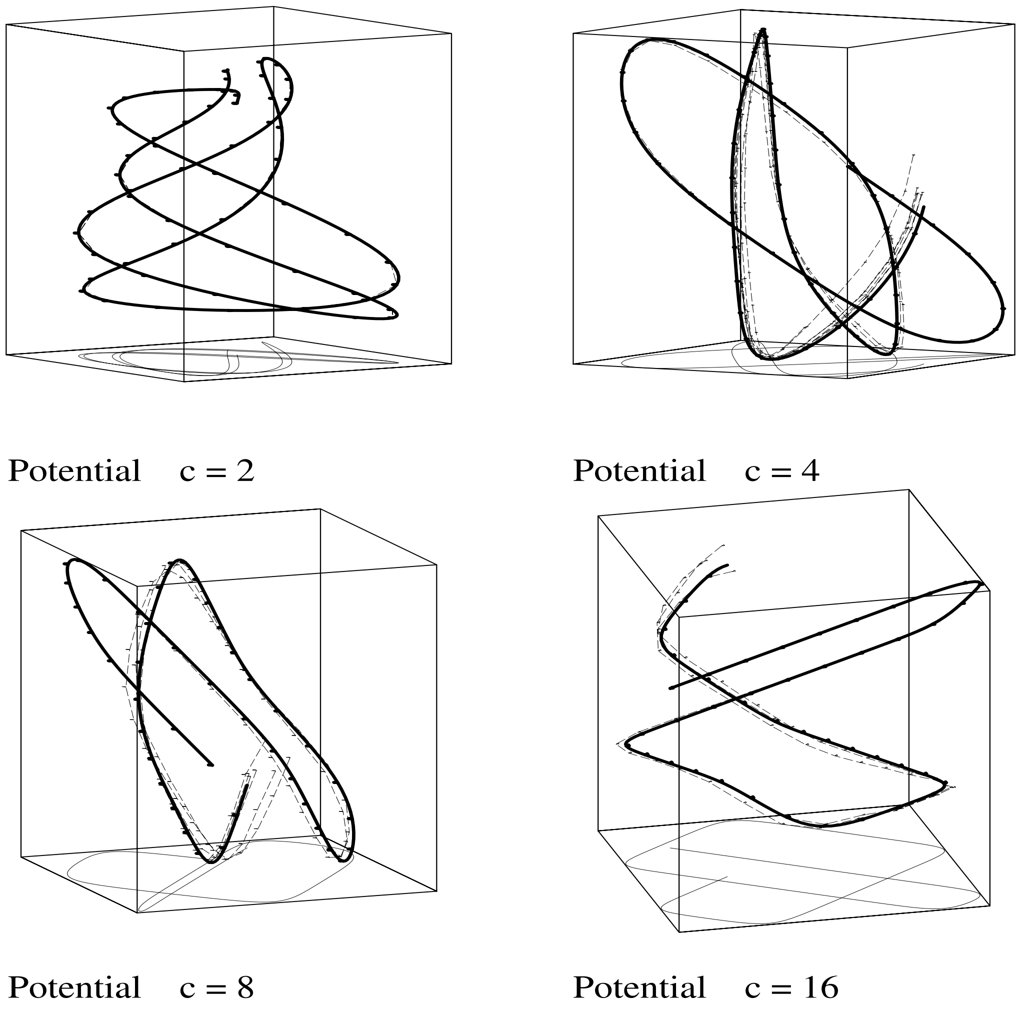


Abbildung 3.2: Beispiele von Trajektorien (*breite Linien*) gemeinsam mit ihren Projektionen auf die Grundebene (*feine Linien*) und jeweils mehreren, vom Netzwerke vorhergesagten Trajektorien (*gestrichelte Linien*). Die Meßpunkte sind mit Marken gekennzeichnet.

500 Lernschritten wurden die Parameter[3] $\sigma = \sigma(t)$ und $\sigma' = \sigma'(t)$ linear von ihren anfänglichen Werten $0.7L$ und $0.8L$ zu den jeweiligen Zwischenwerten $0.2L$ und $0.1L$ verkleinert und schließlich, etwas langsamer, zu ihren Endwerten $0.05L$ and $0.02L$ während der lezten 4500 Lernschritten verringert. Der Parameter $\varepsilon = \varepsilon(t)$ wurde in derselben Weise verändert, mit Anfangs-, Zwischen- und Endwert von 0.5, 0.09 bzw. 0.001. Der Parameter $\varepsilon' = 0.7$ wurde konstant gehalten. Die Referenzvektoren $\mathbf{w}$ wurden mit Zufallswerten initialisiert, die Prädiktorvektoren $\mathbf{a}$ wurden anfänglich gleich Null gesetzt. Diese Wahl von Anfangswerten ist relativ unkritisch. Das Netzwerk wurde mit einer Serie von Trajektorien trainiert, deren Ausgangsbedingungen zufällig bestimmt wurden. Dabei war die Gesamtenergie des Teilchens so beschränkt, daß ein Verlassen des Raumes $-1 < x, y, z < 1$ vermieden wurde.

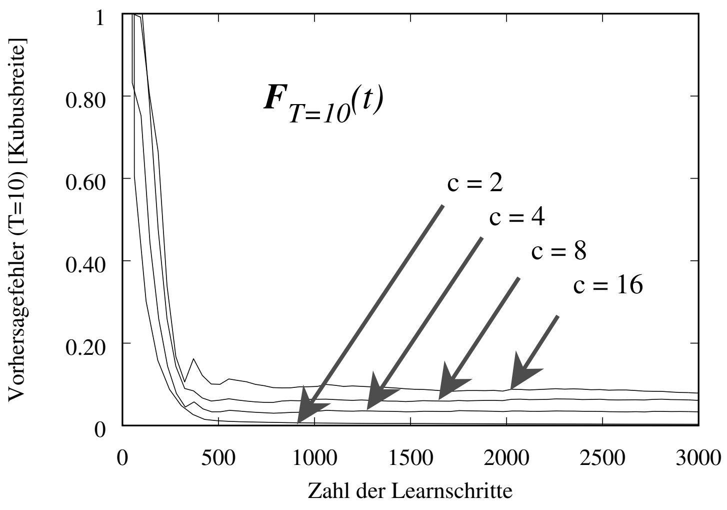
Einige Beispieltrajektorien, zusammen mit erzielen Vorhersagen (*gestrichelte Linien*) zeigt Abbildung 3.2. Trotz der relativ komplizierten Form ist das Netzwerk nach genügend Training in der Lage, zuverlässige Vorhersagen zu treffen. Im Falle von $c = 2$ ist die vorhergesagte Trajektorie praktisch ununterscheidbar von der wahren Trajektorie. Man kann zeigen, daß nur ein *einziger* Koeffizentenvektor $\mathbf{a_r}$ nötig ist, um den harmonischen Oszillator vorherzusagen. Alle Neuronen müssen demzufolge denselben Ausgabewert lernen. In allen anderen Fällen $c > 2$ reicht ein einzelner Prädiktorvektor $\mathbf{a_r}$ nicht aus, und die Neuronen haben verschiedene Werte zu lernen. Dies ist im Bild 3.1 für den Fall $c = 16$ zu sehen (Seite 33, *links oben*). Jede der drei Gebirgsflächen repäsentiert einen Satz von Prädiktorkoeffizienten $a_{i,\mathbf{r}}$ $(i = 1, 2, 3)$ über dem Index $\mathbf{r}$ aus dem 12×12-Gitter des M_x-Modules. Für verschiedene Regionen der Trajektorie müssen verschiedene Koeffizienten gelernt werden, was dem Netzwerk offensichtlich auch gelingt. Die relativ glatte Oberfläche des Gebirges spiegelt dabei den topologieerhaltenden Charakter des Kohonen-Algorithmus wieder.

Um die Vorhersagegenauigkeit zu testen, wird der Euklidische Fehler

$$F_T = \sqrt{(x_{t+T} - \tilde{x}_{t+T})^2 + (y_{t+T} - \tilde{y}_{t+T})^2 + (z_{t+T} - \tilde{z}_{t+T})^2} \tag{3.21}$$

für einen Vorhersagezeitpunkt T Schritte in der Zukunft betrachtet. Abbildung 3.3a zeigt F_{10} ($T = 10$ gemittelt über mehrere Versuche) als Funktion der Anzahl der Lernschritte. Wie man sieht, veringert sich der Vorhersagefehler von einem hohen Anfangswert zu einem viel kleineren Wert und, wie zu erwarten, ist das asymptotische Verhalten um so besser, je kleiner die Werte von c sind. Eine andere Möglichkeit der Auftragung ist, den Fehler F_T nach dem Lernen als Funktion der Zahl von Vorhersageiterationen T in die Zukunkt darzustellen, Abbildung 3.3b zeigt die entsprechende

[3] Für eine Liste der häufig verwendeten Parameter siehe Seite 109.



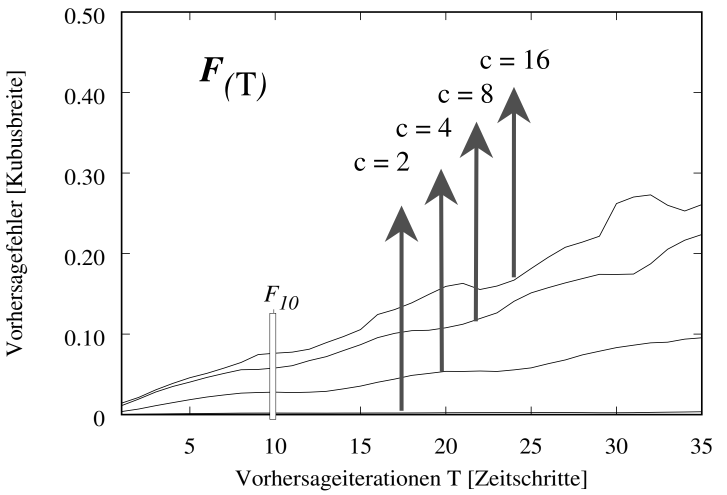


Abbildung 3.3: (a, *oben:*) Der Vorhersagefehler F_{10} für 10 Schritte in die Zukunft in Abhängigkeit von der Anzahl der Lernschritte. (b, *unten:*) Der Vorhersagefehler F_T in Abhängigkeit von der Zahl der Vorhersageiterationen T.

Abhängigkeit. In allen vier dargestellten Fällen beherrscht das Netzwerk die Vorhersage sehr gut, solange T nicht zu groß wird. Für große T bleibt die Vorhersage nur für Potentiale niederer Ordnung genau.

Abbildung 3.4 stellt die Diskretisierung des zwei-dimensionalen Eingangsraums durch das M_x-Modul für den Fall $c = 16$ dar. In beiden Diagrammen wurden die Zentren der Diskretisierungzellen, oder Referenzvektoren $\mathbf{w_r}$ auf den Eingangsraum „projeziert“. Analog zu Abbildung 2.6 wurden die Referenzvektoren, die zu nächsten Nachbarn gehören, mit Linien verbunden. Eine repräsentative Anzahl von Eingangsvektoren $\mathbf{x}_{stim}$, die durch die Teilchenbewegung erzeugt wurden, sind durch Punkte markiert. Die linke Seite zeigt den zufällig initialisierten Anfangszustand, rechts sieht man den topologisch geordneten Zustand nach dem Lernen. Man kann deutlich er-

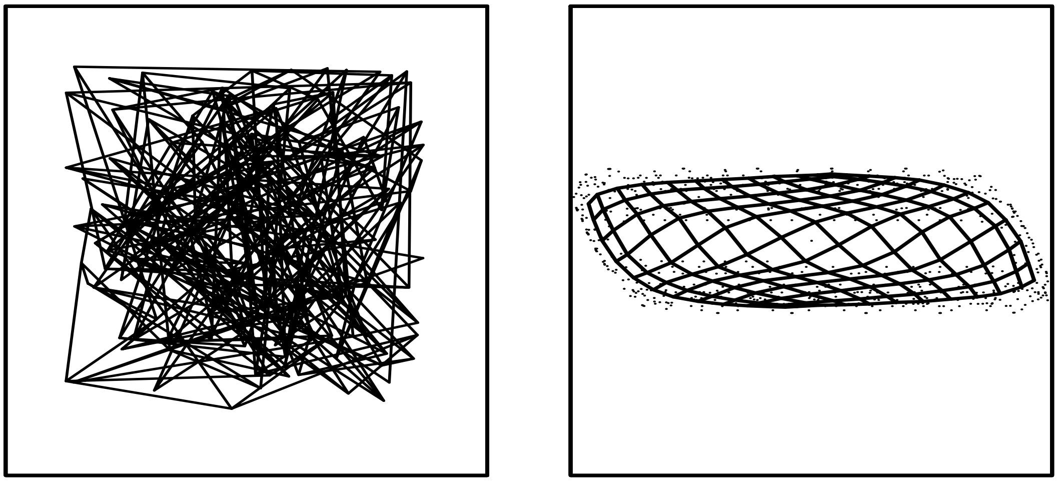


Abbildung 3.4: Korrespondenz zwischen dem Eingangsraum und dem Neuronengitter vor (*links*) und nach dem Lernen (*rechts*). Die *Netzknoten* stellen die in den 2-dimensionalen Eingangsraum projezierten Referenzvektoren $\mathbf{w_r}$ dar, verbunden mit den Referenzvektoren ihrer jeweiligen nächsten Gitternachbarn. Die *Punkte* repräsentieren eine Auswahl von bereits „gesehenen“ Eingangsvektoren $\mathbf{x}_{stim}$ (12×12 Gitter des M_x-Modules).

kennen, wie sich die Referenzvektoren in Regionen mit hoher Punktdichte häufen, was die nachfrage-orientierte Verteilung der Neuronen und damit die ökonomische Allokation der "Rechenresourcen“ demonstriert.

3.6 Simulationsergebnisse für die Mackey–Glass Zeitserien

Wir wollen uns nun der zweiten Problemklasse zuwenden, die für einen Vorhersagealgorithmus weit anspruchsvoller ist, nämlich die Vorhersage einer Zeitserie, die durch Integrieren der verzögerten Differentialgleichung

$$\dot{x}(t) = -\beta \cdot x(t) + \alpha \cdot \frac{x(t)}{1 + x(t-\tau)^{10}} \tag{3.22}$$

entsteht. Sie ist nach Mackey und Glass benannt, die sie erstmalig für die Beschreibung der Konzentration von weißen Blutkörperchen im Blutkreislauf verwendeten [33, 34]. β beschreibt dabei die biologische Zerfallszeit der weißen Blutzellen und der $\alpha-$ Term den Reproduktionsprozeß im Knochenmark. Für die renormierte Zellkonzentration $x(t) < 1$ werden die Stammzellen im Knochenmark proportional zu x zur Zellteilung stimuliert und nach der Reifungszeit τ, werden die entstandenen Zellen in den Blutkreislauf entlassen. Für Zellkonzentration $x \tilde{>} 1$ wird die Stimulation durch den hochgradig nichtlinearen Nenner (Potenz 10) stark unterdrückt. Durch krankhafte Verlängerung der Zellreifungszeit (τ) kann der gesamte Steuerungszyklus außer Kontrolle geraten.

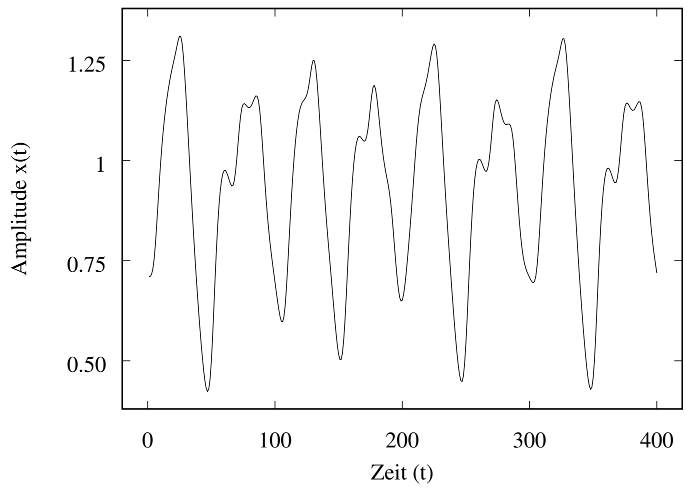


Abbildung 3.5: Vierhundert aufeinanderfolgende Zeitpunkte der Mackey–Glass Zeitserie mit Verzögerungszeitparameter $\tau = 17$. In diesem Parameterbereich zeigt die Kurve quasi-periodisches Verhalten, d. h. keine zwei Zyklen sind jemals exakt identisch.

Abbildung 3.5 zeigt die Zeitserie, die durch Integration der definierenden Gleichung (3.22) mit den Parametern $\tau = 17$, $\alpha = 0.2$ und $\beta = 0.1$ entsteht. Für diesen Parametersatz erzeugt die Mackey–Glass Gleichung eine chaotische, quasi-periodische Zeitserie mit einer fraktalen Attraktordimension $r = 2.1$ ([14], siehe auch Seite 25). Quasiperiodizität bedeutet, daß keine zwei Zyklen jemals exakt identisch sind.

Diese Zeitserie ist sehr schwierig vorherzusagen, klassische Vorhersagetechniken wie zum Beispiel das *globale lineare Autoregressionsschema* oder das *Gabor-Volterra-Wiener* Verfahren versagen gänzlich für Vorhersagezeiträume T größer als die *charteristische Zeit* t_{char} [14]. Dabei ist die charakterische Zeit t_{char} durch das reziproke Mittel des Frequenzspektrums der Zeitserie gegeben, in vorliegendem Fall ($\tau = 17$) ist $t_{char} \approx 50$. Ein Versagen einer Vorhersage läßt sich so formulieren, daß die Vorhersage genauso gut bzw. schlecht ist, wie die konstante Vorhersage $\tilde{x}(t+T) = \langle x \rangle$, wobei $\langle x \rangle$ dem Mittelwert der Zeitserie darstellt.

Dies läßt sich durch den *normalisierten Vorhersagefehler* F_{norm} genauer formulieren. F_{norm} ist als die Quadratwurzel des mittleren Quadrates (rms) des Vorhersagefehlers bezogen auf die Standardabweichung $x(t) - \langle x \rangle$ definiert

$$F_{norm} \stackrel{\text{def}}{=} \frac{\sqrt{\langle (\tilde{x}_{t'+T} - x_{t'+T})^2 \rangle_{t'}}}{\sqrt{\langle (x_{t'} - \langle x \rangle)^2 \rangle_{t'}}}. \tag{3.23}$$

Ein Vorhersagefehler $F_{norm} = 0.0$ entspricht einer perfekten Vorhersage und $F_{norm} \approx 1.0$ bedeutet keine Korrelation zwischen der vorhergesagten und den wahren Werten, und damit ein vollständiges Versagen der Vorhersage.

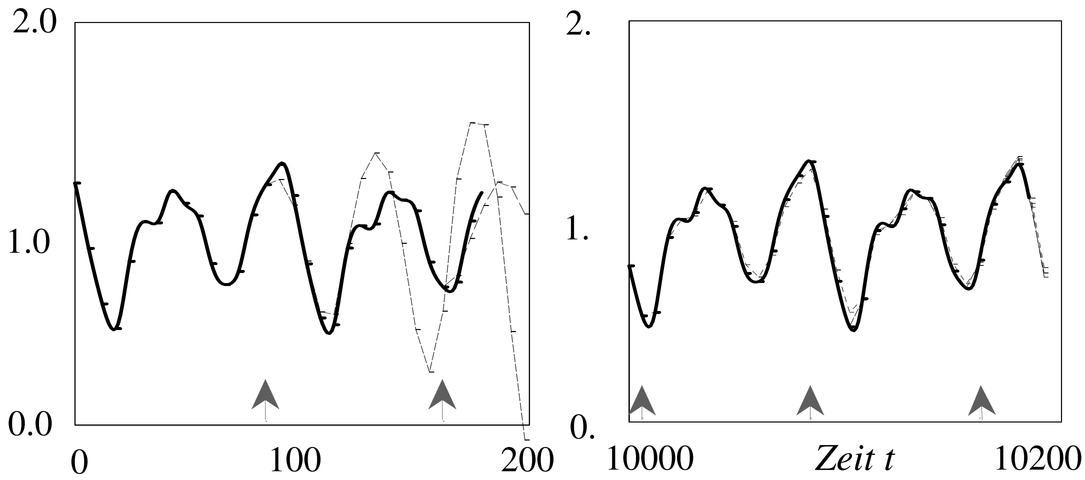


Abbildung 3.6: (a,b) Die Mackey–Glass Zeitserie ($\tau = 16$) gemeinsam mit Trajektorien (*gestrichelt*), die an den mit *Pfeilen* markierten Zeitpunkten vorhergesagt wurden (*links*) unmittelbar zu Beginn der Lernphase und (*rechts*) am Ende der Lernphase von 10000 Schritten. Die Meßpunkte im Abstand $\Delta t = 6$ sind durch kleine Marken gekennzeichnet.

Verkleinert man den Verzögerungsparameter um eins auf $\tau = 16$, ergibt sich eine Zeitentwicklung, die ganz ähnlich chaotisch aussieht, jedoch tatsächlich periodisch ist. Die entsprechende Zeitentwicklung ist in Abbildung 3.6 dargestellt, gemeinsam mit den dünnen gestrichelten Trajektorien, die an den mit Pfeilen markierten Zeitpunkten in die Zukunft vorhergesagt wurden. Auf der linken Seite sieht man zwei Vorhersageversuche unmittelbar nach Beginn der Lernphase und rechts drei Vorhersageversuche, die am Ende der Trainingsphase kaum unterscheidbar dicht bei der wahren Trajektorie liegen.

Aus Gründen der Vergleichbarkeit wurden (sofern nicht anders erwähnt) für alle dargestellten Mackey–Glass Simulationen die Parameter $\Delta t = T = 6$ und die Einbettung (3.18) mit $m = 4$ ($\mathcal{K}_{1,2,3} = 3$) und (3.16) mit $n = 5$ gewählt. Um verschiedene Projektionen des Eingangsraumes einheitlich darstellen zu können, wurden die Daten anfangs um den Nullpunkt gemittelt, d. h. der Mittelwert $\langle x \rangle$ abgezogen (dies kommt einem Herausfiltern des Gleichstromanteils gleich). Unter Verwendung des erweiterten Kohonenmodells und des „Neuronengas“-Modells[4] lassen sich interessante Vergleiche ziehen.

[4] Die Simulationsergebnisse für das „Neuronengas“-Modell sind in Zusammenarbeit mit Stan Berkovich entstanden [2].

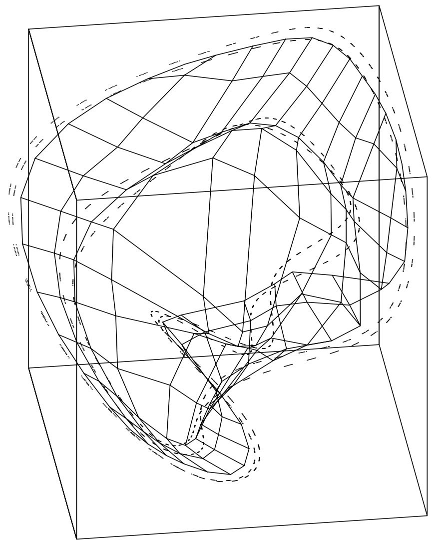


Abbildung 3.7: Illustration der Korrespondenz des Eingangsraumes (Orthogonalprojektion der 2., 3. und 4. Komponente) und des 12×12 Kohonengitters nach 10000 Lernschritten mit Daten der Mackey–Glass Zeitserie ($\tau = 16$, *feine Punkte*).

Abbildung 3.6 zeigt ein 12×12 Kohonennetz nach 10000 Lernschritten ($\tau = 16$) in der zu Bild 3.4 (Seite 37) analogen Auftragung. Die Punkte, die auf der kompliziert gewundenen, getrichelt erscheinenden Kurve liegen, gehören zur Attraktormenge. Die Darstellung zeigt eine Projektion der 2., 3. und 4. Komponente der „gesehenen" Zustandvektoren $\mathbf{x}_{stim}$ *(feine Punkte)* und der Referenzvektoren $\mathbf{w_r}$ *(Gitterknoten)* in den Eingangsraum X_{stim} [siehe (3.18)]. Das zwei-dimensionale Kohonennetz hat „große Mühe" sich dem außerordendlich komplizierten, ein-dimensionalen Attraktor anzupassen. Deutlicher wird die Form des Attraktors im Vergleichsbild des „Neuronengas" Modells in Abbildung 3.8. Die Sequenz von Netzwerkszuständen nach $l = 10$, $l = 200$, $l = 1600$ und $l = 20000$ Lernschritten zeigt, wie die Wolke von 200 zufällig inititialisierten Referenzvektoren $\mathbf{w}_r$ anfängt zu kollabieren, sich entlang der Hauptachse des Attraktors ausbreitet, und sich allmählich aufspaltet, um sich schließlich entlang dem Attraktor gleichmäßig zu verteilen. Die bisher „gesehenen" Meßpunkte $\mathbf{x}(l \cdot \Delta t)$ sind als feine Punkte markiert, die insbesondere im letzten Bild die filigrane Struktur des doppelt geschlungenen Attraktors deutlich machen.

Für die genaue Anpassung an eine komplizierte Topologie ist die Steuerung der Lernparameter $\varepsilon(l)$ und $\lambda(l)$ kritisch. Wir erhielten für einen in Abhängigkeit der Lernschrittzahl l stückweise exponentiell fallenden Parameterverlauf $p \in \{\varepsilon, \lambda\}$

$$p(l) = p_i \left(\frac{p_{i+1}}{p_i}\right)^{\frac{l-l_i}{l_{i+1}-l_i}} \qquad \text{mit} \qquad l_i \leq l < l_{i+1} \tag{3.24}$$

zwischen den $\{(l_j, \varepsilon_j)\}$ Stützstellen $\{(0, 0.4), (2000, 0.4), (20000, 0.28)\}$ und den $\{(l_j, \lambda_j)\}$ Stützstellen $\{(0, 47), (2000, 1.7), (7500, 0.45), (20000, 0.2)\}$ sehr gute Ergebnisse. Dieser Verlauf ist empirisch in einer Reihe von Versuchsläufe am Computerbildschirm bestimmt worden. Dabei wurde versucht, eine optimale Balance zwischen schnellem kollektivem Adaptieren und schneller Individualisierung der Neuronen so zu erreichen, so

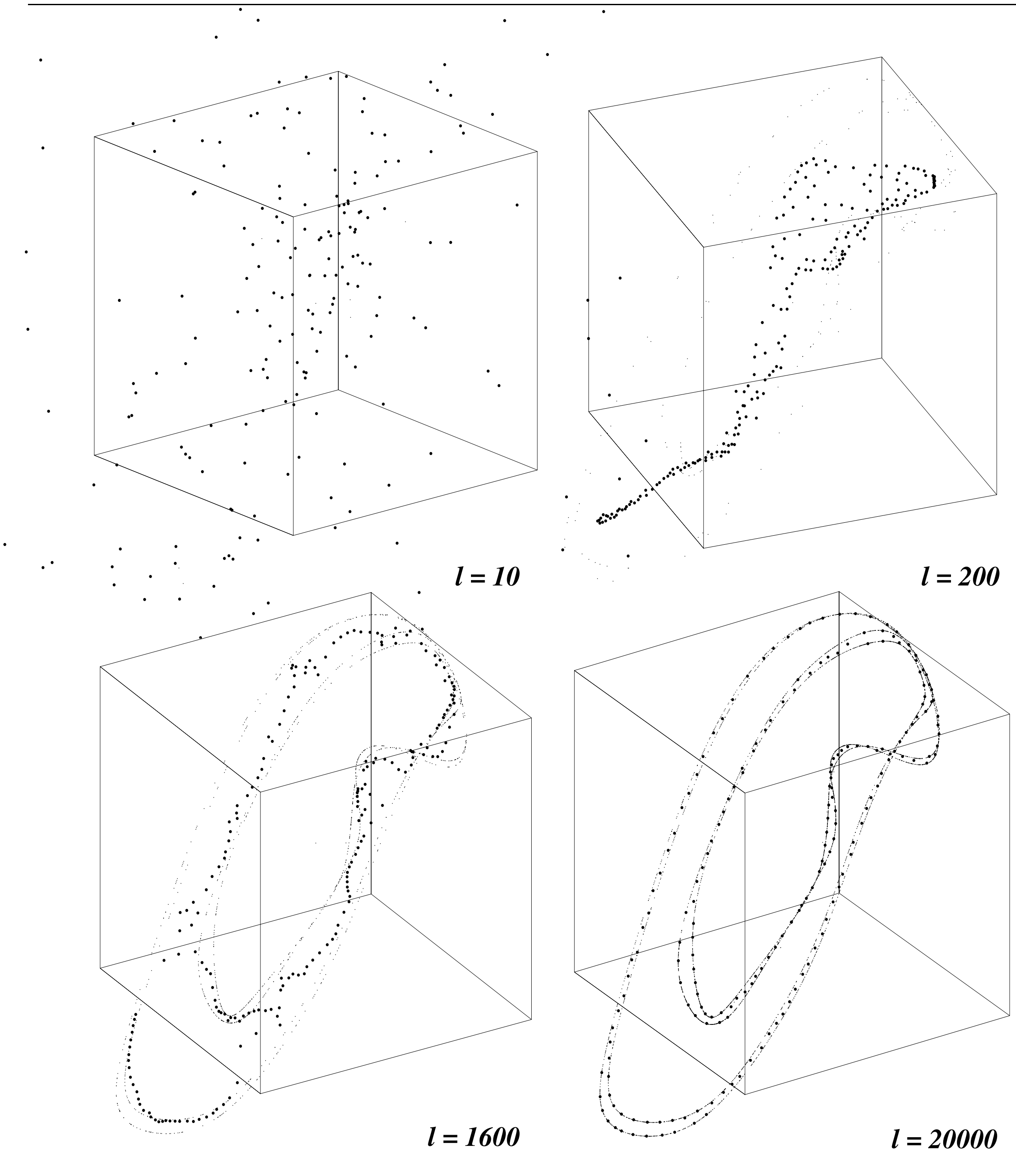


Abbildung 3.8: Sequenz von Netzwerkzuständen nach $l = 10$, $l = 200$, $l = 1600$ und $l = 20000$ Lernschritten. Das Netzwerk bestehen aus 200 Neuronen, die nach dem „Neuronengas“ -Algorithmus mit Mackey–Glass-Zeitseriendaten ($\tau = 16$) trainiert wurden. Die 2., 3. und 4. Komponente deren Referenzvektoren $\mathbf{w}_\mu$ wurden in den Eingangsraum projeziert (*größere Punkte*). Die bisher „besuchten“ Meßpunkte $\mathbf{x}_{stim}(i \cdot \Delta t)$ sind als feine Punkte markiert. Beschreibung siehe Text.

daß insgesamt eine rasche, gleichmäßige Verteilung der „Rechen-resourcen“ auf dem Attraktor erzielt wird.

In Abbildung 3.9 wird die Konsequenz dieser besseren Verteilung der Rechen-Resourcen für das erweiterten „Neuronengas“ im Vergleich zum erweiterten Kohonen-

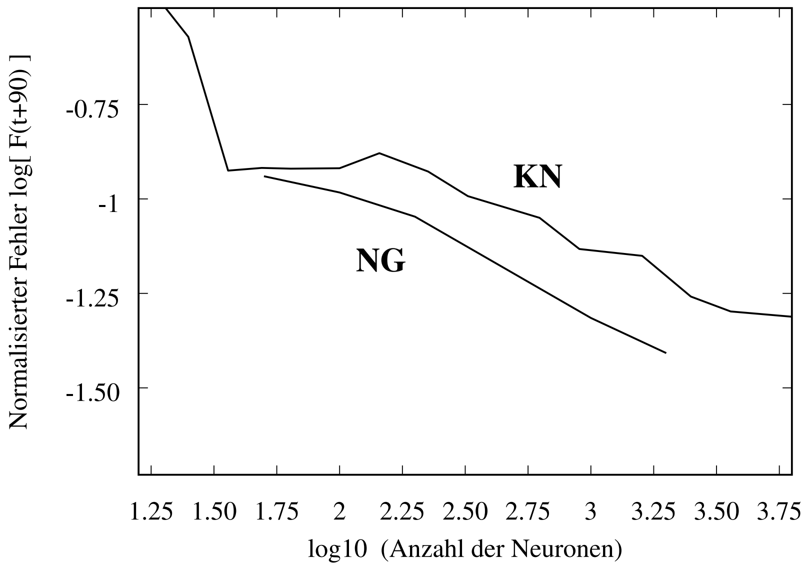


Abbildung 3.9: Skalierungsverhalten der Vorhersagegenauigkeiten F_{norm} des erweiterten Kohonennetzalgorithmus (KN) und des erweiterten „Neuronengas"-Algorithmus (NG) gegen die Zahl der Neuronen mit Daten des Mackey–Glass Attraktors ($\tau = 16$). Im doppel-logarithmischen (zur Basis 10) Graph ist der normalisierte Vorhersagefehler $F_{norm}(T = 90)$ gegen die Anzahl der verwendeten Neuronen aufgetragen. (In beiden Kurven wurde die Einbettung (3.17) mit $n = 4$ gewählt)

netz, für den komplexen, fadenförmigen Mackey–Glass Attraktor mit $\tau = 16$ demonstriert. Im doppelt-logarithmischen Graph ist der normalisierte Fehler F_{norm} für eine Vorhersage $t + 90$ Zeitschritte in die Zukunft gegen die Anzahl der am Lernen beteiligten Neuronen aufgetragen. Um 90 Zeitschritte in die Zukunft vorherzusagen, muß die Vorhersage 15 Mal iteriert werden (siehe Schema auf Seite 33, hier mit $\Delta t = 6$). Wie man aus dem Vergleich der beiden Kurve sieht, hat das „Neuronengas" einen Vorteil, der auf die komplizierte Struktur dieses Datensatzes zurückzuführen ist, wie auch durch Vergleich der Bilder 3.6 und 3.8 plausibel wird.

Eine andere Art der Visualisierung der Vorhersagefähigkeit ist in Abbildung 3.10 in Form von Korrelationsdarstellungen zu sehen. Hier sind für sechs verschiedene Iterationsschrittzahlen $T/\Delta t$ jeweils 100 vorhergesagte Differenzen $\tilde{x}(t + T) - x(t)$ gegen die jeweiligen wahren Werte $x(t + T) - x(t)$ aufgetragen. Für *eine* Iteration ($T = 6$) ergibt sich eine sehr präzise Vorhersage, die sich als diagonal gepunktete Linie darstellt. Für mehrfach iterierte, längere Zeitvorhersagen weitet sich die Linie langsam zu einer länglichen Wolke auf, die für 38 Iterationen ($T = 228 = 38\Delta t$) immer noch einen Korrelationskoeffizient von besser als 0.979 besitzt.

In Abbildungen 3.11 ist das Trainingsergebnis von 200 nach dem „Neuronengas"-Algorithmus trainierten Neuronen für eine Zeitserie mit $\tau = 17$ dargestellt. Der

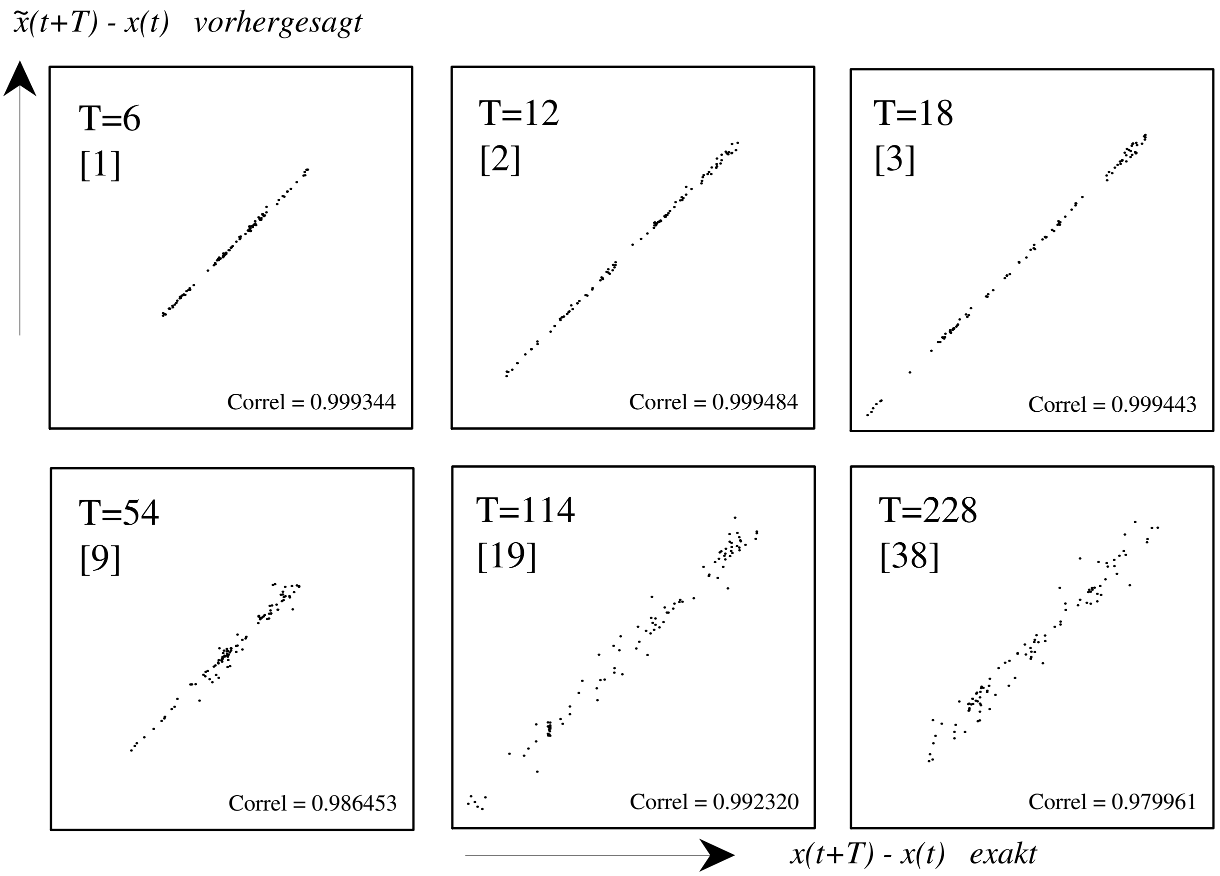


Abbildung 3.10: Korrelationsdarstellung zwischen je 100 vorhergesagten und gemessenen Datenpunkten für [1], [2], [3], [9], [19] und [38] Vorhersageiterationen mit $\Delta t = 6$ ($T = 6, 12, 18, 54, 114, 228$). Die Daten stammen von einem 10×10 Kohonennetz nach 10000 Trainingsschritten für die $\tau = 16$ Mackey–Glass Zeitserie. Die jeweiligen *Zahlen* recht unten, geben den aus der linearen Regression gewonnenen Korrelationskoeffizienten an.

fadenförmige Attraktor ist im Falle $\tau = 17$ zu einem verdrehten, bandartigen Gebilde der Dimension $d = 2.1$ [14] ausgeschmiert, das in feinerer Auflösung eine fraktale Struktur aufzeigt. Im Gegensatz zum Fall $\tau = 16$, haben es die Neuronen hier einfach, sich auf dem Attraktor zu verteilen.

In Abbildung 3.12ab sind diesem Verhalten des „Neuronengasmodells“ zwei Entwicklungszustände eines quadratisches Kohonengitter mit 50×50 Neuronen für dieselben Trainingsdaten gegenübergestellt. Im *linken* Bild sieht man, wie sich das Netz in kontinuierlicher Weise in das verwundene Attraktorgebiet nach 7500 Trainingsschritten hineingelegt hat. Dabei zieht die Wechselwirkung zwischen den Gitternachbarn viele Neuronen noch in Gebiete, wo sie eigentlich nutzlos sind, d. h. es gelingt den im Gitter starr miteinander gekoppelten Neuronen nicht, sich schnell voneinander zu lösen und ihre Referenzvektoren $\mathbf{w_r}$ auf das Attraktorgebiet zu fokussieren. Eine andere Gittertopologie würde hier Besserung schaffen, z. B. wäre eine Ringstruktur

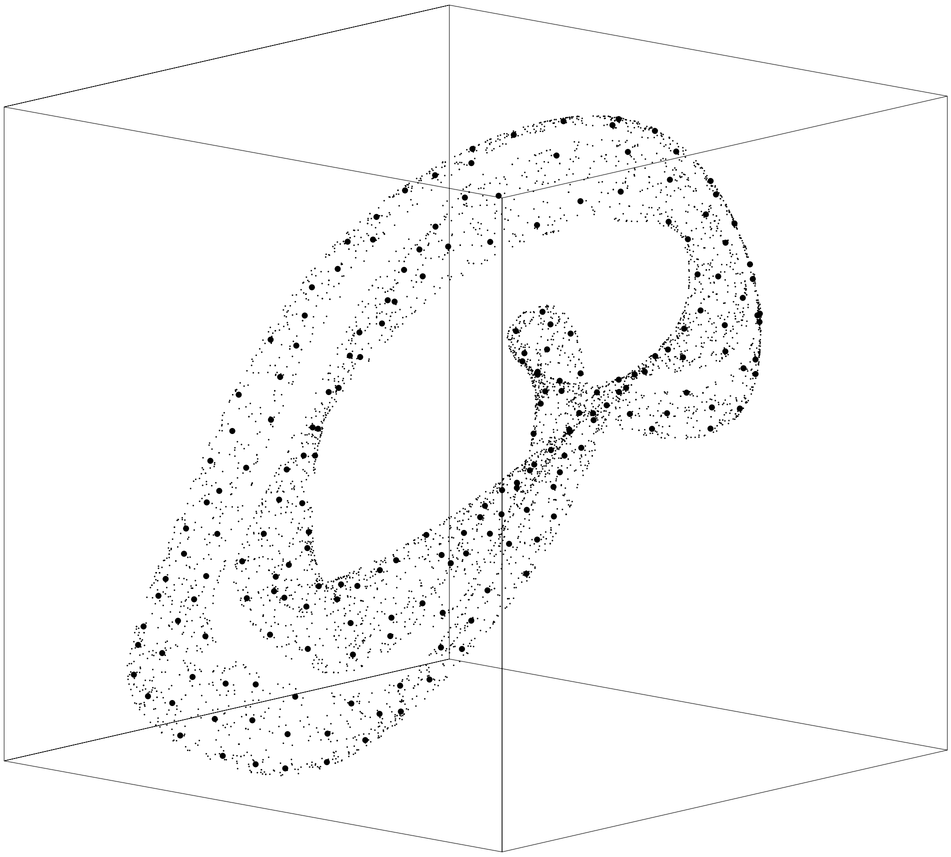


Abbildung 3.11: Das „Neuronengas“ Netzwerk nach dem Training mit Mackey–-Glass Daten ($\tau = 17$). Dem Attraktor (*feine Punkte*) kann hier die fraktale Dimension $d = 2.1$ zugewiesen werden. Die (*dickeren Punkte*) repräsentieren die Lage der 200 Referenzvektoren $\mathbf{w}_\mu$ im Eingangsraum als Orthonalprojektionen der 2., 3. und 4. Dimension. ($\mathcal{K}_{1,2,3,} = 1$)

naheliegend. Doch genau für diese Entscheidung benötigt man das Vorwissen um die Topologie der Eingangsraummannigfaltigkeit, ein Vorwissen, das man nicht voraussetzten möchte.

Wenn die Nachbarchaftskopplung klein genug wird, können sich die Neuronen, die zunächst in der (leeren) Mitte der Attraktorschleifen verweilten, gänzlich einer Seite zuwenden, im Limes $\lambda, \sigma \to 0$ werden beide Verfahren, der Kohonen-Algorithmus und der „Neuronengas“-Algorithmus gleichwertig zu einem konventionellen Vektorquantisierungsverfahren (siehe 2.2.1). Dies wird in Abbildung 3.12 (*rechtes* Bild) demonstriert: man sieht, daß das Kohonennetzwerk sich dem Attraktorgebiet im Verlauf einer Lernphase von 75000 Schritten gut angepasst hat und die durch feine Punkte markierte Attraktormenge abdecken konnte.

An dieser Stelle wird nochmal ein Vorteil des „Neuronengas“-Modelles gegenüber dem Kohonen-Algorithmus deutlich. (i) man benötigt keinerlei a-priori Kenntnis über die Struktur der Eingangsraumsmannigfaltigkeit; (ii) der „Neuronengas“-Algorithmus besitzt ein hohes Maß an Adaptationsfähigkeit an verschiedendste Struktu-

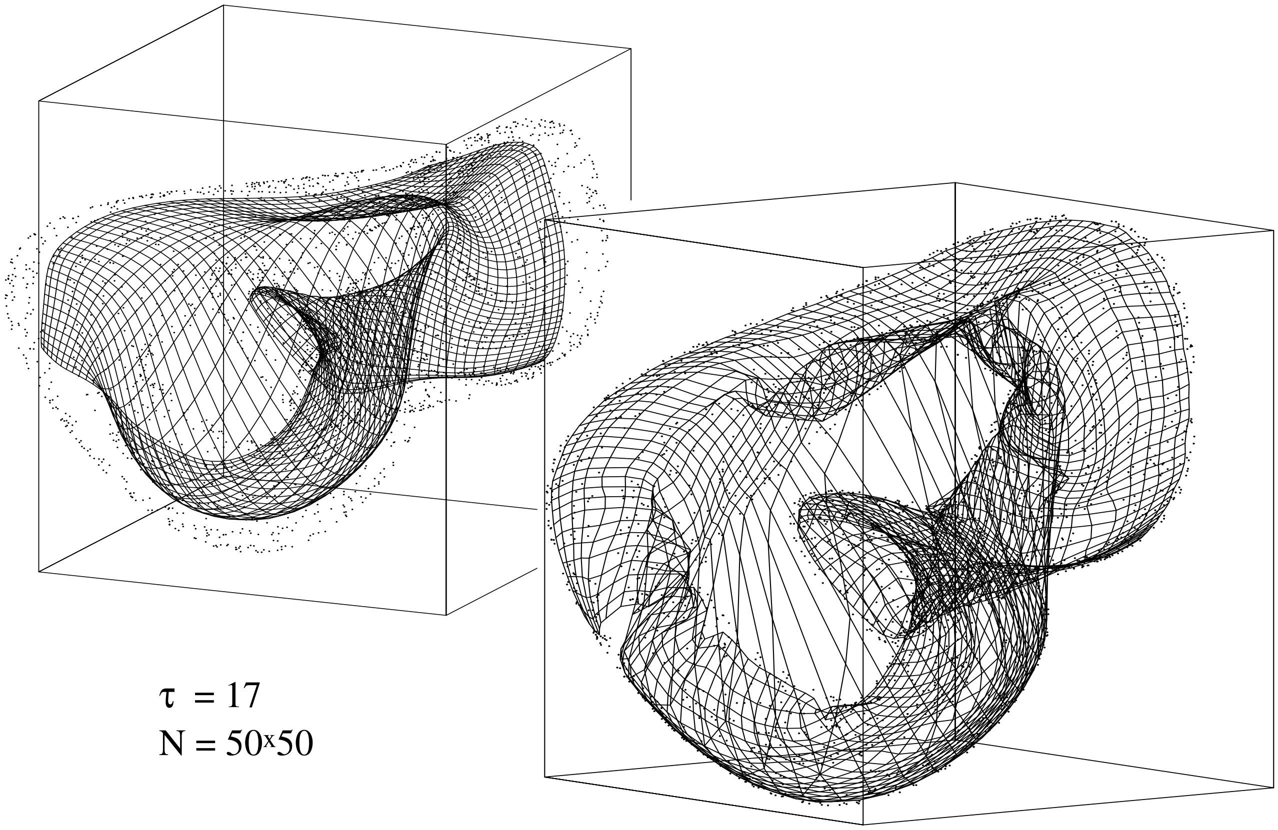


Abbildung 3.12: Kohonennetzwerk mit 2500 Neuronen nach (*links*) 7500 Trainingsschritten und nach (*rechts*) 75000 Trainingsschritten mit Mackey–Glass Daten ($\tau = 17$). Die *Gitterknoten* repräsentieren die Lage der 2500 Referenzvektoren $\mathbf{w_r}$ im Eingangsraum als Orthogonalprojektionen der 2., 3. und 4. Komponente.

ren, wie am obigen Beispiel und auch im Bild 2.6 demonstriert wurde. Der Hauptnachteil des „Neuronengas"-Algorithmus ist sein mit der Neuronenzahl stark anwachsender Sortieraufwand, der allerdings durch geschicktes Wechseln der Sortierverfahren begrenzt gehalten werden kann. In späteren Lernphasen werden typischerweise die Skalenparameter für die Nachbarschaftsregion drastisch verkleinert und selbst deren größter Skalenparameter (s. Seite 109) $\lambda^{max} = \max(\lambda, \lambda', \lambda^{mix})$, wird nur noch höchstens einige wenige Nachbarneuronen am Lernschritt effektiv mitbeteiligen. Beschränkt man das Sortieren auf etwa $5 \cdot \lambda^{max}$ Neuronen, die am nächsten zu $\mathbf{x}_{stim}$ liegen, ergibt sich eine beträchtliche Beschleunigung des Lernverfahrens.

In Abbildung 3.13 wird das Skalierungsverhalten der Vorhersagegüte der beiden oben beschriebenen Algorithmen mit Ergebnissen aus der neuronalen Netzwerkliteratur verglichen. Aufgetragen ist wieder der normalisierte Vorhersagefehler F_{norm} für prognostizierte Zeitpunkte $t + 90$ Zeitschritte gegen die Anzahl von verwendeten Neuronen.

Die Kurve **KN** zeigt die Skalierungskurven der für die nach dem erweiterten Kohonenmodell trainierten $N = L \times L$ Neuronen nach jeweils $30N$ Lernschritten.

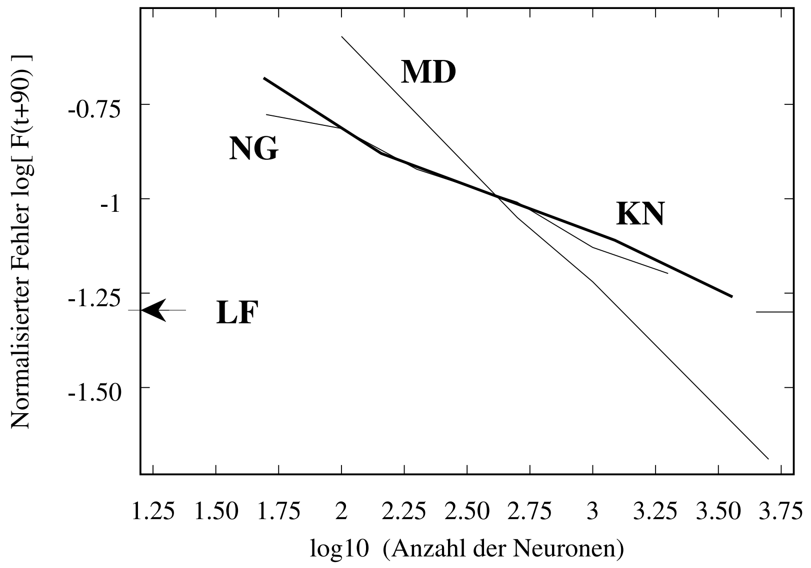


Abbildung 3.13: Skalierungsverhalten der Vorhersagegüte der beiden oben beschriebenen Algorithmen (**KN** Kohonennetz; **NG** „Neuronengas") für eine Mackey–Glass Zeitserie mit $\tau = 17$ zusammen mit Ergebnissen anderer Untersuchungen (**LF** Lapedes und Farber; **MD** Moody und Darken). Im doppellogarithmischen (Basis 10) Graph ist der normalisierte Vorhersagefehler $F_{norm}(T = 90)$ gegen die Anzahl der verwendeten Neuronen aufgetragen.

Die Lernparameter wurden zwischen folgenden Parameterwerten linear interpoliert (die genaue Wahl ist nicht sehr kritisch, l ist wieder die Anzahl der Lernschritte): $\{(l_j, \varepsilon_j)\} = \{(1, 0.8), (3N, 0.05), (30N, 0)\}$ und $\{(l_j, \sigma_j)\} = \{(1, 0.7L), (3N, 0.05L), (30N, 0L)\}$; $\{(l_j, \varepsilon'_j)\} = \{(1, 0.7), (3N, 0.3), (30N, 0.3)\}$ und $\{(l_j, \sigma'_j)\} = \{(1, 0.7L), (3N, 0.1L), (30N, 0.02L)\}$;

Die Kurve **NG** stellt vorläufige Simulationsergebnisse des erweiterten „Neuronengas"-Algorithmus dar. Die Lernparameter für die N Neuronen wurden zwischen folgenden Stützstellen linear interpoliert gewählt: $\{(l_j, \varepsilon_j)\} = \{(1, 0.4), (20000, 0.0001), (40000, 0001)\}$, $\{(l_j, \varepsilon'_j)\} = \{(1, 0.9), (6000, 0.9), (10000, 0.23), (40000, 0.1)\}$ und $\{(l_j, \lambda'_j)\} = \{(1, 0.17N), (10000, 0.045N), (20000N, 0.025N), (40000, 0.0005N)\}$. Zwischen den $\{(l_j, \lambda_j)\}$ Stützstellen $\{(l_j, \lambda_j)\} = \{(1, 0.23N), (2000, 0.01N), (7500, 0.001), (40000, 0.001)\}$ wurde exponentiell interpoliert. Die Vorhersagefehlerwerte werden nach 40000 Lernschritten anhand eines separaten Testdatensatzes bestimmt, jedoch konvergiert der Fehler des „Neuronengas"-Netzwerks meist schon nach ein paar Tausend Lernschritten. Beim Vergleich der Ergebnisse beider verwendeten Algorithmen **KN** und **NG** muß man feststellen, daß der Unterschied zwischen den beiden Netzwerktypen weitgehend geschrumpft ist. Möglicherweise ist dies auf die annäherd zwei-dimensionale Attraktorstruktur zurückführen, mit der das zwei-dimensionale Kohonenetz nach genügendem Training zurechtkommt. Aus Zeitgründen konnten jedoch keine Experimente ausgeführt werden, die die beiden Algorithmus ausführlich bezüglich ihres wirklichen

Rechenaufwandes vergleichen. Der erkennbare Trend ist, daß das „Neuronengas" signifikant schneller auf komplizierte Topologien reagieren kann, da es sich schon bei viel größeren λ-Werten, d. h. schon zu Zeiten noch besonders schnellem, kollektiven Lernens der Neuronen, auf sein zuständiges Gebiet festlegen kann, ohne daß die Neuronen ggf. von Gitternachbarn in deren abgelegenen Zuständigkeitsbereiche gezogen werden, wie dies beim Kohonen-Algorithmus vorkommen kann.

Die Kurve **MD** stellt das Ergebnis von Moody und Darken [40] für das selbige Vorhersageproblem dar. (Die Autoren sagen eine geringfügig kürzere Zeitspanne, nämlich $t + 85$ voraus). Moody und Darken verwenden N radiale Basisfunktionen ϕ_i in der Form (3.12) wie sie auf Seite 29 beschrieben wurden. Deren Zentren $\mathbf{w}_i$ werden *vor* der eigendlichen Trainingsphase durch ein sog. *k-means clustering* Verfahren bestimmt, und die c_i nach einer Heuristik festgelegt. Die a_i Werte werden ähnlich wie bei unserem Verfahren durch ein Gradientenabstiegsverfahren mit $10N$ Trainingsdaten adaptiert.

Der mit **LF** markierte Datenwert ist ein Ergebnis von Lapedes und Farber [30], der mit einem nichtlinearen *feed-forward* Netzwerk vom Typ (3.13) erzielt wurde. Die Autoren verwendeten zwei *hidden layers* mit insgesamt 541 Gewichten, die mit dem *Backpropagation* Algorithmus adaptiert wurden. Ebenso wie unsere Methode berechneten auch sie den Trajektorienverlauf durch rekursive Anwendung des Vorhersageschemas, anstatt nur einen Datenpunkt $\tilde{x}(t + T)$ in einem *Ein*schritt Verfahren, wie bei Moody und Darken, vorherzusagen.

Lapedes und Farber finden im Experiment sogar eine Verbesserung der Vorhersagegenauigkeit durch iterative Vorhersage des gewünschten Zunkunftswertes, d. h. $\mathbf{x}(t) \rightarrow \tilde{\mathbf{x}}(t+\Delta t) \rightarrow \tilde{\mathbf{x}}(t+2\Delta t) \rightarrow \cdots \rightarrow x(t+T)$, im Vergleich zu direkter Vorhersage, d. h. $\tilde{\mathbf{x}}(t) \rightarrow \tilde{x}(t+T)$. Dies steht in Einklang mit den Untersuchungen von Farmer und Sidorowich, die eine theoretische Analyse zur Fehlerfortpflanzung für diese beide Fälle betrachten [15]. Im Anschluß berichten Farmer und Sidorowich jedoch von einem experimentellen Gegenbeispiel, daß von einem Verallgemeinerungsversuch abrät. Auch sie erzielen sehr gute Resultate mit ihrem, auf Seite 30 beschriebenen, lokal interpolierenden Approximationsverfahren, das jedoch im eigentlichen Sinne nicht einen adaptiven, neuronalen Netzwerkalgorithmus darstellt.

Tabelle 3.1 faßt die Hauptcharakteristika der verschiedenen neuronalen Netzwerkprädiktionsmethoden zusammen.
Moody und Darkens Algorithmus eignet sich offensichtlich besser für sehr große Netzwerke, was möglicherweise auf die günstigeren Approximationseigenschaften der Summe über einer großen Anzahl von Basisfunktionen zurückzuführen ist. Die Ergebnisse von Lapedes und Farber erreichen mit nur 541 Parametern ein sehr gutes Ergebnis, allerdings ist dieses Verfahren mit einem sehr großen Rechenaufwand ver-

	Netzwerktyp	Prädiktionsmethode	Vorhersage	Echtzeitfähig
NG	erw. „Neuronengas“	lokale lin. Approx.	ganze Trajektorie	Ja
KN	erw. Kohonennetz	lokale lin. Approx.	ganze Trajektorie	Ja
MD	*k-means clustering*	radiale Basisfunktionen	ein Datenpunkt	Ja
LF	nichtlin. *feed-forward* Netz	Summenbildung	ganze Trajektorie	Nein

Tabelle 3.1: Zusammenfassung der Charakteristika verschiedener „neuronaler“ Prädiktionsmethoden.

bunden. Die Autoren benötigen knapp eine Stunde CRAY-XMP – Rechenzeit, um mit einem hochoptimierten *Backpropagation*-Algorithmus (konjugiertes Gradientenverfahren) diese Parameter zu ermitteln, was dieses Vorhersageschema für die Verarbeitung von Echtzeitdaten (*on-line*) ungeeignet erscheinen läßt.

Kapitel 4

Visuo-motorische Koordination

Der zweite Hauptteil dieser Arbeit beschäftigt sich mit einem Thema aus der Robotik. Es wird gezeigt, wie ein System aus einem Roboterarm und zwei Kameras durch bloßes Üben das Erreichen eines im Arbeitsbereich beliebig vorgegebenen Punktes mit seinem Endeffektor[1] erlernen kann. Der Zielpunkt soll dabei nicht durch Angabe von präzisen Raumkoordinaten, sondern durch eigenes „Sehen" im tatsächlichen Arbeitsbereich vorgegeben werden. Zwei Kameras sind auf den Arbeitsbereich des Roboters gerichtet und dienen dieser visuellen Zielerkennung und zusätzlich der visuellen Rückkopplung. Dies genügt zum Lernen. Es wird weder ein externer Lehrer noch irgendein a-priori Wissen über die Geometrie des Armes oder der Kameras verwendet.

Die Fähigkeit der Endeffektorpositionierung ist eine der Grundvoraussetzungen, auf der fast alle denkbaren Aufgabenstellungen eines Roboters aufbauen. Dazu gehören alle Arten von Objektmanipulationen, z. B. Bauteilbestückung oder Schweißen, oder auch Trajektorienplanung zur Umgehung von Hindernissen – ein sehr schwieriges und bislang nur unbefriedigend gelöstes Problem.

Die vorliegende Arbeit beschränkt sich auf das Erlernen der *inversen Kinematik* für den nicht-redundanten Fall mit drei Freiheitsgraden, d. h. der Transformation der Raumkoordinaten, bzw. im vorliegenden Falle, Bildkoordinaten, in einen Satz von drei Gelenkwinkeln. Die Transformation ist das Gegenstück zur *Vorwärtskinematik*, welche die Transformation von Gelenkwinkel in Raumkoordinaten beschreibt und von der physischen Existenz des Arm selbst dargestellt wird [3, 6, 44]. Die Kinematik muß sowohl die Geometrie des Robotarmes, dessen relative Position zu den Kameras, die auf ihn gerichtet sind, als auch die optischen Abbildungseigenschaften berücksichtigen. Sie berücksichtigt keine Trägheitseffekte der massebehafteten Armsegmente,

[1] Als Endeffektor bezeichet man in der Robotik ein am Ende eines Armes befestigtes Werkzeug z. B. ein Greifer, Schweißzange oder ein Lacksprühkopf.

die bei schnellen Bewegungen eine zunehmende Rolle spielen. Diese Betrachtung ist ausreichend für genügend langsame Bewegungen, oder wenn die Gelenkantriebe genügend stark sind, um ihre Bewegung den Gelenken aufzuzwingen.

In diesem Kapitel soll der für das Erlernen dieser Aufgabe verwendete neuronale Algorithmus präsentiert werden, gefolgt von zwei Kapitel über die durchgeführte Implementation auf einem PUMA 562, ein Roboterarm, der in der Industrie häufig Verwendung findet. In Kapitel 7 werden die experimentellen Ergebnisse berichtet und diskutiert.

4.1 Der „Neuronengas"-Algorithmus für die visuo-motorische Koordination

Der für die visuo-motorische Koordination eines Roboter-Kamera-Systems verwendete Algorithmus beruht auf einer Fortentwicklung früherer Arbeiten von (Martinetz, Ritter und Schulten 1988, 1990 [39, 35, 48]), die den Kohonen-Algorithmus erweiterten (siehe auch [48]). Die hier verwendete Version [56] macht sich das „Neuronengas"-Netzwerk zunutze, um die nötige nichtlineare Abbildung der Zielposition im Kamerabild in geeignete Motorkommandos zu lernen (siehe auch Abschnitt 2.4).

Das Roboter-Kamera-System ist im Blockschema 5.5 (Seite 67) illustriert. In jedem Trainingsschritt wird ein Zielpunkt aus dem Arbeitsbereich zufällig ausgewählt und dem Robotersystem gezeigt. Zwei Videokameras betrachten die Szene und ein Bildverarbeitungssystem extrahiert für jede der Kameras ein Koordinatenpaar (x, y), das den Ort des Zielpunktes auf den (zwei-dimensionalen) Kamerabildern, den „Kameraretinas", bezeichnet. Die beiden Kamerakoordinatenpaare werden zu einem vier-dimensionalen Vektor $\mathbf{u}_{target} = (x_{links}, y_{links}, x_{rechts}, y_{rechts})^T$ gruppiert, der als Eingangssignalvektor für das Netzwerk dient[2].

Um seinen Endeffektor korrekt positionieren zu können, muß das Robotersystem die Transformation von $\mathbf{u}_{target}$ in den korrespondierenden Satz von Motorkommandos für die drei Gelenke $\vec{\theta}(\mathbf{u}_{target})$ kennen. Wie schon erwähnt, hängt diese Transformation von den Antriebscharakteristika und der Kinematik (Geometrie) des Robotarms sowie von der Positionierung der Kameras und deren optischen Eigenschaften ab.

Die fundamentale Idee hinter unserem Ansatz ist wieder ähnlich dem letzen Beispiel: Der Eingangsraum der Zielvektoren $\mathbf{u}_{target}$ wird in N disjunkte Zellen mit dem

[2] $\mathbf{u}_{target}$ entspricht der Rolle von $\mathbf{x}$ in Kapitel 2; für ein Verzeichnis häufig verwendeter Symbole siehe Seite 109.

Index μ und den Zellzentren $\mathbf{w}_\mu$ diskretisiert, und jeder Zelle wird ein Neuron zugewiesen. Jedes Neuron hat zwei Ausgabewerte: einen Vektor (ein Satz von Gelenkwinkeln) $\vec{\theta}_\mu = \vec{\theta}(\mathbf{w}_\mu)$ und eine 3×4-Matrix $\mathbf{A}_\mu = \mathbf{A}(\mathbf{w}_\mu)$, die sogenannte Jakobimatrix, die zusammen eine lineare Taylorentwicklung

$$\vec{\theta}(\mathbf{u}_{target}) = \vec{\theta}(\mathbf{w}_\mu) + \mathbf{A}(\mathbf{w}_\mu) \cdot (\, \mathbf{u}_{target} - \mathbf{w}_\mu \,)\,, \tag{4.1}$$

um den Disketisierungspunkt $\mathbf{w}_i$ konstituieren.

Sowohl die Wahl der Diskretisierungszellen als auch das Adaptieren der Ausgabewerte folgt dem Lernschema, das in Kapitel 2 vorgestellt wurde. Der Teil der für die Adaptation der Ausgabewerte zuständig ist, ändert sich, da nun zwei von einander abhängige Größen gelernt werden.

Das Netzwerk operiert als ein "*winner takes most network*", d.h. die Ausgabewerte werden über alle Neuronen gemittelt. Gleichung (2.22) ändert sich zu

$$\vec{\theta}_{initial} = \mathcal{S}^{-1} \cdot \sum_k g^{mix}(k) \left(\vec{\theta}_{\mu_k} + \mathbf{A}_{\mu_k} \cdot (\mathbf{u}_{target} - \mathbf{w}_{\mu_k}) \right). \tag{4.2}$$

Die Positionierung des Roboterarmes erfolgt in zwei Phasen. Eine erste Phase besteht aus einer „Grobbewegung“ zur Gelenkwinkelkonfiguration $\vec{\theta}_{initial}$, gegeben durch Gleichung (4.2). Die resultierende Endeffektorposition wird von den Kameras beurteilt und soll mit $\mathbf{v}_{initial}$ bezeichnet werden ($\mathbf{v}_{initial}$ ist, wie $\mathbf{u}_{target}$, ein vierdimensionaler Bildkoordinatenvektor). In einer zweiten Phase wird die verbliebene Abweichung von der gewüschten Position ($\mathbf{u}_{target} - \mathbf{v}_{initial}$) mit Hilfe der Jakobimatrix $\mathbf{A}(\mathbf{u}_{target})$ in eine Korrekturbewegung übersetzt. Um $\mathbf{A}(\mathbf{u}_{target})$ zu bestimmen, wird über alle Jakobimatrizen $\mathbf{A}_\mu$ gemittelt, wiederum mit der Funktion g^{mix} gewichtet. Wie in (4.2) wird somit der Beitrag des (*winner-*) Neurons μ_0 am größten, dessen $\mathbf{w}_{\mu_0}$ am nächsten zu $\mathbf{u}_{target}$ liegt; das Neuron μ_1 mit $\mathbf{w}_{\mu_1}$ am zweitnächsten zu $\mathbf{u}_{target}$ trägt am zweitmeisten bei, und so weiter. Dies führt zur Gelenkwinkelkorrektur

$$\Delta\vec{\theta} = s^{-1} \cdot \sum_k g^{mix}(k) \mathbf{A}_{\mu_k} \cdot (\mathbf{u}_{target} - \mathbf{v}_{initial}), \tag{4.3}$$

die den Endeffektor schließlich in die Position führt, die die Kameras als $\mathbf{v}_{final}$ sehen. Prinzipiell läßt sich diese Rückkopplungsschleife beliebig oft wiederholen, jedoch zeigen die experimentellen Resultate, daß in der Regel ein Korrekturschritt genügt. Während der Lernphase ist auf alle Fälle mindestens ein solcher Korrekturschritt erforderlich, da er durch die spezielle visuelle Erfolgskontrolle die Grundlage der Lernerfahrung schafft.

Während der Lernphase müssen geeignete Werte für $\mathbf{w}_\mu$, $\vec{\theta}_\mu$ und $\mathbf{A}_\mu$ gefunden werden. Dies erfolgt adaptiv durch folgende Lernregeln, die nach jedem Versuch ausgeführt werden (vgl. Gleichungen (2.23-2.24), Seite 18):

$$\mathbf{w}_{\mu_k}^{new} = \mathbf{w}_{\mu_k}^{old} + \varepsilon \cdot g(k) \cdot (\mathbf{u}_{target} - \mathbf{w}_{\mu_k}^{old}) \tag{4.4}$$

$$\vec{\theta}^{new}_{\mu_k} = \vec{\theta}^{old}_{\mu_k} + \varepsilon' \cdot g'(k) \cdot \Delta\vec{\theta}_{\mu_k} \tag{4.5}$$

$$\mathbf{A}^{new}_{\mu_k} = \mathbf{A}^{old}_{\mu_k} + \varepsilon' \cdot g'(k) \cdot \Delta\mathbf{A}_{\mu_k}. \tag{4.6}$$

Gleichung (4.4) schafft automatisch eine homogene Diskretisierung der relevanten *drei-dimensionalen* Untermanigfaltigkeit im vier-dimensionalen Eingaberaum (siehe auch [36, 37]). Ein kurzes Rechenbeispiel zeigt, wie wichtig diese Fähigkeit ist. Angenommen eine homogene Diskretisierung des gesamten vier-dimensionalen Eingaberaum mit i Teilungen pro Dimension müßte a-priori festgelegt werden. Nimmt man noch zwei Reserveteilungen für Fehljustierung und Drifteffekte hinzu, so benötigt man $(i+2)^4$ statt i^3 Neuronen — z. B. für eine grobe Teilung mit $i = 7$ ergibt sich ein Bedarf von 6561 statt, wie im Falle einer dynamischen Diskretisierung, von nur 343 Neuronen.

$\Delta\mathbf{A}_\mu$ ergibt sich aus der Forderung, daß die Jakobimatrix $\mathbf{A}_\mu$ die Korrespondenz zwischen der Gelenkwinkelkorrektur $\Delta\vec{\theta} = \vec{\theta}_{final} - \vec{\theta}_{initial}$ und dem visuellen Resultat $\Delta\mathbf{v} = \mathbf{v}_{final} - \mathbf{v}_{initial}$ des gegenwärtigen Korrekturschrittes korrekt beschreiben soll. In anderen Worten $\Delta\mathbf{A}_\mu$ muß die quadratische Kostenfunktion

$$E[\mathbf{A}_\mu] = \frac{1}{2}\left(\Delta\vec{\theta} - \mathbf{A}_\mu \Delta\mathbf{v}\right)^2 \tag{4.7}$$

minimieren, woraus sich die Fehlerkorrekturregel

$$\Delta\mathbf{A}_\mu = \left(\Delta\vec{\theta} - \mathbf{A}_\mu \Delta\mathbf{v}\right) \frac{\Delta\mathbf{v}^T}{\|\Delta\mathbf{v}\|^2} \tag{4.8}$$

vom Widrow-Hoff Typus ergibt (siehe Seite 13, [60]). Die Nachbarschaftsgewichtungsfunktion $g'(k)$ in (4.6) unterdrückt dabei das Adaptieren von Neuronen, die nicht „zuständig“ sind, und somit bei der Minimierung der Kostenfunktion (4.7) für diesen Versuch unwesentlich beitragen (s.a. 2.24, 2.25).

Mit der Annahme, daß $\mathbf{A}_\mu$ bereits eine gute Näherung der wahren Jakobimatrix $\mathbf{A}$ darstellt, kann man die lokal gültige Beziehung

$$\vec{\theta}_{initial} - \vec{\theta}_\mu^{\,*} = \mathbf{A}_\mu \cdot (\mathbf{v}_{initial} - \mathbf{w}_\mu) \tag{4.9}$$

mit $\vec{\theta}_\mu^{\,*}$ als einem neuen Schätzwert für $\vec{\theta}_\mu$ aufstellen. Dies ergibt eine Korrektur in Richtung

$$\Delta\vec{\theta}_\mu \;=\; \vec{\theta}_\mu^{\,*} - \vec{\theta}_\mu^{\,old} \;=\; \vec{\theta}_{initial} - \vec{\theta}_\mu^{\,old} - \mathbf{A}_\mu \cdot (\mathbf{v}_{initial} - \mathbf{w}_\mu), \tag{4.10}$$

was auch dann schon ein Verbesserung von $\vec{\theta}_\mu$ ergibt, wenn $\mathbf{A}_\mu$ die korrekte lokale Abbildung erst grob annähert. Durch die nichtlineare Kopplung der beiden Gradientenabstiegsverfahren — $\Delta\mathbf{A}_\mu$ ist von $\vec{\theta}_\mu$, und $\Delta\vec{\theta}_\mu$ von $\mathbf{A}_\mu$ abhängig — ist die Konvergenz von $\vec{\theta}_\mu$ und $\mathbf{A}_\mu$ für das einzelne Neuron nicht mehr garantiert. Erst das kollektive Zusammenwirken benachbarter Neuronen gewährleistet hier die Stabilität des Algorithmus (siehe Kapitel 2 und [48]). In Abschnitt 7.4 werden zwei Experimente vorgestellt, die den Beitrag dieses kollektiven Zusammenwirken explizit testen, und im Resultat noch einmal die Wichtigkeit der Nachbarschaftskooperation unterstreichen.

Wie man an den Korrekturschritten (4.8) und (4.10) sieht, benötigt das Robotersystem ausschließlich Information, die von den Kameras geliefert wird, alle anderen Größen sind selbst generiert. Somit kann es das Lernziel ohne jede Art von externen „Lehrer“ erreichen. Dies soll die Erläuterung zum Algorithmus beschließen.

Kapitel 5

Die Ausstattung des Roboterlabors

5.1 Überblick

In diesem Kapitel soll das Konzept und die Ausstattung des Roboterlabors beschrieben werden. Es richtet sich an den technisch interessierten Leser[1] und beschreibt die *Hardware*-Aspekte dieser Diplomarbeit.

Das Roboterlabor ermöglicht, die Funktionstüchtigkeit und Anwendbarkeit der neuronalen Algorithmen in der Praxis zu demonstrieren und Erfahrungen für weiterführende Experimente zu gewinnen. Planung, Beschaffung und Aufbau des Labors waren ein wesentlicher Bestandteil dieser Diplomarbeit. Der Aufbau und die Arbeitsweise von Systemkomponenten sollen detaillierte Beschreibung finden, ferner sollen auch Überlegungen, Schwierigkeiten und Lösungen erläutert werden, welche die Planung und den Aufbau des Labors begleiteten.

Die Hauptkomponenten des Roboterlabors sind in Abbildung 5.1 dargestellt. Es handelt sich dabei um (*i*) den Roboter mit seiner Steuerung, (*ii*) einen Rechner auf dem der neuronale Netzwerk-Algorithmus ablaufen kann und (*iii*) ein Bildvorverarbeitungssystem, das mit den Videosignalen zweier Kameras gespeist wird. Die Abschnitte 5.2, 5.3 und 5.5 werden sich mit diesen drei Komponenten auseinandersetzen. Bei der Verwendung eines handelsüblichen Roboters in Verbindung mit einem Computer entsteht ein Kompatibiliätsproblem. Abschnitt 5.4 beschreibt den gefundenen Lösungsweg.

[1] Dieses Kapitel ist nicht Voraussetzung für das Verständnis der folgenden Abschnitte.

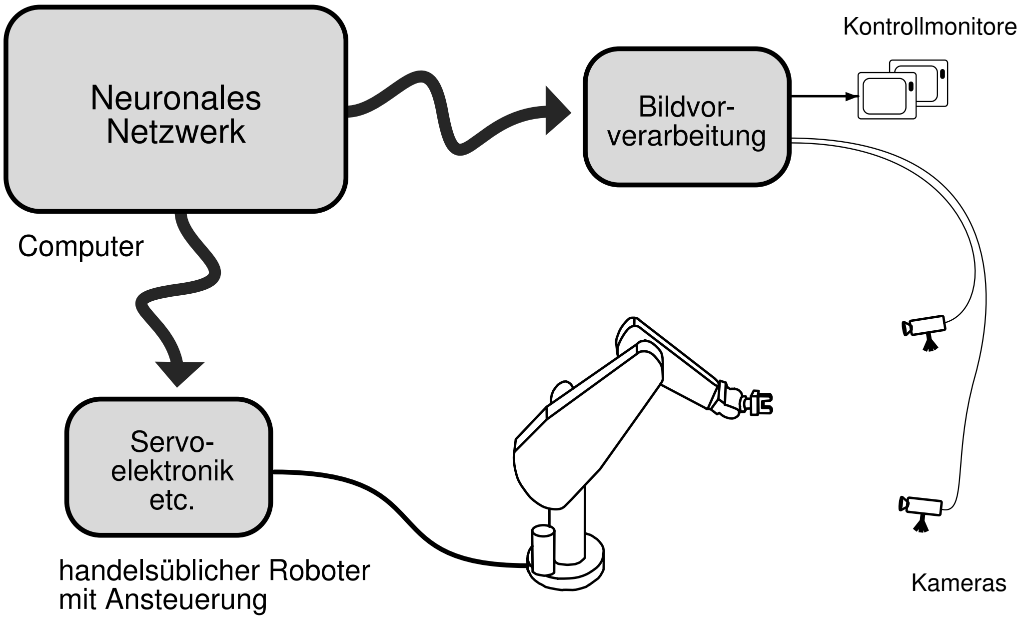


Abbildung 5.1: Die Hauptkomponenten des Roboterlabors

Die Anforderungen

Zur Planung des Roboterlabors waren die Anforderungen festzulegen, die die einzelnen Komponenten zu erfüllen haben ("Plichtenheft"). Sie sind hier stichpunktartig aufgelistet und werden in den folgenden Abschnitten besprochen.

1. Der Roboter soll in der Lage sein, sowohl nach stationären als auch nach bewegten Objekten zu greifen.

2. Zentralrechner, im folgenden auch *Hostcomputer* genannt, realisiert

   - C – Programmierumgebung
   - Kompatibilität zu anderen Systemkomponenten
   - ausreichende Leistungsfähigkeit für folgende Aufgaben
     - Kontrollalgorithmus in Echtzeit ablauffähig
     - Datenauswertung
     - Handhabung und Speicherung von Bilddaten
   - Integrationsfähigkeit in bestehendes Computernetzwerk

3. Der Roboterarm besitzt

   - mindestens fünf Freiheitsgraden (Studium von Greifbewegungen)
   - hohe Geschwindigkeit

- Reichweite der Größenordnung 1 Meter (sollte keine besonderen Ansprüche an die Laborraumhöhe stellen)
- handelsübliche Ausstattung (komplette Eigenentwicklung nicht möglich)

4. Robotersteuerung
   - Kraftsteuerbarkeit
   - „Echtzeitfähigkeit", d. h. die momentane Bewegung ist aufgrund von sensorischen Information modifizierbar
5. Die Bildvorverarbeitung beinhaltet
   - Zwei hochauflösende Kameras
   - Leistungsfähigkeit für Bildanalyse von komplizierteren Szenen und Szenen mit bewegten Objekten
   - Kontrollmonitore
   - Erweiterbarkeit mit zusätzlichen Kameras bzw. Farbkameras
6. Die Gesamtkosten sollen gering gehalten werden (begrenztes Budget)
7. Die Anlage soll sich durch Flexibilität und Mehrzweckfähigkeit auszeichnen.
8. Der Aufbau des Labors soll im Zeitrahmen dieser Arbeit möglich sein.

5.2 Der Roboter

5.2.1 Roboteranatomie

In Bild 5.2 sind zwei gebräuchliche Roboterarmarchitekturen abgebildet. Links ist die sogenannte SCARA Architektur[2] dargestellt, die in der Industrie gerne für Montier- und Bestückungsaufgaben verwendet wird [12]. Auf der rechten Seite sieht man einen Roboterarm in Drehgelenksarchitektur. Um spätere Verwendung des Roboters (Anforderung 7) zum Zwecke des Studiums von Greifbewegungen, die vergleichbar zu biologischen Vorbildern sind, möchlich zu machen, erscheint es sinnvoll, einer Drehgelenksarchitektur den Vorzug zu geben.

[2]Selective Compliance Assembly Robot Arm

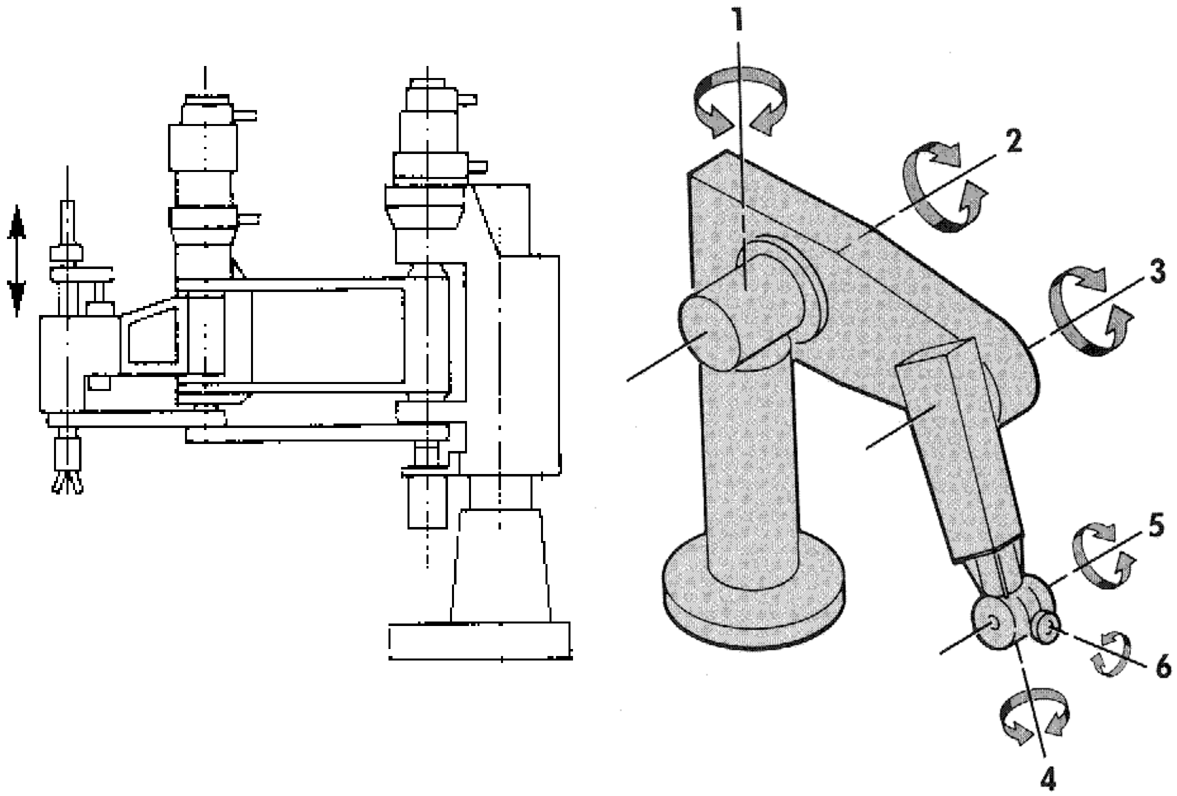


Abbildung 5.2: Zwei wichtige Roboterstrukturen: *(links:)* SCARA *(Selective Compliance Assembly Robot Arm)*, eignet sich besonders für planare Montagearbeiten und Positionieraufgaben auf Fließbändern. *(Rechts:)* Drehgelenksroboter (PUMA 562) mit sechs Freiheitsgraden.

5.2.2 Roboterantrieb

Als Roboterantrieb sind im wesentlichen — abgesehen von hydraulischen Motorantrieben in Schwerlast- und explosionsgefährdeten Bereichen — folgende zwei Typen gebräuchlich:

Motoren mit Getriebe (meist Schrittmotoren): Zeichnen sich durch besonders einfache Positionierbarkeit und gutes Leistungsgewicht aus. Das Getriebe übersetzt Schnellauf in hohes Drehmoment und gestattet, das Gewicht des Motors "körpernah" zu halten, was das Trägheitsmoment und damit die Dynamik des Arms günstig beeinflußt.

Direktantrieb (*direct drive*): hier sitzt der Motor direkt auf der Gelenkachse. Das an der Welle erzeugte Drehmoment ist bei dieser Bauweise in sehr guter Näherung proportional zum Motorstrom, was Kraftsteuerung erleichtert. Weitere Vorzüge liegen in großer Schnelligkeit, keinerlei Getriebeschlupf und geringem Wartungsbedarf — auf Kosten einer geringeren Leistungsdichte und schlechteren Dynamikeigenschaften.

Die Vorzüge der Kraftsteuerbarkeit (*force control*) werden beispielsweise beim Einsatz eines Roboters zum Fensterputzen deutlich: Eine präzise Positionierbarkeit ist hier von untergeordneter Bedeutung gegenüber der Anforderung, einen maximalen Anpressdruck nicht zu überschreiten.

5.2.3 Der Wunschroboter

Bei Studium des Marktes von Industrierobotern kamen zwei Modelle in die engere Wahl: Der eine ist der UNIMATE PUMA 562, ein klassischer, industriell bewährter Roboter, der sich durch hohe Zuverlässigkeit, Präzision und mittlere Schnelligkeit auszeichnet. Der andere ist der SOFT-ARM, ein interesssanter, außergewöhnlicher Robotertyp der von der japanischen Firma Bridgestone in kleiner Stückzahl gefertigt wird und bislang nicht regulär exportiert wurde.

Die Lieferschwierigkeiten des SOFT-ARM Roboters waren Anlaß, die leihweise Benutzung eines PUMAs zu arrangieren. Dies ermöglichte, in der Zeit bis zur Lieferung des SOFT-ARMs, wertvolle Hardwareerfahrung zu sammeln und ferner die in folgenden Abschnitten dargestellten Experimente durchzuführen.

5.2.3.1 Der SOFT-ARM

Die japanische Firma Bridgestone Inc. stellt den sogenannten SOFT-ARM her, einen pneumatisch angetriebenen Roboter mit fünf Freiheitsgraden. Jeder Gelenkantrieb besteht aus zwei parallelen Gummischläuchen, den sogenannten RUBBERTUATORen. Wie in Abbildungen 5.3a erkennbar, sind sie über eine Kette (oder Draht), die auf der Drehwelle läuft, verbunden, ganz ähnlich dem Agonisten und Antagonisten bei biologischen Gelenken. Beim Aufblasen des Schlauches wird die Blähung durch ein kreuzweise um den Schlauch gewobenes Geflecht in eine Längskontraktion übersetzt. Die Luftdrücke in den beiden auch als “künstliche Muskeln” bezeichneten Luftschläuchen bestimmen dann die Position (Differenzdruck) und die Steifigkeit (Summendruck) des Arms.

Die Vorteile dieser Konstruktion sind die leichte und kompakte Bauweise (Armgewicht von ca. 5 kg, bei einer Reichweite von 90 cm). Die Elastitzität des Armes bedeutet einen signifikanten Gewinn an inhärenter Sicherheit gegen Kollisionsschäden. Die Nachteile dieses Antriebtyps sind geringe Geschwindigkeit, geringe Maximalantriebskraft und schlechte Wiederholgenauigkeit.

Letzteren Nachteile auszugleichen scheint eine interessante Herausforderung für visuo-motorische Kontrollalgoritmen. Zudem eröffnet die strukturelle Ähnlichkeit

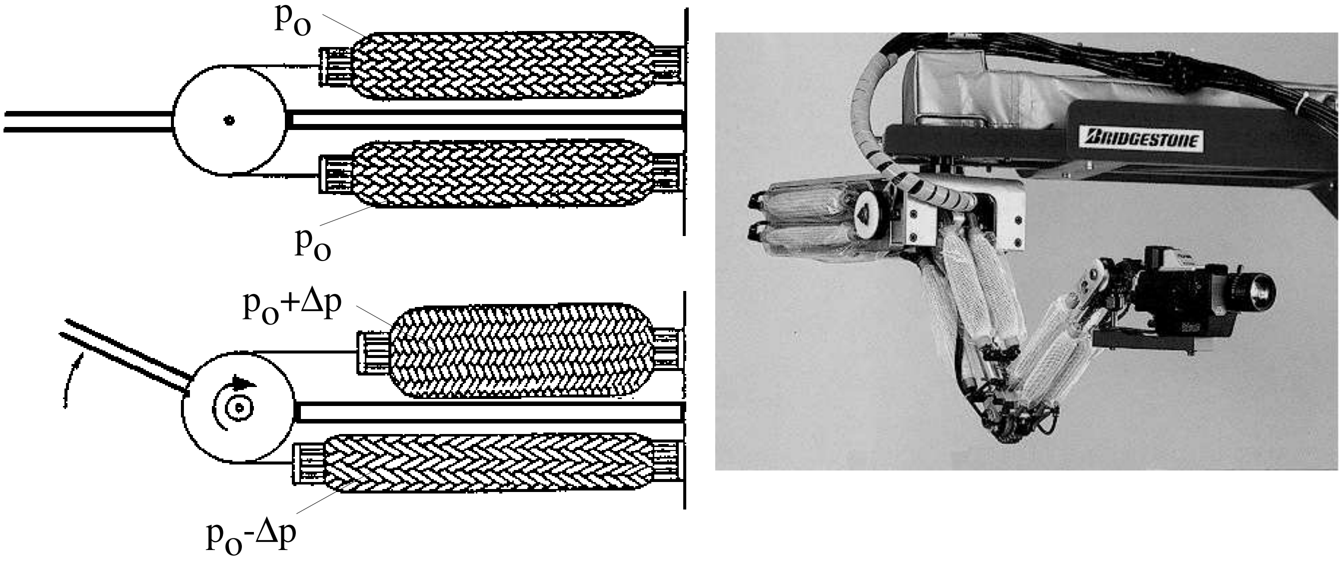


Abbildung 5.3: Funktionsprinzip und Anwendungsbeispiel des RUBBERTUATOR. *Links oben:* Mit zwei RUBBERTUATORen aufgebautes Drehgelenk in Ruhestellung. Beide „künstliche Muskeln" haben hier den selben Luftdruck. *Links unten:* Das Aufblasen des mit einem Kreuzgeflecht umspannten Gummischlauches führt zu Kontraktion (oberer Muskel). Entsprechende Druckverminderung führt zu Elongation (unterer Schlauch). Hiermit lassen sich Differenzdrücke in Drehmomente auf der Gelenkwelle umwandeln. Der mittlere Druck bestimmt die Gelenksteifigkeit. *Rechts:* Der SOFT-ARM-Manipulator mit fünf Freiheitgraden (Achsen 1,2,3,5 und 6, in Notation der vorhergehenden Abb.).

zum menschlichen Arm interessante Möglichkeiten zum Studium biologischer Kontrollstrategien.

5.2.4 Der Puma 562

Der in Abbildung 5.4 dargestellte Westinghouse-Unimation UNIMATE PUMA 562 (heute AEG-Westinghouse) ist ein klassischer Drehgelenksroboter mit sechs Freiheitsgraden; Tabelle 5.1 enthält einige Schlüsseldaten. Er wurde uns dankenswerterweise von Professor Ahuja (University of Illinois) zugänglich gemacht.

Zur Standardausrüstung des Roboters gehört ein sogenannter *Controller*, eine Konsole und ein Handeingabegerät mit einer Menütastatur und einem Not-Aus-Schalter *(Teachpendant)*.

Der WESTINGHOUSE UNIMATION CONTROLLER MARK III vereinigt in einem großen Chassis die Stromversorgung und die Steuerlogik [59]. Für jeden Antriebsmotor (Servo) ist ein seperater Servoelektronikeinschub und ein Mikrocomputereinschub

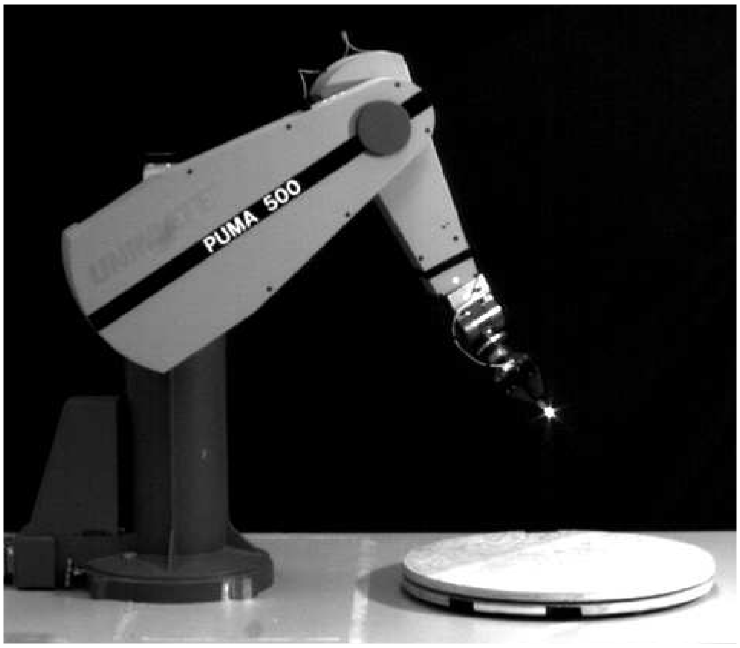


Abbildung 5.4: Ansicht des Westinghouse-Unimation Roboter UNIMATE PUMA 562. (Die Aufschrift bezieht sich auf die Serie)

Wiederholgenauigkeit	0.1 mm
maximale Geschwindigkeit (gerade Linie)	$0.5\frac{m}{s}$
maximale Last	4 kg
Reichweite (ohne Greifer)	920 mm
Armgewicht	63 kg

Tabelle 5.1: Einige wichtige technische Daten des verwendeten Roboters, des WESTINGHOUSE PUMA 562.

(*joint micros*) vorgesehen. Die sechs (Rockwell-6503) Mikroprozessoren arbeiten parallel und werden über einen separaten Datenbus von einem DEC LSI-11/73 Steuerprozessor mittels eines kleinen Satzes von Kommandoprotokollen kontrolliert. Ferner enthält der *Controller* 32 freie Eingabe- und Ausgabesignalleitungen, die über Optokoppler bzw. Relais galvanisch entkoppelt sind (binäre Signale).

Typische industrielle Anwendungen für einen vergleichbaren Roboter von heute sind, zum Beispiel, Schweißvorgänge, Manipulationen von Werkstücken und Lackierarbeiten. Die anzufahrenden Positionen werden dazu im Lernmodus programmiert, indem der Arm mittels Handeingabegerät in die gewünschten Lagen gebracht wird und diese als Solltrajektorienpunkte abgespeichert werden. Alternativ dazu werden im zunehmenden Maß auch CAD[3]–Daten der Werkzelle und des Objektes direkt für die Berechnung der Bewegungstrajektorien benutzt. Im Produktionsprozeß wird dann die Zielsequenz mit möglichst hoher Präzision und ggf. hoher Frequenz wiederholt. Das Steuerprogramm kontrolliert dabei die Einhaltung von bestimmten Systemzuständen in Form von Kontrollsignalen (z. B. „Werkstück vorhanden und richtig plaziert“ oder „Teil fertig für Abtransport“) und kann somit auch auf bestimmte Ereignisse durch Änderung des Programmablaufes reagieren (z. B. „Warten auf Teil“, Warnsignal geben oder Not-Aus).

In der Industrie erfolgt die Programmierung der Roboter in einer Spezialsprache, in unserem Falle VAL II, die heute in der Industrie als die „flexibelste“ Roboterprogrammiersprache gilt [58]. Dennoch ist Leistungsfähigkeit und Tranzparenz dieser Kombination des Steuerprozessors LSI-11 und der Programmiersprache VAL II, wie bereits erwähnt, für unsere Zwecke ungenügend. Da VAL II auch die Echtzeitkontrolle durch einen externen Rechner nicht unterstüzt, war eine Umprogrammierung des *Controllers* und eine direkte Steuerung des PUMAs durch den Zentralrechner die beste Lösung. Näheres zur gewählten Schnittstelle zwischen Roboter und Rechner soll im Kapitel 5.4 erläutert werden.

5.3 Der Hostcomputer

Als Hostcomputer wurde die *Workstation* SPARCSYSTEM 370-GX-8 der Firma SUN Mircosystems Inc. ausgewählt, die sich durch folgende Eigenschaften auszeichnet:

- VME Bus mit 12 freien (9U) Steckplätzen
- Schnelle RISC[4] Workstation, die mit dem auf 25 MHz getakteten SPARC Prozessor 16 MIPS[5] und gemeinsam mit dem Fließkommaprozessor 2.6 MFLOPS[6] leistet
- Großer Hauptspeicher von 32 Mbyte (mit *Parity Check*; die Standardkonfiguration wurde durch Austausch der 1 Mbyte SIMMs[7] durch steckerkompatible

[3] *Computer Aided Design;* Computergestützte Konstruktion schafft einfachen Zugang zu Geometrieinformation des Werkstückes in elekronisch weiterverarbeitbarer Form.

[4] *Reduced Instruction Set Computer. (Microcode* Interpretation ist „verdrahtet“).

[5] *Million Instructions Per Second.* (Bei Vergleich mit anderen Prozessorarchitekturen ist große Vorsicht geboten.)

[6] *Million Floating Point Operations Per Second.*

[7] *Single In-line Memory Module.* Speicher Steckmodule.

4 MByte Exemplare erweitert)

- Betriebssystem SUN OS 4.1[8]
- Netzwerkfähigkeit. Als *multiuser*, *multitasking*[9] Rechner fügt er sich vorteilhaft in das bestehende Computernetzwerk ein[10]
- Massenspeicher mit 688 Mbyte Platte (SMD) und 150 Mbyte Bandlaufwerk (SCSI 1/4" *cartridge*).

Damit stellt der Hostcomputer nicht nur eine hervorragende Programmierumgebung zur Verfügung, sondern erfüllt auch die Anforderung nach hoher Leistungsfähigkeit. Insbesondere gewährleistet er die vollständige Kompatibilität zu allen anderen Komponenten, sowohl auf der Software- als auch auf der Hardwareseite, siehe Abschnitte 5.4 und 5.5.

5.4 Die Roboterschnittstelle

Wie bereits erwähnt, unterscheidet sich die typische industrielle Anwendung des Roboters von heute (noch) stark von der unserigen. Heutige Robotersteuerungen sind konzipiert um programmierte Bewegungen selbstständig zu wiederholen. Wenig Wert ist auf flexible Programmierbarkeit oder Anschlußfähigkeit an ein Bildverarbeitungssystem mit hohen Datenströmen gelegt. Dies schafft ein Problem, das noch dadurch verschärft wird, daß auch die Echtzeitansteuerbarkeit durch einen anderen Rechner von Herstellerseite praktisch nicht unterstüzt wird. Die Lösung dieses Problems war die Substitution eines Teiles der *Robot-Controller* Software (VAL II) und Übernahme der Roboterkontrolle durch den Hostcomputer. Dies gelang in relativ kurzer Zeit Dank des Programmpaketes RCCL / RCI, über das in diesem Abschnitt ein Überblick gegeben werden soll.

Die Kürzel RCCL und RCI stehen für **R***obot*–**C***ontrol*–**C**–**L***ibrary* und **R***eal*–-*Time*–**C***ontrol*–**I***nterface*. RCCL / RCI ist ein Programmpaket, das es ermöglicht,

[8]SUN OS 4.1 gilt als mustergültige UNIX Implementation.

[9]*Multitasking:* Mehrere Programme laufen im Zeitmultiplexverfahren ab. *Multiuser:* Mehrere Benutzer können mit ihrer persönlichen „Umgebung“, an einem Rechner arbeiten und Dateizugriffsrechte können definiert werden.

[10]Fähigheit zu *remote procedure call (rcp), login (rlogin, rsh)*, wechselseitigem *ftp* und *nfs-mount* von Plattenspeicherdateisystemen; Dies ermöglicht es, von verschiedenen Rechnern eines Computernetzwerkes aus effektive Programmentwicklung und Auswertung durchzuführen (TCP-*Internet Protocol*).

C–Programme zur Robotersteuerung in Echtzeit unter dem Betriebssystem UNIX einzusetzen. Die Kommunikation zwischen dem Roboter und dem Hostcomputer wird durch ein zusätzliches *Parallel Interface* sowie eine serielle Verbindung ermöglicht und von RCI-Routinen überwacht. RCI ist die Grundlage von RCCL und bleibt dem Anwender weitgehend verborgen.

Ursprünglich wurde RCCL an der Purdue Universität um 1983 von V. Hayward programmiert [44, 21] und zwischen 1985 und 1990 von Lloyd und Parker an der McGill University (Montréal, Canada) sowie am Jet Propulsion Laboratory (NASA-JPL) zur heutigen Reife weiterentwickelt [31]. Anläßlich dieser Installation wurde es auf das Betriebssystem SUN OS 4.1 portiert. Dank der hervorragenden Netzwerkverbindung nach Montréal konnte das Aufspielen via *Internet* vor einem Installationsbesuch erfolgen, zu dem wir John Lloyd gewinnen konnten.

5.4.1 Warum Echtzeitkontrolle?

Komplexe Steuerungsaufgaben für einen Roboterarm sind oft von komplexen Sensorsignalen beeinflußt. Beispielsweise kann die Bewegung des Armes von der relativen Lage zu einem Objekt – sei es Ziel oder Hindernis – abhängig sein. Sollte sich diese Lage unvorhergesehen ändern, muß das Robotersystem entsprechend reagieren. Daraus erwächst die Notwendigkeit, breitbandige Eingangssignale, z. B. von Kameras, eventuell Drehmoment-Kraft-Sensoren oder taktiler und vielleicht sogar akustischer Sensoren, *in Echtzeit* zu verarbeiten und daraus Motorkommandos zu bestimmen. Echtzeit bedeutet, daß eine Reaktion in einer definierten Zeit erfolgen muß, und nicht von der momentane Auslastung mit anderen Aufgaben abhängen darf. Dies läßt sich nur erreichen, wenn der Roboter *eng* an eine leistungsfähige, flexibel programmierbare Recheneinheit gekoppelt ist, d. h. schneller und häufiger Datenaustausch untereinander möglich ist.

5.4.2 Echtzeitkontrolle unter UNIX: RCCL/RCI

UNIX ist ein *multiuser, multitasking* Betriebssystem. Dies bedeutet, daß normalerweise mehere Dutzend Programme[11], genannt Prozesse, „gleichzeitig“ abgearbeitet werden. Gleichzeitigkeit bedeutet in diesem Zusammenhang, daß ein *scheduler* die Verwaltung aller Resourcen wie CPU, Speicher, Platten I/O etc., übernimmt und je nach Bedarf (gesteuert über *Interrupts*) und Vereinbarung (Programmkode, Priorität) diesem oder jenem Prozeß die "Aufmerksamkeit" zuwendet (*time sharing*).

[11] Davon etliche, die für das Betriebssystem selbst wichtig sind (*daemons* für *Internet*, Plattenspeicher etc.).

Das Umschalten zwischen Prozessen (*context switching*) kann mit mehr oder weniger kurzer, jedoch meist kaum merklicher Zeitverzögerung erfolgen, abhängig davon, ob der *Speicherkontext*[12] des neuen Prozesses aus Platzgründen auf einem extra Plattenspeicherplatz (*swap area*) ausgelagert war. Ein Eichzeitkontrollprogramm hat härtere Anforderungen als ein gewöhnlicher Rechenjob zu erfüllen: bei Bedarf *muß* der Prozeß innerhalb einer definierten Zeit bedient werden, was einem UNIX–Prozeß normalerweise *nicht garantiert* wird.

Das RCCL/RCI System regelt das Problem durch Starten eines speziellen Prozesses, (*control task*) der dem Betriebssystem, ähnlich einem Kuckucksei, untergeschoben wird. Dieser Kontrolljob läuft dann im Hintergrund – dem Benutzer im wesentlichen verborgen – mit hoher Priorität als *connect–to–interrupt task*. Dies bedeutet, er wird in regelmäßigen Zeitabständen (≈10-100 Hz[13]) von einem *timer-interrupt* aufgeweckt und erledigt dann folgende Aufgaben:

1. Umschalten zum Speicherkontext des *control task* Prozesses.
2. Zustandsdaten vom Roboter abfragen und überprüfen.
3. Berechnung neuer Kommandos (zukünftige Winkelkoordinaten etc.), die dann über ein Parallelinterface an den Roboter zurückgesendet werden.
4. Zusätzlich können an dieser Stelle C–Funktionsaufrufe eingebunden werden, für die allerdings bestimmte Restriktionen und Vorsichtsmaßregeln gelten, da sie im Betriebssystemmodus[14] operieren. Damit können Echtzeitkorrekturen der Trajektorie durchgeführt werden, zum Beispiel Greifkorrekturen oder Ausweichreaktionen aufgrund Kamera- oder anderer Sensorsignale. Da diese Funktionen periodisch im Hintergrund ablaufen, eignen sie sich hervorragend für (zusätzliche) benutzerdefinierte „Wachhundfunktionen“, bespielsweise zur Überwachung von Kollisionsbereichen etc.
5. Restauration des alten Speicherkontextes

Um dieses Konzept des Roboterinterfaces zu realisieren, sind einige ungewöhnliche Modifikationen auf Seiten der Betriebssystemsoftware sowie zusätzliche Hardware notwendig. Die wesentlichen Grundzüge sollen im folgenden Abschnitt beschrieben werden.

[12]Speicherkontext eines Prozesses besteht aus Programmkode, Datenpuffer, etc.

[13]Defaultwerte werden beim Starten von einer Datei `./.rciparams` eingelesen, typisch 50 Hz.

[14]Im sog. *kernel mode* ist die Benutzung von *system calls* eingeschränkt, ferner sind keinerlei Schutzmechanismen gegen irrtümliche (oder mißbräuchliche) Zugriffe auf prozeßfremde Resourcen wirksam (führt sonst zu unschädlichen *segmentation-fault* Fehlermeldungen).

5.4.3 Modifikationen

5.4.3.1 Software

Der Prozeß, der durch einen *Interrupt* unterbrochen wird, ist nicht notwendigerweise derjenige, der die *control task* inititiert hat (Folge des *multi-tasking*). Da, wie schon angedeutet, nicht genügend Zeit vorhanden ist, um einen „Ausflug" in die *swap area* des Plattenspeichers zu unternehmen und nach Instruktionen und Daten zu suchen, muß der gesamte von der *control task* benötigte Speicherplatzkontext im Hauptspeicher residieren und vor dem möglichen Auslagern geschützt werden (*memory locking*). Die nötigen Vorkehrungen werden automatisch beim Start des RCI Programms getroffen[15].

Auf Seiten des *Robotcontrollers* wird die Steuersoftware VAL II komplett durch ein Kommunikationsprogramm (sog. *"moper"*) ersetzt. Es programmiert im wesentlichen den LSI-11 Steuerrechner im UNIMATION-*Controller* dazu um, neben elementaren Sicherheitchecks (z. B. Überlast oder NOT-AUS) Daten zu verteilen. Der *Controller* empfängt Daten über das *Parallel–Interface* vom Hostcomputer und verteilt sie auf die angegliederten Mikrocomputer (*joint mircos*) und das I/O *Interface*. Ferner sammelt er deren Zustandsdaten, um sie auf Abruf an den Hostrechner zurückzuschicken. Diese Software wird einmalig beim Anschalten des *Controllers* mittels eines Hilfsprogrammes (Konsolen-Emulation, siehe 5.4.4.2 und [32]) vom Hostcomputer geladen und gestartet.

Das Zusammenspiel der einzelnen Komponenten ist im Blockdiagramm 5.5 illustriert: Im Gehäuse des Hostrechners befinden sich neben der Recheneinheit (*Host*-CPU) die Bildverarbeitungseinschübe (siehe 5.5), sowie zusätzliche *Interface Hardware*. Die Kommunikation zwischen dem steuernden Programm (*"planning level"*) und der *"control level"* Software erfolgt asynchron. Die *"control level"* Software tritt, vom *timer board* initiiert, in regelmäßigen Zeitintervallen mit dem Steuerprozessor im *Robot Controller* in Verbindung (typ. 50 Hz). Die Gelenkmotor–*controller* arbeiten ebenfalls synchron mit etwa 1 kHz. (*timer board*; serielle Verbindung zwischen Rechner und *Robot-Controller* sowie *Interfaceboards* sind im Bild weggelassen)

[15]Dazu gehört neben dem *memory locking* selbst, das *patchen* von *Kernel*-Routinen. Das ist nötig, um beim Beenden des Prozesses das *memory locking* wieder rückgängig zu machen und *Interrupts* abzuschalten etc.

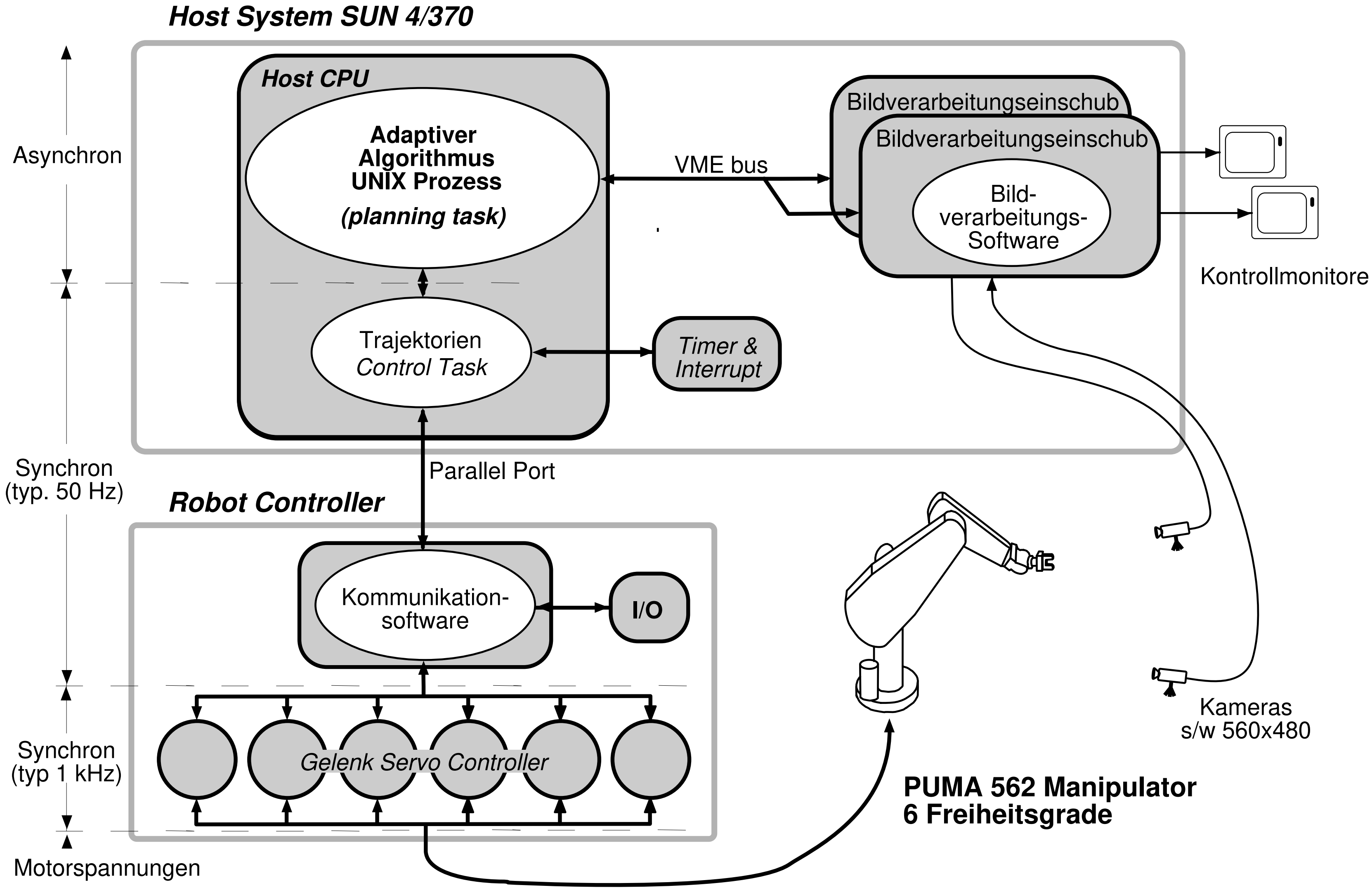


Abbildung 5.5: Blockdiagramm des Aufbaues, Erläuterung siehe Text.

5.4.3.2 Hardware

Es wurde folgende *Interface-Hardware* installiert:

- eine serielle Verbindungsleitung zwischen Unimation- *Controller* und seiner Konsole (wird durch einen einfachen Umschalter auf einen freien seriellen Port der SUN umgeleitet, zum Aufspielen von "*moper*" bei der Initialisierung des *Controllers*).
- dem LSI-11 Steuerprozesser im UNIMATION-*Controller* wurde eine *Parallel-Interface*-karte zugefügt (Q-Bus Parallel-Port Modul (48 bit) DEC DRV11 A).
- eine *Parallel-Interface*-karte im Hostrechner (VME-Bus *Parallel-Port* Modul (64 bit) XYCOM XVME 240).
- Kabel-Interface (verbindet zwei 40-adrige mit zwei 80-adrigen Breitbandkabeln; ein Kontroll-LED [an *Handshake*–Signal] zeigt aktiven RCI Prozeß).
- ein programmierbarer *Real-Time-Counter* und *Interrupt*–Einschub im Hostrechner (XYCOM XVME 293). Dies vermeidet das Verlieren von Zeittakten, und bietet zudem die Möglichkeit der Zeitmessung mit sub-μs Präzision.
- fernsteuerbare "*Arm–Power–ON*" Schaltung. Dies erlaubt, innerhalb eines Programmes die Motor-Stromversorgung ein- und auszuschalten. Normalerweise muß beim Programmstart ein Taster am *Controller* gedrückt werden. Dies entfällt durch Verschaltung eines Parallel-Port Ausganges (#2) mit einer entsprechenden Steuerleitung am *Controller* [59].

5.4.4 Die Programmbibliothek

5.4.4.1 Die Robot–C*ontrol*–C–L*ibrary* (RCCL)

RCCL ist ein umfangreiches Paket von C–Routinen, die sich wie folgt kategorisieren und kurz beschreiben lassen:

- Bewegungskommando-Primitive in kartesischen oder Gelenkwinkelkoordinaten. Letztere werden durch ein Tupel von Endwinkeln spezifiziert. Kartesische Zielpositionen werden durch eine Transformationsgleichung der Form

$$T_{Basis}\, T_6\, T_{Werkzeug} = T_{Tisch}\, T_{Objekt}\, T_{Greifposition} \quad (T_i \in \mathbf{R}^4 \times \mathbf{R}^4) \tag{5.1}$$

dargestellt. Dabei ist T_6 die zu lösende homogene Koordinatentransformationsmatrize[16] von der Schulter zum Handgelenk. Die Bewegung wird wahlweise im kartesischen oder im Gelenkwinkelraum von der momentanen Armposition linear interpoliert.

- Kontrollfunktionen für die Bewegungssynchronisation. Bewegungsbefehle werden automatisch in eine Warteschlage eingereiht (*scheduling*) und werden zur frühest möglichen Zeit ausgeführt. Flexiblere Synchronisationsmöglichkeiten werden durch einen seperaten Funktionensatz gewährleistet.

- Kontrollfunktionen für den Trajektoriengenerator. Trajektorien werden mittels *Spline*-Funktionen fünfter Ordnung berechnet. Randbedingungen wie maximale Geschwindigkeiten und Beschleunigungen werden von einem Konfigurations-File gelesen, lassen sich aber innerhalb des Programmes bequem verändern.

- Allgemeine Kontrollfunktionen (z. B. für das Verhalten der *control task* im Falle einer abnormalen Situation; für Kommunikation mit dem Handeingabecomputer, dem *teachpendant*, und den I/O Ports) und Fehlermeldungsverwaltung (error stack).

- Routinen für kinematische Berechnungen (Vorwärts- und inverse Kinematik etc.)

- Datentyp-Definitionen für in der Robotik nützliche Objekte, sowie Routinen für deren Manipulation (z. B. Vektoren, 4×4 Matrizen für homogene Koordinatentransformationen, Eulerwinkel-Repräsentationen etc., sowie spezielle RCCL–Datenbasis Strukturen, in denen nützliche Roboterparameter unterhalten werden) .

5.4.4.2 Hilfsprogramme

Zusätzlich steht ein Bündel von Hilfsprogrammen zu Verfügung (*utilities*):

`down`

Programm zum Aufspielen des LSI–ausführbaren Kodes auf den Unimation–*Controller* (alias `startBachus`).

`termlink` Programm zur Emulation der Unimation–*Controller* Konsole in einer UNIX–Shell (alias `talkBachus`).

[16] Homogene Koordinatensysteme beschreiben kompakt u.a. Raumtranslationen sowie Rotationen und Projektionen. Sie sind auf den Gebieten der Computergraphik schon früh verbreitet gewesen und wurden von Denavit und Hartenberg [10] als Beschreibungkonvention für Robotergliedmaßen etabliert. Siehe [6, 44, 3].

`pumacal`

Programm zum Kalibrieren der inkrementalen Winkel-Encoder[17] (erfolgt automatisch mit `startBachus`).

`move`

Programm, das den Arm in eine spezifische Winkelkonfiguration bewegt. Die Zielposition kann als absoluter oder zur momentanen Situation relativer Satz von Winkeln oder als eine in einer Datei definierte Standardkonfiguration (z.B.: `park, ready, rcclpark, learn`) gegeben werden. (Beachte Fußnote 18).

`zerograv`

Programm, das den Roboter in einen "gewichtslosen" Modus (*zero-gravity mode*) versetzt, in dem alle Gravitationsgelenkmomente kompensiert werden, und das erlaubt, den Arm mit einem Minimum an Kraftaufwand frei zu bewegen.

`playback`

Programm, das die RCCL Routine `rcclTeach()` aufruft, und in einfacher Weise erlaubt, den Roboter mit der Tastatur und dem *teachpendant* zu bewegen (Taste *"FREE"* schaltet den *zero-gravity mode* ein). Eine Sequenz von Trajektorienpunkten kann gespeichert und wiederholt werden.

`free`

Programm zum selektiven Lösen der Gelenkbremsen[18].
Vorsicht ist hier geboten: der Arm fällt wirklich frei.

`robotsim`

Ein Robotsimulatorprogramm, das vorgibt, der *Robot–Controller* zu sein und damit ermöglicht, RCCL–Programme ohne aktiven Roboter zu testen. Ein Graphikpaket, das die simulierte Roboterbewegung am Bildschirm visualisiert, ist in Entwicklung.

On-Line **Dokumentation**

Die gesamte Referenzdokumentation zu RCCL und RCI ist sowohl in Papierform als

---

[17] auf der Basis der Potentiometerkalibrierung, die ursprünglich mit `primecal` und `potcal` gewonnen wurde (siehe man pages).

[18] Das Programm `free` muß benutzt werden, wenn der folgende Fall eintritt: Die Winkelencoderpulse werden von einem 16-bit Zähler zu einem Registerwert verarbeitet. Im *zero-gravity mode* wird die Überschreitung der Software-Winkellimits nur durch Warnsignale angezeigt. Sie verhindern aber nicht, daß in ungünstigen Situationen – jenseits des normalerweise zulässigen Bereiches – der Zähler "überläuft" und von plus nach minus 32767 (oder umgekehrt) springt. RCI beantwortet dieses Ereignis mit einem Abort wegen „Geschwindigkeitslimits überschritten". Ein Neustarten eines normalen Programmes wird nun nicht mehr gelingen, da sich der Arm außerhalb zulässiger Bereiche befindet. In diesem Falle muß `free` zweimal unter Drehung des betroffenen Gelenkes in den Normalbereich aufgerufen werden.
Warnung: hier könnte ein Befehl (z.B.) "`move to park`", zu fatalem Rückwärtslaufen führen, da das Encoderregister einen Winkelwert auf der anderen Seite der Normallage vorspiegelt.

auch als *on-line manual pages*[19] in elektronischer Form verfügbar, was beim Programmieren eine große Hilfe ist.

5.5 Bildverarbeitung

Für den Einsatz des Roboters in Arbeitsumgebungen mit sich bewegenden Objekten ist eine effektive Bildvorverarbeitung notwendig, die den Hostcomputer so wenig wie möglich belastet und doch in hoher Wiederholfrequenz Bildinformation wie z. B. Objektkoordinaten und charakteristische Abmessungen liefert.

An der Spitze der technischen Lösungen stehen Bildverarbeitungsanlagen in *Pipeline*-Architektur, wie sie z. B. von der Firma Datacube Inc. angeboten werden. Dieses Konzept basiert auf einer Palette von VME-Einschüben, die hochspezialisierte Prozessoren beherbergen und mittels Flachbandkabel (von Einschub zu Einschub) verschaltet werden. Beispiele sind Digitalisierer (*Framegrabber* für Videosignal), Bildspeicher, Arithmetikprozessor, Konvolutionsprozessor, Display- und Graphikkarte. Verschiedene Drittanbieter erstellen Hilfssoftware, die die äußerst komplizierte Register- und Multiplexerprogrammierung transparenter machen und den Bilddatenstrom zu den verschiedenen Prozessoren leiten (*routing*). Änderungen des Algorithmus machen typischerweise ein Umkonfigurieren der Hardware erforderlich, u. a. durch Hinzufügen oder Entfernen von Einschüben. Ein flexibles leistungsfähiges System für zwei unabhängige Kamerakanäle benötigt damit eine Minimalkonfiguration, die sich im Rahmen des vorgegebenen Budget nicht realisieren ließ.

Wir haben uns für die folgende Alternative entschieden: Es ist das Bildverarbeitungssystem ICS-400, das von der Firma Androx Inc. vermarktet wird, und auf dem Konzept paralleler **d**igitalen **S**ignal**p**rozessoren (DSP) basiert. Der Unterschied zum oben dargestellten Konzept der Firma Datacube Inc. besteht darin, daß für kompliziertere Bildoperationen der Bilddatenstrom des Eingangssignales kurzzeitig unterbrochen werden muß, hingegen in der *Pipeline*-Architektur dies mit einem genügend großem Sortiment an VME-Einschüben meist vermieden werden kann.

Unsere Konfiguration besteht aus zwei unabhängigen VME-Einschüben, jeder wie folgt bestückt mit (siehe Abbildung 5.6)

- vier Videoeingangskanälen von dem jeder mit 8 frei programmierbaren *Lookup Tables* (LUT 8 bit Auflösung, 8*[8 bit→8 bit] = 8×256×8 bit; siehe 5.5.1) ausgerüstet ist; jeweils einer dieser *Lookup Tables* kann ausgewählt werden; Maxi-

[19] Die *On-Line* Dukumentation ist mit dem UNIX Befehl "`man <name>`" lesbar und mit dem Befehl "`apropos <keyword>`" kann nach Schlüsselwörtern gesucht werden.

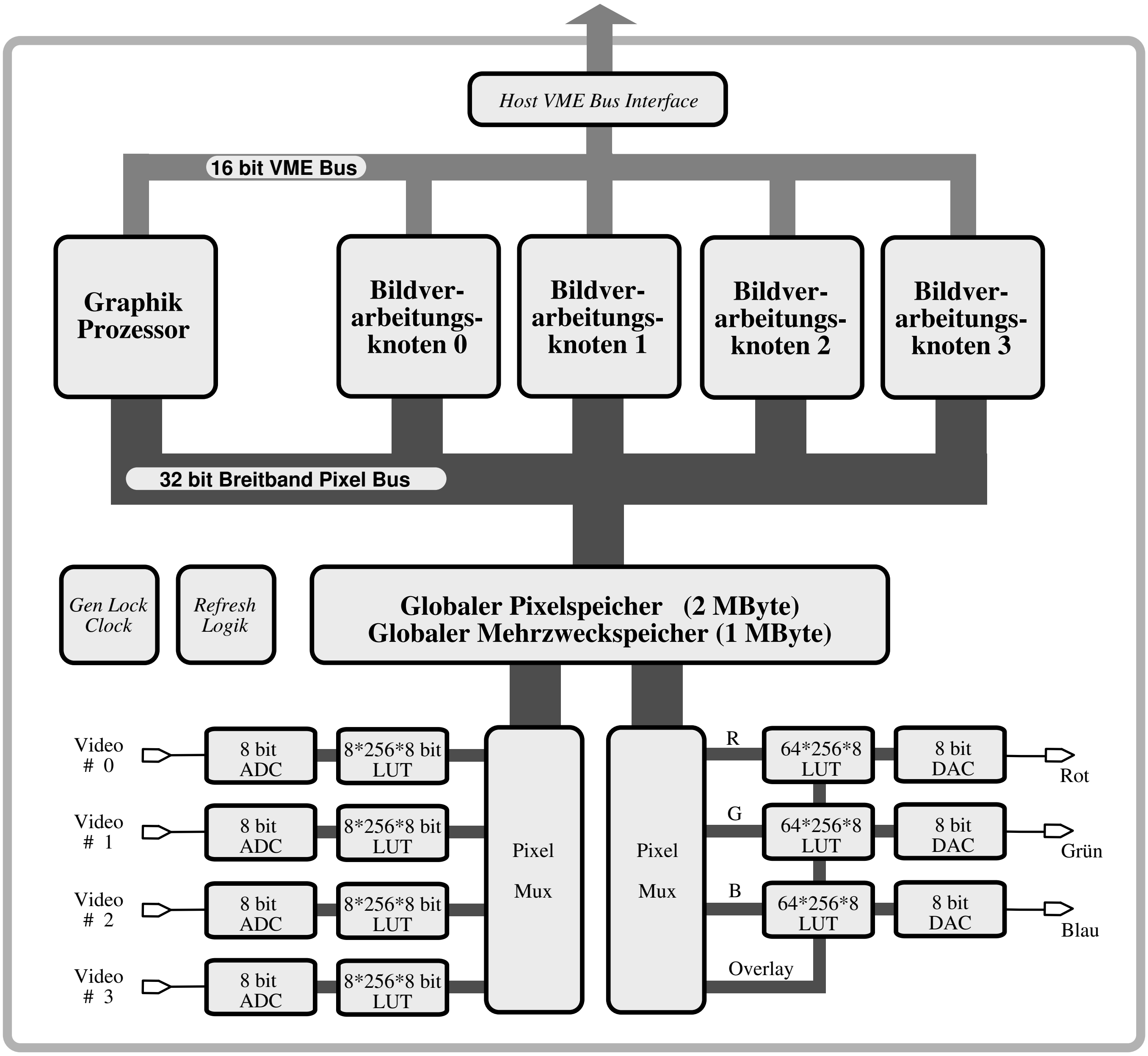


Abbildung 5.6: Die *shared–memory* Architektur des Bildverarbeitungseinschubes Androx ICS-400 gestattet simultan mehrere Videosignale einzulesen (links unten), anzuzeigen (rechts unten) und von den vier Bildverarbeitungsknoten bearbeiten zu lassen. Jeder dieser vier Knoten besitzt einen zusätzlichen Befehls- und Daten-*cache* (schneller Speicher); Text und Graphik lassen sich mittels des Graphikprozessors gestalten. Die Steuerung erfolgt vom Hostrechner über das VME–*Interface*.

mal 4 monochrome Kameras oder eine RGB Farbkamera können angeschlossen werden (intern oder extern synchronisiert, Trigger optional);

- einem 2k $\times$ 1k (2 MByte) globalen Bildspeicher (frei konfigurierbar, typischerweise in Kombinationen von 512$\times$512 Pixelblöcken).

- vier digitale 16-bit Signalprozessoren (ADSP 2100), mit 64kB Bildspeicher *(cache)*;
- einem Graphikprozessor für Bildbeschriftung und *Overlays* (siehe 5.5.1);
- einem 1 MByte Graphik- und Mehrzweckspeicher (*Scratch Pad Memory*);
- drei Videoausgangskanälen mit je 64 umschaltbaren, frei programmierbaren *Lookup Tables* (64*[8 bit$\rightarrow$8 bit] = $64\times256\times8$ bit) für drei schwarz-weiß Bildschirme oder (typischerweise) einen RGB Farbmonitor.

Somit steht pro Kamera die volle Rechenkapazität eines Bildverarbeitungseinschubes (im folgenden auch nur *"Board"* genannt) zur Verfügung. Charakterisierende *Benchmark*-werte sind in Tabelle 5.5 gegeben.

Komplexe 2D *Fast Fourier Transformation* FFT	800 ms
Differenzbild	28 ms
3×3 Konvolution (beliebige Koeffizienten; 8 oder 16-bit)	66 ms
Grauwerthistogramm ($\leq$ 12 bit)	33 ms
Rotation mit Interpolation (beliebige Winkel; am 256×256 Bild)	33 ms

Tabelle 5.2: Zeitbedarf für ausgesuchte Bildoperationen auf den Bildverarbeitungskarten Androx ICS-400; sofern nicht anders angegeben gelten die Werte für die Bildgröße 512×512 und 8-bit Auflösung

Die Bildverarbeitungseinschübe erscheinen im UNIX Dateisystem als /dev/ax0 bzw. /dev/ax1, und ein Gerätetreiber *(device driver)* übernimmt die Kommunikationsabwicklung (*Handshaking, Interrupts*), sowie die Verwaltung der Prozessorresourcen *(event-scheduling* oder *Resource Management).* Die *shared–memory* Architektur wird dabei dynamisch als MIMD[20] oder SIMD[21] System betrieben, d. h. alle vier Prozessoren führen verschiedene oder gleiche Instruktionen auf verschiedenen Daten im gemeinsam zugänglichen Bildspeicher aus. Wie die Koordination der Signalprozessoren transparent gemacht wird, läßt sich an einem Beispiel kurz erläutern. Ein typischer Befehl im aufrufenden C–Programm, z. B. zur Subtraktion zweier Bilder, sieht so aus:

$$\begin{aligned} new\_event\_num \;=\; & \texttt{isp\_event(SUB} \mid \texttt{BRD(}board\_num\texttt{)} \mid \\ & \texttt{PEV(}previous\_event\_num\texttt{)} \mid \texttt{NPROC(}num\_proc\texttt{)}, \\ & X1, Y1,\ Xsize, Ysize,\ X2, Y2, Xresult, Yresult\texttt{);} \end{aligned} \tag{5.2}$$

[20] *Multiple instruction multiple data.*

[21] *Single instruction multiple data.*

new_event_num ist eine ganze Zahl (1-255), die der *scheduler* zurückgibt (im Falle eines auftretenden Fehlers negativ). `SUB` ist der Name der Funktion; *board_num* spezifiziert eines der beiden *Boards*; *num_proc* ist die Anzahl der zu beteiligenden Prozessoren (Standard ist die maximal verfügbare Anzahl). Die *num_proc* Prozessoren werden mit dem entsprechenden Programmkode instruiert und beginnen die Ausführung, sobald der vorherige Befehl mit der Nummer *previous_event_num* beendet ist (bei Nichtangabe von *previous_event_num* wird die Ausführung begonnen, sobald die Prozessoren frei sind). Durch geschicktes Arrangieren der Funktionsaufrufe im Hauptprogramm können beide *Boards* simultan Bildinformation extrahieren, ohne den Zentralrechner ernsthaft zu belasten oder das aufrufende Programm zu blockieren.

Die restlichen Argumente varieren je nach Befehl. Im obigen Beispiel werden drei gleichgroße Fenster im Bildspeicherbereich definiert, in denen die Operation ausgeführt werden soll. Diese sind stets rechteckig und werden durch die linken oberen Eckkoordinaten (*X1, Y1*), (*X2, Y2*), (*Xresult, Yresult*) und deren Seitenlängen (*Xsize, Ysize*) angezeigt.

5.5.1 Bildaufnahme und Anzeige; LUT und *Overlay* Technik

Blockdiagramm 5.6 zeigt die wesentlichen Komponenten und die Datenpfade in einem Bildverarbeitungs-*board*. Bis zu vier monochrome Kamerasignale können simultan digitalisiert und in einen globalen Bildspeicher geladen werden. Bei Verwendung einer Farbkamera werden drei für die RGB-Signale genutzt (folgende Synchronisationquellen sind programmierbar: extern oder intern von einem RS 170, RS 370 oder PAL Signal; auch alternative Zeitbasen und Triggerung sind möglich).

Die drei Videoausgänge speisen in der Regel einen RBG-Monitor, wobei durch die Software verschiedene Auflösungen (512×512 @ 30/60 Hz oder 1024×1024 @ 30 Hz; *interlaced/non-interlaced*[22]) ausgewählt werden können. Ferner wird ein Bildspeicherbereich definiert, der permanent ausgelesen und angezeigt wird. Dabei kann gleichzeitig ein Bild in einem Speicherbereich aufgenommen und ein anderes angezeigt werden.

Das Konzept mehrerer *Lookup–Tables* (LUT) bietet pro Kanal instantan umschaltbare, frei programmierbare Abbildungen von einem 8-bit zu einem (ggf. anderen) 8-bit Wert. Dies ermöglicht zum einen einfache Kennlinienkorrekturen der Videoeingangssignale (8 LUTs, z. B. Gamma-, Farbkorrektur etc.) und zum anderen

[22] *interlaced* Mode ergibt ein weniger flackerndes Bild, wobei abwechseld zwei *Halbbilder* kodiert werden: zunächst ein Halbbild mit den ungeraden Zeilen 1,3,5,... , dann die geraden Zeilen 2,4,6... .

vielfältige Anwendungen für die Videosignalausgabe (64 LUTs) in Verbindung mit der sogenannten *Overlay*-Architektur. Dazu wird ein Bereich des globalen Bildspeichers als *Overlay-frame* deklariert. Dessen Inhalt (obere 6 bit) selektiert pro Bildpunkt eine der 64 LUTs pro Farbkanal (Rot, Grün und Blau, siehe Abbildung 5.6). Bei entsprechender LUT Initialisierung ergibt sich z. B. eine effektive Farbskalenkodierung (kein zusätzlicher Rechenaufwand) oder ein *Overlay*-Bild. In einem *Overlay*-Bild lassen sich Graphik und informationsreiche Beschriftung über dem angezeigten Bild erzeugen. Ein Pixelwert 0 im *Overlay-frame* selektiert dann eine (1:1)–Abbildungs–LUT (transparent). Alle anderen Werte wählen LUTs, die unterschiedliche Farbtöne generieren (deckende und/oder opake).

5.5.2 Die Softwarebibliothek

Das Bildverarbeitungssystem kann auf zwei Arten angesteuert werden. Erstens über Funktionsaufrufe in einem C–Programm und zweitens mittels eines menüunterstützten, interaktiven Kommandointerpreters AADE (*Androx Application Development Environment*, mit einer C ähnlichen Sprache). Es läuft als Applikation unter der Standardbenutzeroberfläche des SUN OS (SUNVIEW) und erlaubt alle C–Funktionsaufrufe interaktiv mit Auswahlmenüs einzugeben. Dies hat sich für die Programmentwicklung als nützlich erwiesen und kompensiert teilweise das Fehlen einer *On-Line* Dokumentation[23].

Die Bibliothek von etwa 400 Funktionen gliedert sich in folgende vier Bereiche [1]:

(1) **Videokontrollkomandos:**
Diese Gruppe ermöglicht die Programmierung der Register, Multiplexer und LUTs, die die Videodatenpfade kontrollieren. Neben dem allgemeinen Funktionaufruf `vid_event()` enthält sie einige wichtige Initialisierungsroutinen, die die *Boards* in eine der Standardkonfigurationen versetzt (z. B. `mono_init()`, `init_video_512()`).

(2) **Bildverarbeitungsfunktionen** {`isp_event()`}:
Diese Gruppe enthält einen umfangreichen Satz von Operationen, die in einem beliebigen rechteckigen Bereich des Bildspeichers von den Signalprozessoren ausgeführt werden. Darunter sind:

- Filter Operationen (z. B. Sobel, Laplace; beliebige Konvolutionen);

[23]Da die Syntax der Funktionsaufrufe einer ausgefeilten durchgängigen Struktur entbehrt, schleichen sich relativ leicht Programmierfehler ein. Aufgrund der speziellen Art der Softwareimplementation generieren Argumentfehler weder beim *compilieren* noch beim Ablauf hilfreiche Fehlermeldungen und sind somit relativ schwer zu lokalisieren (für symbolisches *debugging* fehlt der Quellkode).

- Morphologische Transformationen (z. B. Erosion, Dilatation);
- Logische Operationen (UND, ODER etc.);
- Arithmetische Operationen (Subtraktionen, Multiplikation etc.);
- Farbtransformationen zwischen Koordinatensystemen (RBG, HSI u.a.);
- Erzeugung und Modifikationen von Grauwerthistogrammen;
- Spezielle Bildtransformationen (z. B. FFT und beliebige Rotationen);
- Funktionen für Matrizen- und Vektorarithmetik.

(3) **Graphikprozessorfunktionen {`grp_event()`}:**
Diese Gruppe enthält Funktionen für das Erzeugen von Text, Liniengraphik und Darstellungen einfacher 2D–Objekte.

(4) **Diverse Applikationsfunktionen:**
Diese Gruppe enthält Funktionen, die eine Beteiligung des Hostrechners erfordern. Erwähnenswert sind:

- `attach()` und `release()`: Routinen, die eine Bildverarbeitungskarte dem aufrufenden UNIX-Prozeß exklusiv zuordnen bzw. die Karte wieder freigeben;
- Datentransferkommandos zum Kopieren von Daten zwischen Bildspeicher, LUTs, Hostspeicher und Dateisystem;
- *Scheduler*–Kontrollkommandos;
- Komplexere Funktionen wie z. B. `do_blob()`, eine Funktion, die zusammenhängende Gebiete in einem Binärbild analysiert.

Es soll dem interessierten Leser nicht verschwiegen werden, daß die technische Dokumentation hinsichtlich Vollständigkeit und Didaktik Wünsche offen läßt[24]. Das System hat sich jedoch in der Praxis gut bewährt und die Wahl dieses flexiblen, preiswerten und leistungsfähigen Bildverarbeitungssystems erscheint auch im Nachhinein gerechtfertigt.

5.5.3 Kameras

Zur Zeit ist das Labor mit zwei hochauflösenden, monochromen Industriekameras des Typs COHU 4815-3000 ausgestattet. Dabei wurde auf ein Standardobjektivflansch geachtet, der zu eine große Palette von alternativen Objektiven kompatibel ist (Standard C–*mount*; mit Weitwinkelobjektiven 25 mm und leihweise z. Z. 16 mm). Die Bildgebung basiert auf einem 2/3"CCD–Chip, der ein 560×480 Pixel aufgelöstes

[24]Einige Softwarefehler sind vorhanden und offensichtlich teilweise bekannt, doch nicht dokumentiert.

Bild liefert. Das Videosignal wird als RS 170 Standard an 70 Ohm ausgekoppelt. Stative sorgen für die nötige Standfestigkeit.

5.5.4 Kontrollmonitore

Zwei kleine schwarz / weiß Kontrollmonitore zeigen die unverarbeiteten Kamerabilder u. a. zum Überwachen der Fokusierung (PANASONIC TR930, 9”). Zwei weitere RGB Farbmonitore dienen als Anzeigebildschirme für die beiden Bildvorverarbeitungseinschübe. Der eine ist ein 13 Zoll großer SONY PVM 1545 Bildschrim (hohe Farbbrillianz), der andere ist ein *multi-sync*fähiger MIRCO VITEC 1919 LP mit einem 19” großen Schirm (d. h. automatische Umschaltung zwischen *interlaced/noninterlaced* und 30/60 Hz; zur Anzeige eines 1k$\times$1k großen Bildspeicherbereiches nötig).
Ein Regalwagen der dem Zentralrechner mit Bildverarbeitungssystem und den Monitoren Platz gewährt, hat sich im Labor als sehr nützlich erwiesen (Mobilität und Zugänglichkeit aller Steckverbindungen).

Kapitel 6

Implementation des visuomotorischen Kontrollalgorithmus

Im vorigen Kapitel wurde das Design und die Ausstattung des Roboterlabors beschrieben. Nun soll die Erläuterung der Implementation des bereits in Kapitel 4 besprochenen Kontrollalgorithmus erfolgen.

Wie bereits erläutert, gelingt das Erlernen der Positionierfähigkeit des Roboterendeffektors durch eine Folge von Übungsversuchen. In diesen Trainingsversuchen muß den Kameras eine Zielposition im Arbeitsbereich vorgegeben werden. In Abschnitt 6.2 soll das *"split-brain"* Lernverfahren erläutert werden, welches ein autonomes Üben des Roboters ermöglicht.

Darauf folgt ein Abschnitt über das verwendete Verfahren zur Extraktion der Ziel- bzw. Endeffektorposition aus den Grauwertbildern der Kameras. In Abschnitt 6.4 sollen einige Erläuterungen zum Programm `rob` folgen, in das der neuronale Kontrollalgorithmus eingebettet wurde. Die für die Experimente gewählten Parameterinitialisierungen beschreibt der Abschnitt 6.5. Zunächst wird jedoch der experimentelle Versuchsaufbau vorgestellt.

6.1 Der Versuchsaufbau

Aufgrund der räumlichen Gegebenheiten ergab sich der folgende Versuchsaufbau: Der Puma Roboter ist aufrecht auf einem Tisch befestigt. Die beiden Kameras stehen in 3–4 m Abstand auf Stativen montiert und blicken mit einem Disparitätswinkel von 53° auf den relevanten Arbeitsbereich. Eine Kamera ist horizontal, die andere leicht nach unten (Elevation 11°) ausgerichtet. Der gewählte Arbeitsbereich ist im wesentlichen ein den Kameras zugewendetes Kugelsegment von einem Meter Radius, siehe Abbildung 6.1 (Seite 90).

Um den mit der Bildsegmentation zusammenhängenden Problemkomplex (Aufteilung des Bildes und Identifikation zusammengehöriger Objekte, siehe z. B. [16]) für die ersten Experimente überschaubar zu halten, wurde eine einfache Szene gewählt. Dazu wurde in der Greiferhand eine kleine Glühlampe fixiert, welche programmgesteuert ein- und ausgeschaltet werden kann. Damit läßt sich die Raumposition dieses Endeffektorpunktes einfach gegen jeglichen Hintergrund diskriminieren.

Als besonders geeignet fand sich eine 3 V-Spezialbirne für Miniaturtaschenlampen (Maglite Inc, CA) welche sich durch sehr geringe Wendelabmessungen (kleiner Raumwinkel aus Kamerasicht) und eine isotrope Abstrahlcharakteristik auszeichnet. Um einen unerwünscht großen, überstrahlenden Leuchtfleck in den Kamerabildern zu vermeiden, wird die Lampe gedimmt, was sich auch günstig auf deren Lebensdauer auswirkt. Die Stromversorgung wird von einem Relaisausgang des UNIMATION CONTROLLERs geschaltet[1]. Ein- und Ausschalten erfolgt mit den Aufrufen `SET_PIO()` und `CLEAR_PIO()` aus dem C-Programm. Dabei ist eine geringfügige Zeitverzögerung zu berücksichtigen: diese Befehle werden erst beim nächsten *control task*-Zyklus von der SUN an den UNIMATION CONTROLLER übermittelt (max. ca. 30 ms), welcher ein Relais umschaltet. Hinzu kommt die thermischer Trägheit der Lampe, was letztlich das Hell- bzw. Dunkelwerden der Lampe um ca. 100 ms verzögert und bei der Bildauswertung berücksichtigt werden muß

Für die manuelle Zielpräsentation steht eine an einem Stab montierte Lampe[2] zur Verfügung, mit der ein Zielpunkt vorgegeben wird.

6.2 *"Split Brain"* Lernverfahren

In der Lernphase benötigt der in Kapitel 4 vorgestellte „Neuronengas"-Algorithmus mehrere hundert Positionierziele, die zufällig im vorgegebenen Arbeitsbereich gewählt werden. Die unkorrelierte Sequenz von Zielpunkten mit einer bestimmten, räumlichen Häufigkeitsverteilung im Arbeitsbereich ist wichtig, um eine gleichmäßige, „unvoreingenommene" Verteilung der Neuronen zu erzielen.

[1] Die Lampenspannung wird unter Nutzung der internen Verdrahtung des Puma-Roboters direkt an einem Buchsenpaar am Oberarm des Puma zugänglich gemacht (Relaisausgang #1 = Parallel Port 0, Bit 1; Kontroll-LED #1; s. a. 5.2.4 und [59]).

[2] Diese Lampe wird vom Relaisausgang #17 des *Controllers* kontrolliert und ist damit vom Programm über die Parallelschnittstelle (Port 1, Bit 1) steuerbar.

Am einfachsten läßt sich dies dadurch erreichen, daß ein Zufallszahlengenerator einen Raumpunkt auswählt und der Roboter diesen selbst anfährt. Dies vermeidet jeglichen manuellen bzw. zusätzlichen experimentellen Aufwand.

Der Gesamtablauf besteht aus zwei Teilen, die abwechselnd den Roboter ansteuern:

Zielvorgabe (mit konventioneller Steuerung)

- Zufällige Auswahl eines Punktes im Arbeitsbereich;
- Zielvorgabepunkt anfahren — Stop;
- Zielposition in den Kamerabildern $\mathbf{u}_{target}$ merken;
- zur alten Position zurückfahren[3].

Positionieren üben (Aufgrund der Bildinformation ist das Ziel mit Hilfe des neuronaler Kontrollalgorithmus wiederzufinden)

- Ziel anfahren $\rightarrow \vec{\theta}_{initial}$ (Grobbewegung);
- Endeffektorposition in den Kamerabildern $\mathbf{v}_{initial}$ feststellen;
- Feinkorrektur $\Delta\vec{\theta}$ der Armstellung $\rightarrow \vec{\theta}_{final}$;
- Endeffektorposition in den Kamerabildern $\mathbf{v}_{final}$ feststellen;
- Augrund der Positionierfehler und vermittels der Lernschritte (4.4)-(4.6) die Ausgabewerte $\vec{\theta}_\mu$ und $\mathbf{A}_\mu$ verbessern.

Beide Teile, Zielvorgabe und Positionieren üben, laufen alternierend im selben Steuerprogramm ab, ohne untereinander motorische Informationen auszutauschen. Daraus resultierte der Name *"Split-Brain"* Verfahren. Zu Demonstrationszwecken kann ein Ziel natürlich auch manuell mit dem in Abschnitt 6.1 erwähnten Stab präsentiert werden.

6.3 Bilddatenextraktion

Die Aufgabe der Bilddatenextraktion ist das verläßliche Erkennen eines kleinen Objektes (Miniaturlampe), welches heller ist als der Hintergrund. Das gewünschte Resultat ist ein vierdimensionaler Vektor, der die momentane Ziel- bzw. Endeffektorposition in Pixelkoordinaten enthält. Der dazu verwendete Algorithmus ist relativ einfach und soll hier kurz beschrieben werden.

[3] Das Wiederaufsuchen der alten Armposition ist für den Lernprozess nicht relevant und kann optional unterdrückt werden.

6.3.1 Bestimmung der Objektposition

Die Bildvorverarbeitung arbeitet simultan auf beiden Bildverarbeitungseinschüben, im folgenden durch den Index $k \in \{links, rechts\}$ angedeutet und stellvertretend für beide, anhand eines Bildes erläutert: Zunächst wird (pro Kamera) ein aufgenommenes Bild mit eingeschalteter Lampe in einen separaten Bildspeicherbereich kopiert. Die mit einer Genauigkeit von 8 bit aufgelösten Pixel-Grauwerte in den Bildkoordinaten i und j seien als ${}^{k}B_{ij} \in \{0, 1, ...255\}$ bezeichnet. Ein Grauwerthistogramm (Häufigkeitsbestimmung eines Grauwertes im Bild) wird erstellt und daraus die maximale Bildhelligkeit ${}^{k}m_{hell}$ sowie die Zahl der Elemente ${}^{k}n_{hell}$, in der Menge der *maximal hellen* Bildpunkte ${}^{k}F_{hell}$,

$$
\begin{aligned}
{}^{k}F_{hell} &:= \left\{(i,j) \mid {}^{k}B_{ij} = {}^{k}m_{hell}\right\}, \text{ mit} \\
{}^{k}m_{hell} &:= \max_{i,j} {}^{k}B_{ij} \qquad (6.1)
\end{aligned}
$$

bestimmt. Um Rechenzeit zu sparen (zur Erinnerung: es handelt sich bei jedem Bild um einen Datensatz von 0.26 Mbyte) wird nun die *minimale, rechteckige* Bildfläche ${}^{k}F_{rect}$ gesucht, die alle hellsten Pixel ${}^{k}F_{hell}$ enthält. $({}^{k}_{\square}i,\ {}^{k}_{\square}j)$ mögen den Schwerpunkt dieser Rechteckfläche ${}^{k}F_{rect}$ in Pixelkoordinaten bezeichnen.

Der gewünschte vierdimensionale Positionsvektor wird aus den Schwerpunktskoordinaten beider Bilder zusammengegestellt und wird als $({}^{links}_{\square}i,\ {}^{links}_{\square}j, {}^{rechts}_{\square}i,\ {}^{rechts}_{\square}j)$ Vektor zurückgegeben. Die Auflösung ist bei diesem Verfahren 0.5 Pixel.

6.3.2 Konsistenztests

Um nun sicherzustellen, daß $({}^{k}_{\square}i,\ {}^{k}_{\square}j)$ wirklich das gewünschte Objekt darstellt, werden eine Reihe von Konsistenzbedingungen überprüft. Dies ist erforderlich, um nicht fälschlicherweise einen hellen Reflex in einem anderen Bildbereich als Teil des Objekt zu interpretieren und um zum Beispiel zu klären, ob sich das Objekt überhaupt im Gesichtsfeld *beider* Kameras befindet.

Im Fehlerfall wird das gegebene Objekt als nicht–identifiziert (FEHLER) markiert, was dazu führt, daß der Versuch wiederholt wird. Erscheint das Objekt nur suspekt, wird dies mit einer WARNUNG quitiert und ein Zusatztest ausgeführt.

6.3.2.1 Bedeckungsverhältnis: (Kompakte Form?)

Der hellste Teil des Zielobjekt sollte eine massive runde Form haben. Dazu wird der Bedeckungsanteil $R_{Bedeckungsverhältnis}$ der hellsten Pixel in der (sie einschließenden, minimale) Rechtecksfläche F_{rect}

$$\begin{aligned} R_{Bedeckungsverh\ddot{a}ltnis} &:= \frac{{}^k n_{hell}}{{}^k n_{rect}} \begin{cases} \geq C_1 = 0.7 & \text{sonst } \textsc{Warnung}. \\ \geq C_2 = 0.4 & \text{sonst } \textsc{Fehler}. \end{cases} \\ \text{mit} \quad {}^k n_{hell} &:= \text{Pixelzahl in } {}^k F_{hell} \\ {}^k n_{rect} &:= \text{Pixelzahl in } {}^k F_{rect} \end{aligned} \tag{6.2}$$

bestimmt und minimale Annehmbarkeitsschranken überprüft. Ein kreisförmiges (oder in xy–Richtung elliptisches ausgedehntes) Leuchtobjekt sollte ein $R_{Bedeckungsverh\ddot{a}ltnis}$ von $\pi/4 \simeq 0.79$ ergeben; Werte, die weit darunterliegen, werden von verstreuten Pixelklustern erzeugt, die dann als suspekt bzw. als Irrtum klassifiziert werden.

6.3.2.2 Ausdehnung des hellsten Objektbereichs: (Klein genug?)

Die verwendete gedimmte Miniaturlampe erzeugt in gegebenen Anordnung mit voll abgeblendeten Objektiv eine hellste Leuchtfläche von ca. 6 Pixel.

$${}^k n_{hell} \begin{cases} \leq C_3 = 8 & \text{sonst } \textsc{Warnung}. \\ \leq C_4 = 16 & \text{sonst } \textsc{Fehler}. \end{cases}$$

(Für den Demonstrationsfall, bei der Verwendung des manuellen Zielvorgabemodus wird C_3 und C_4 verdoppelt, da dann die Lampe etwas größer und heller ist).

6.3.2.3 Helligkeitsmaximum in Intervall: (Hell wie erwartet?)

$${}^k m_{hell} \in [C_5, C_6] \ , \ \text{sonst } \textsc{Warnung}.$$

Die Maximalhelligkeit wird größer als 60% Grauwert erwartet (die obere Schranke ist im Moment mit $C_6 = 255$ unwirksam). Die Kamerablenden werden so eingestellt, daß $\langle m_{hell} \rangle$ in diesem Intervall liegt.

6.3.2.4 Variation der Lampenhelligheit: (Hell wie sonst auch?)

Helligkeitschwankungen von mehr als $\pm 8\,\%$ Abweichung vom zeitlichen Mittel werden als suspekt betrachtet.

$$R_{Helligkeitfluktuation} := \frac{{}^k m_{hell}}{\langle {}^k m_{hell} \rangle} \in [C_7, C_8] \ , \ \text{sonst } \textsc{Warnung}. \tag{6.3}$$

6.3.2.5 Im Zweifelsfall Dunkeltest: (War das die Lampe?)

Im Falle eines FEHLERs wird der Versuch abgebrochen und ein neuer Zielpunkt gesucht. Führt eine der obigen Bedingungen zu einer WARNUNG, wird die Zeit investiert, ein Kontrollbild mit ausgeschalteter Lampe aufzunehmen.

$$^k m_{hell} - {}^k m_{dunkel} > C_9 \quad \text{sonst FEHLER.}$$

Der Test wird bestanden, wenn die maximale Bildhelligheit $^k m_{dunkel}$ nun um mindesten 7.8 % gesunken ist.

Der hier beschriebene Algorithmus hat sich in der Praxis als robust und verläßlich bewährt. Es wurde keinerlei irrtümliche Zielidentifikation beobachtet.

6.4 Anforderungen an das Kontrollprogramm

Beim Übergang von Computersimulation mit Graphikanimation zu realen Robotersteuerung müssen an das Steuerprogramm zusätzliche Anfordungen gestellt werden, die in den folgenden Abschnitten erläutert werden:

- Zuverlässiger Schutz gegen Kollisionen;
- Kontrollierte Reaktion auf Ausnahmezustände und Fehler;
- Wiederaufnahme eines unterbrochenen Lernvorganges;
- Kontrollierte Lernbedingungen;
- Fernüberwachbarkeit und Autonomie des Roboters;
- Interaktive Steuerungsmöglichkeit.

6.4.1 Kollisionssicherheit

Der PUMA 562 ist ein kräftiger, schneller und steifer Roboterarm. Das Programm muß daher zuverlässigen Schutz bieten gegen Kollisionen des Roboters sowohl mit Hinderissen als auch mit sich selbst. Wenn das Positioniervermögen des neuronalen Netzwerkes gering ist, wie zum Beispiel zu Anfang der Lernphase, kann es Bewegungskommandos ausgeben, die den Arm möglicherweise in kollisionsgefährdete Gebiete führen.

Um solche Fälle sicher zu handhaben, werden „verbotene Bereiche“ definiert und einem Kollisionsmonitor übergeben, anhand derer er die Zulässigkeit der aktuelle Bewegung überprüft.

Für die allgemeine Definition von Raumvolumina wurden die speziellen Objektstrukturen `OBSTACLE` und `OBSTACLE_LIST` entwickelt, die die kompakte Darstellung von verbotenen Bereichen in Anlehnung an sog. *Super-Quadriken* erlaubt. Die Idee ist, einen 3D Körper K als verallgemeinertes Ellipsoid durch

$$K = \left\{ (x_0, x_1, x_2)^T \in \mathbf{R}^3 \,\middle|\, \|\frac{x_0 - c_0}{a_0}\|^{p_0} + \|\frac{x_1 - c_1}{a_1}\|^{p_1} + \|\frac{x_2 - c_2}{a_2}\|^{p_2} \leq 1 \;, p_i > 0 \right\} \tag{6.4}$$

zu beschreiben. Für $p_i = 2$, $(i = 0, 1, 2)$ ist dies ein gewöhnliches Ellipsoid am Ort $\mathbf{c}$ mit den Halbachsen a_0, a_1 und a_3, welches mit steigendem p_i mehr und mehr zum Quader deformiert. Die Definition 6.4 wird nun erweitert, um auch Halbräume bequem einschließen zu können. Im Falle $p_i = 0$ wird der Term $t_i = |\frac{x_i - c_i}{a_i}|^{p_i}$ durch

$$t_i = \left\{ \begin{array}{c} 2 \\ 0 \end{array} \right\} \text{ für } \frac{x_i - c_i}{a_i} \left\{ \begin{array}{c} > \\ \leq \end{array} \right\} 0. \tag{6.5}$$

ersetzt, was für $a_i > 0$ den Halbraum $x_i > c_i$ als ***außerhalb*** von K definiert.

Ferner ist es praktisch, anstatt $\mathbf{a}$ den Vektor $\mathbf{r} = (r_0, r_1, r_2)$ zu spezifizieren, dessen Komponenten reziprok zu den Achsausdehnungen in Richtung i sind: $r_i = 1/a_i$.
Zur Verdeutlichung mögen folgende drei Beispiele dienen:

Elliptisches Prisma am Ort $\mathbf{c}$ mit den Halbachsen a_1, a_2 und der Länge L in x_0-Richtung:

$$\begin{array}{rcll} \mathbf{r} & = & (\frac{2}{L}, \frac{1}{a_1}, \frac{1}{a_2})^T & \text{reziproke Längen} \\ \mathbf{p} & = & (99, 2, 2)^T & \text{Potenzen} \end{array}$$

Potenz $p_0 = 99 < \infty$ ist aus numerischen Gründen reduziert. Die dabei entstehende Abrundung der Kanten ist aber in der Praxis unbedeutend.

Senkrecht nach unten reichender Halbzylinder am Ort $(0, 0, h)$ mit Radius r:

$$\begin{array}{rcll} \mathbf{c} & = & (0, 0, h)^T & \text{Halbachsenanfang} \\ \mathbf{r} & = & (\frac{1}{r}, \frac{1}{r}, 1)^T & \text{reziproke Längen, sign}(r_2) \text{ definiert Richtung} \\ \mathbf{p} & = & (2, 2, 0)^T & \text{Potenzen; Spezialfall } p_2 = 0 \end{array}$$

Halbraum über horizontaler Ebene in Höhe h:

$$\begin{array}{rcll} \mathbf{c} & = & (X, X, h)^T & \text{Höhe} \\ \mathbf{r} & = & (0, 0, -1)^T & \text{reziproke Längen; unendlich ausgedehnt =0} \\ \mathbf{p} & = & (X, X, 0)^T & X \text{ ist beliebig.} \end{array}$$

Ein Dauerhindernis ist z. B. der Tisch, auf dem der Roboter montiert ist. Dieser ist als Halbraum unter der Tischebene (plus Sicherheitsspanne) dargestellt. Ein weiteres Hindernis ist die Basis des PUMAs selbst, welche durch einen Halbzylinder beschreibbar ist. Gedrehte Körper werden durch eine vorhergehende Raumdrehung dargestellt, die in der Struktur ebenso definierbar ist, sowie zwei Hilfsfelder zur Identifikation.

Eine beliebige Anzahl von Körper kann nun kompakt in der Struktur `OBSTACLE_LIST` zusammengefaßt werden und an einen Kollisionsmonitor weitergegeben werden. Dieser Monitor prüft, ob eine Gelenkwinkelkonfiguration garantiert, daß sowohl die Handgelenksmitte als auch die Endeffektorspitze außerhalb der in `OBSTACLE_LIST` spezifizierten „verbotenen Bereiche“ zu liegen kommt. Aktiviert wird diese Überwachungsfunktion bei jeder Bewegung, sowohl beim Aufruf, als auch bei Zwischenschritten auf dem Weg zur neuen Position. Ein Eintreten in einen „verbotenen“ Bereich wird mit einer Fehlermeldung quittiert und die begonnene Bewegung wird abgebrochen.

6.4.2 Reaktion auf Ausnahmezustände und Fehler

Der vorhergehende Abschnitt beschrieb die Behandlung von „ordnungswidrigen“ Bewegungskommandos, die ein selbstlernender Algorithmus im untrainierten Stadium eventuell anfordert. Zusätzlich müssen eine Reihe potenieller Ausnahmezustände und Fehler des Robotersystems bedacht werden:

- NOT–AUS Schalter;
- Motor-Überlast;
- Überschreiten von erlaubten Gelenkwinkelgrenzen;
- Störungen der Kommunikation.

Diese Tests sind bereits in RCI / RCCL eingebaut und werden in *low-level* Routinen durchgeführt. In einem solchen Störfall wird der Roboter zuverlässig stillgelegt. Die Standardreaktion auf einen der obigen Zustände ist: Ausschalten der Antriebstromversorgung → Einlegen der Motorbremsen → Abbruch des Programmes.

6.4.3 Kontrollierte Lernbedingungen

Um ein Lernen mit irreführenden Informationen zu vermeiden, wird die Bildinformation auf Vollständigkeit und Konsistenz geprüft. Signalisiert eine der aufgerufenen Unterfunktionen Unstimmigkeit, z. B. wenn das Ziel nicht in beiden Kameras sichtbar ist (siehe 6.3), wird dieser Trainingsschritt verworfen und wiederholt. Alle Abnormalitäten werden in einem temporären Fehlermeldungsspeicher *(stack)* aufgenommen und nach mehr als drei aufeinanderfolgenden Fehlversuchen ausgegeben. Treten mehr als 12 Fehlversuche hintereinander auf, wird angenommen, daß ein schwerwiegender Defekt aufgetreten ist, und das Training wird abgebrochen.

6.4.4 Protokollierung und Wiederstartfähigkeit

Im Mittel benötigt das Programm ca. 8 Sekunden pro Trainingsschritt (mittlerer Geschwindigkeitswert, `speedfactor` = 5). Ein gesamter Lerndurchgang von 4000 Versuchen dauert über 9 Stunden und ist daher zeitlich relativ aufwendig. Für den Fall eines unvorhergesehenen Programmabbruchs ist es daher äußerst wünschenswert, die bereits erlangte „Erfahrung“ abzuspeichern und ein Wiederstarten mit verschiedenen Erfahrungszuständen möglich zu machen. Ein automatisches Abspeichern aller relevanten Daten ist in konstanten Lernabschnitten (typischerweise alle 25 Schritte) vorgesehen. Um auch Zwischenzustände zu dokumentieren (und restaurierbar zu halten), kann der Dateiname automatisch mitgeändert werden (optional).

Zusätzlich wird jeder Trainingsschritt in einer Log-Datei aufgezeichnet. Diese umfaßt: Lernschrittnummer, Positionierfehler nach Grob- und Feinbewegung, Zielabweichung (nach Feinbewegung) in kartesischen (xyz) und Kamerakoordinaten, sowie Index des ausgewählten Neurons. Zielabweichungen werden aus den Winkeltupeln anhand eines kinematischen Models gewonnen (siehe 7.5). Gemeinsam mit den *Restart*-Dateien wird eine ausführliche, nachträgliche Analyse des Lernprozesses ermöglicht.

Um Veränderungen in der Positionierung der Kameras oder der Endeffektorlampe nicht unbemerkt zu lassen, werden bei Programmstart Kamerakoordinaten von drei Referenzpunkten protokolliert. Auf Wunsch oder im Falle eines Wiederstartens werden Differenzen überprüft und aufgezeichnet.

6.4.5 Fernüberwachbarkeit

Wegen des bereits erwähnten hohen Zeitbedarfs der Lerndurchläufe war es ein Anliegen, das Kontrollprogramm so zu gestalten, daß das Training mit einem Minimum an Aufsicht auskommt, ohne dabei Sicherheitsaspekte zu kompromitieren. Jeder der oben angesprochenen Punkte leistete dazu einen wesentlichen Betrag. Auf Wunsch (*debug level* 0–6) – bzw. im Falle des Auftretens besonderer Abnormalitäten automatisch – gibt das Programm zahlreiche Diagnosemeldungen und gestattet damit eine gute Monitormöglichkeit von außerhalb des Labors. Zusätzlich besteht die Option, jederzeit ein Kamerabild in eine EPS[4]-Datei zu kodieren und von einem anderen (PostScript–fähigen) Computerbildschirm aus zu betrachten. Auf Tastendruck oder Prozeßsignal[5] wird ein Vollbild aufgenommen, kontrastverstärkt, verkleinert, entsprechend formatiert und verpackt als Datei ausgeschrieben. (Der überwiegende Teil, der im nächsten Kapitel präsentierten Daten resultiert aus derart ferngesteuerten Experimenten.)

6.4.6 Interaktive Steuerungsmöglichkeit

Mit der Tastenkombination *CRTL–C* (= signal `SIGINT`) springt das Programm nach Beendigung der momentanten Bewegung in einen Menümodus. Dies erlaubt, eine hierarchische Menüeingabe mit kontrollierten Eingabewerten bequem von jedem Terminal aus durchzuführen (keine Window-Umgebung nötig). Befehlskomplettierung mit *Space* oder *Return* und Auflistung von möglichen Submenüs mit '?' bzw. '*', sind eingebaut.

In der gegenwärtigen Version des Programmes `rob` sind unter anderem folgende Eingaben zulässig:

`set number_netfiles_to_change_name`: Bestimmt die Anzahl voll–dokumentierter Zwischenzustände;

`set speed %f`: Nominalgeschwindigkeitsfaktor 0.1–10;

`set camera_calibration`: Definiert neue Referenzpunktwerte;

`get_camera_calibration`: Überprüft Referenzpunktelagen;

`target manual_with_speed %f`: Manuelle Zielvorgabe;

`target auto`: Automatische Zielvorgabe mit "*Split–Brain*" Prozedur;

`histrograms %i`: Anzeige einer Serie von Grauwerthistogrammen (zur Kontrolle der Intensitätsverhältnisse von Objekt und Hintergrund);

`show parameter`: Auflistung einiger Parameter;

`show error_with_scale %f`: Anzeige der Lernkurve — gemittelt und ungemittelt;

`net_display <yes|no|color %i>`: Regelt die permanente Anzeige der Position der Gewichtsvektoren $\mathbf{w}_\mu$ sowie deren letzter Lernschritt (Erscheint auf den beiden Kontrollmonitoren);

---

[4] "Encapsulated PostScript" Graphiksprache von Adobe-Inc.

[5] z. B. von einem anderen Terminal aus mit `kill -SIGUSR2` oder `-SIGQUIT`

Lernparameter werden in einer separaten Parameterdatei definiert und beim Übersetzen eingebunden. Nichtstandard–Dateinamen von Protokoll-, Logdateien und ggf. *Restart*dateien sind als Option beim Programmaufruf zu spezifizieren[6].

6.5 Gewählte Parameter

Für das Experiment sind eine Reihe von Parameter noch unbestimmt, die in Kapitel 4 eingeführt wurden, und deren Initialisierung Gegenstand dieses Abschnittes ist. Für eine Liste der hier verwendeten Symbole siehe Seite 8.

Die Zeitabhängigkeit der Lernparameter $\lambda, \lambda', \epsilon$ wurde in der Form

$$p(t) = p_i \left(\frac{p_f}{p_i}\right)^{\frac{t}{t_{max}}} \qquad \text{mit} \quad p \in \{\lambda, \lambda', \epsilon\} \tag{6.6}$$

gewählt. Dies beschreibt ein exponentielles Abfallen des einzelnen Parameters vom Initialwert p_i zum Endwert p_f. Die Reichweiten der Nachbarschaftswechselwirkung fällt von den Initialwerten $\lambda_i = 150$, $\lambda'_i = 50$ zu den Endwerten $\lambda_f = \lambda'_f = 1$ ab. Die Lernschrittweite ϵ für die Referenzvektoren $\mathbf{w}_\mu$ verringert sich in derselben Weise von $\epsilon_i = 0.3$ zu $\epsilon_f = 0.05$. Die Adaptationsstärke $\epsilon' = 0.9$ für die Ausgabewerte (θ_μ und $\mathbf{A}_\mu$) bleibt konstant. $N = 300$ Neuronen waren am Lernprozeß beteiligt. Die obigen Werte wurden in vorausgehenden Computersimulationen von Martinetz bestimmt (siehe auch [35, 38] .

Die Referenzvektoren $\mathbf{w}_\mu$ sind zufällig aus dem insgesamt möglichen Eingangsraum der Kamerabildkoordinaten gewählt. Die Initialisierung der $\vec{\theta}_\mu$ Ausgabewerte erfolgte homogen verteilt im Raum der Drehgelenkswinkel:

$$\vec{\theta}_\mu \in [-130°, -50°] \otimes [-180°, -70°] \otimes [-40°, 70°] \;\cap\; \mathbf{R}^3_{FREI}\,, \qquad \forall\, \mu \tag{6.7}$$

Die Reihenfolge der Drehachsen folgt der Notation in Abbildung 5.2 (Seite 58). Dabei entspricht die tiefste Position des Oberarmes $(Gelenk2) = -180°$ der Horizontallage und $(Gelenk3) = 0°$ dem rechtwinklig nach unten gebeugten Ellenbogen (Ellenbogen-Oben Konfiguration), Stellungen mit gestrecktem Arm gehören damit gerade nicht mehr zum Initialisierungsbereich. Zudem werden Gelenkwinkelkonfigurationen $\vec{\theta}_\mu \notin \mathbf{R}^3_{FREI}$ unterdrückt, die in „verbotenes“, d. h. kollisionsgefährdetes Gebiet führen (siehe unten). Wie bereits erwähnt, ist das Handgelenk ruhiggestellt[7].

[6] Eine vollständige Liste erhält man bei Angabe einer undefinierten Option (z. B. '`rob --`').

[7] Achsstellung $(4) = 0°$, $(5) = -20°$, $(6) = 0°$. siehe [59, 6]

Die Anfangsbelegung aller Jakobimatrixelemente war

$$\mathbf{A}_{ij}\,| = \eta\;\frac{0.4\;rad}{512\ \text{Pixel}}\;, \qquad \forall\;\mu, i, j \tag{6.8}$$

mit η als homogen verteilter Zufallswert aus $[-1, 1]$[8]. Die Wahl dieser Intervalle ist unkritisch. Eine geeignete Voreinstellung sinnvoller Anfangswerte befähigt des Netzwerk dazu, besonders schnell zu guten Resultaten zu gelangen.

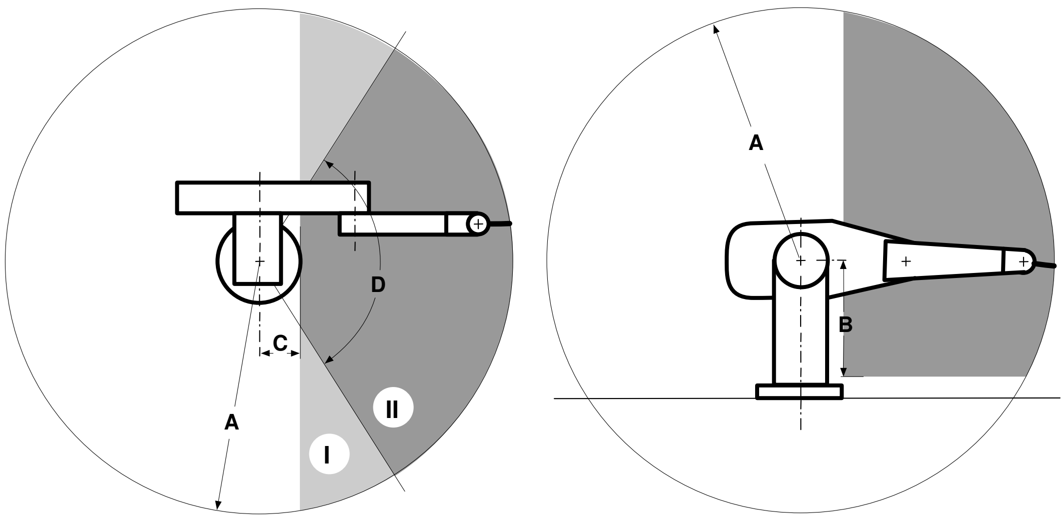


Abbildung 6.1: Die Grenzen des Arbeitsbereiches: Volumen *I+II* ist der „erlaubte“ Manöverraum ($= \mathbf{R}^3_{FREI}$), auf dessen Unterraum *II* sich die Vorgabe von Trainingszielen beschränkt. Die beiden Kameras blicken von rechts auf den Roboter. A = 1010 mm, A = 500 mm, C = 240 mm und D = 80°.

Die Grenzen des gewählten Arbeitsbereiches sind in Abbildung 6.1 als Auf- und Ansicht skizziert. Aus Gründen der Sicherheit (Gefahr der Kollision mit dem Tisch oder der Basis) und der guten Sichtbarkeit im Kameragesichtsfeld wurde das maximale Arbeitsvolumen des Puma auf den *erlaubten* Bereich $\mathbf{R}^3_{FREI}$ beschränkt, der in Abbildung 6.1 als Vereinigungsmenge des hell- und dunkelgrauen Bereiches *I* und *II* definiert ist. Bewegungsbefehle, die aus diesem Bereich führen würden, werden mit einer Warnung quittiert (siehe 6.4.1). Die tatsächliche Zielvorgabe beschränkt sich auf den Teilbereich *II* (in Abb. 6.1 dunkelgrau). Dabei wird ein Satz von Gelenkwinkeln homogen zufallsverteilt aus dem durch Relation (6.7) definierten Winkelraum ausgewählt.

Als „Ellenbogensingularität“ bezeichnet man die Ellenbogenstellung mit gestrecktem Arm. In ihrer Nähe bedarf eine kleine Armstreckung einer beliebig großen

[8] Dies bedeutet, daß im Extremfall einer Zielabweichung einer vollen Bildlänge (512 Pixel), eine Korrekturbewegung von maximal $0.4\,rad \simeq 23°$ ($|\eta| = 1$) erfolgt.

Winkeländerung im Ellenbogengelenk, was zum Divergieren der Jacobimatrix $\mathbf{A}$ führt. Damit können die Korrekturmatrizen $\mathbf{A}_\mu$ der für diese Raumbereiche zuständigen Neuronen nicht sinnvoll konvergieren, was in diesem Bereich in einem deutlich vergrößerten Positionierfehler (im Mittel bis zu versechsfacht) resultiert. Um das Erreichen dieser Randgebiete der Arbeitssphäre zu verhindern, wurde die Reichweite A um etwa 10 cm von der maximal möglichen Reichweite reduziert.

Kapitel 7

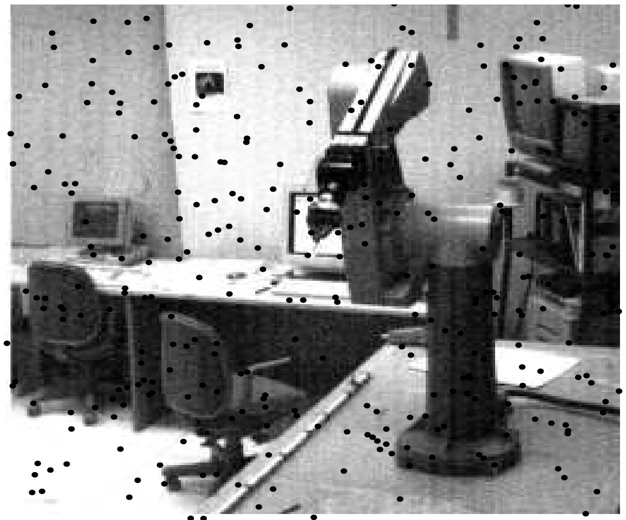
Experimentelle Ergebnisse und Diskussion

Der in Kapitel 4 vorgestellte visuo-motorische Koordinationsalgorithmus wurde im Roboterlabor getestet. Dieses Kapitel widmet sich der Darstellung und Diskussion der experimentellen Ergebnisse.

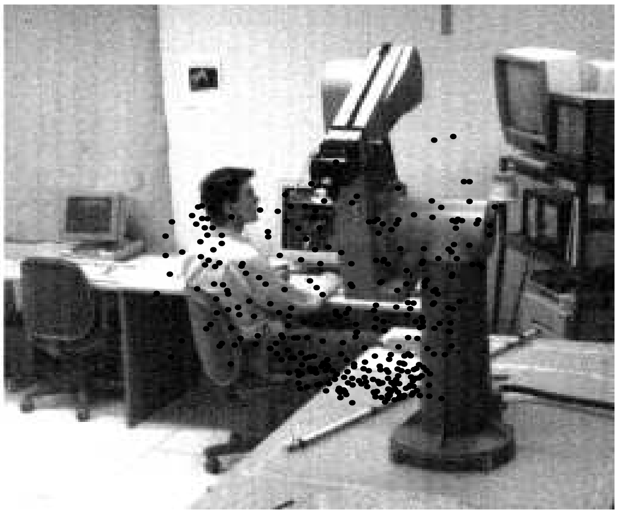
7.1 Die zeitliche Entwicklung der Neuronen

Abbildungen 7.1a-l (Seiten 93ff) zeigen eine typische Sequenz von Entwicklungszuständen des Neuronengases mit 300 Neuronen in einem Lernzyklus von $t_{max} = 4000$ Lernschritten. Da die Ausgabewerte der Neuronen schwierig zu visualisieren sind, beschränkt sich die Darstellung auf die Projektion der Referenzvektoren $\mathbf{w}_\mu$ auf die Bildebene der rechten Kamera.

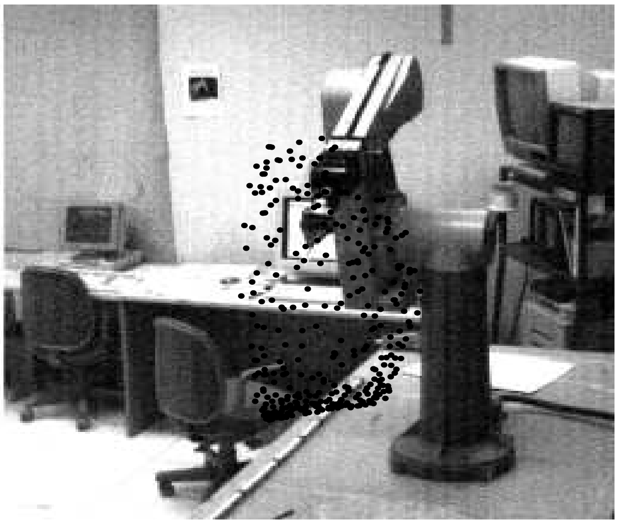
Abbildung 7.1: Nächste Seiten a-f, g-l: Zeitliche Entwicklung der Referenzvektoren $\mathbf{w}_\mu$ visualisiert als Projektionen *(schwarze Punkte)* auf die Bildebene der rechten Kamera. Abgebildet sind die Zustände des Netzwerks nach *(nächste Seite)* $t = 1$, $t = 25$, $t = 50$, $t = 100$, $t = 150$, $t = 300$, *(übernächste Seite)* $t = 500$, $t = 750$, $t = 1000$, $t = 1500$, $t = 2000$, und $t = 3000$ Lernschritten.



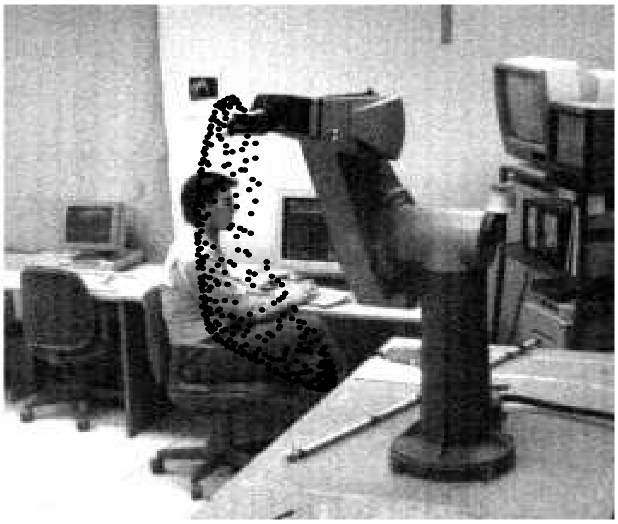
net.1.dat -- Kamera 1



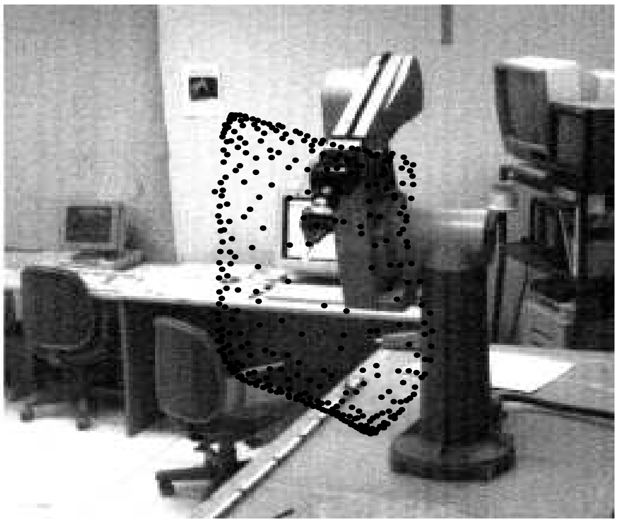
net.25.dat -- Kamera 1



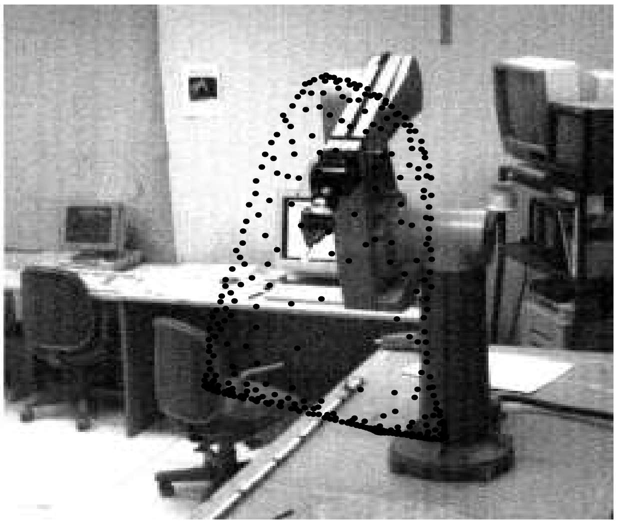
net.50.dat -- Kamera 1



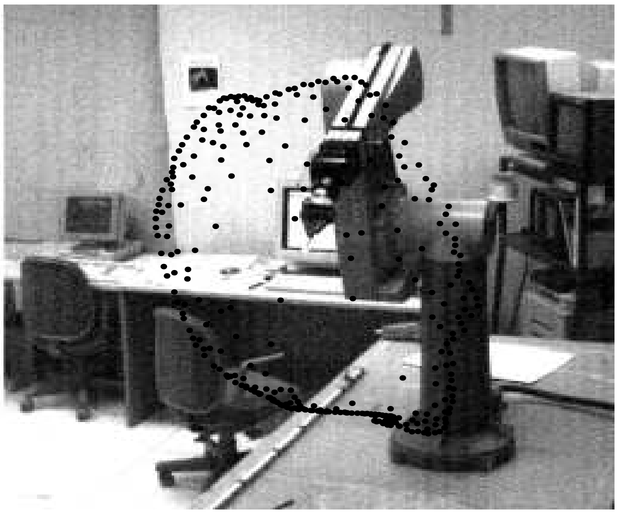
net.100.dat -- Kamera 1



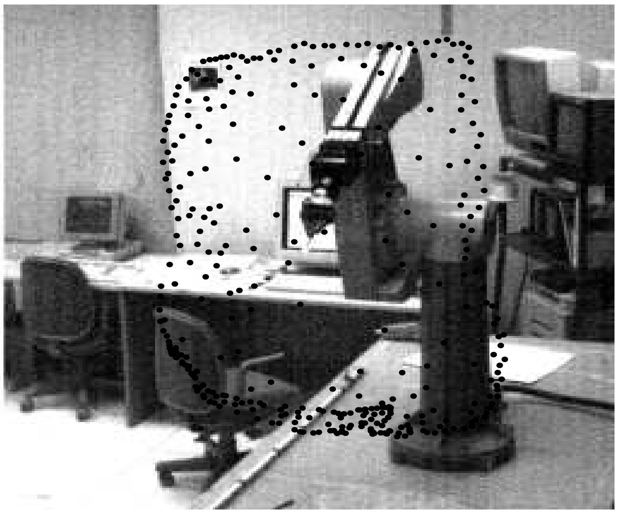
net.150.dat -- Kamera 1



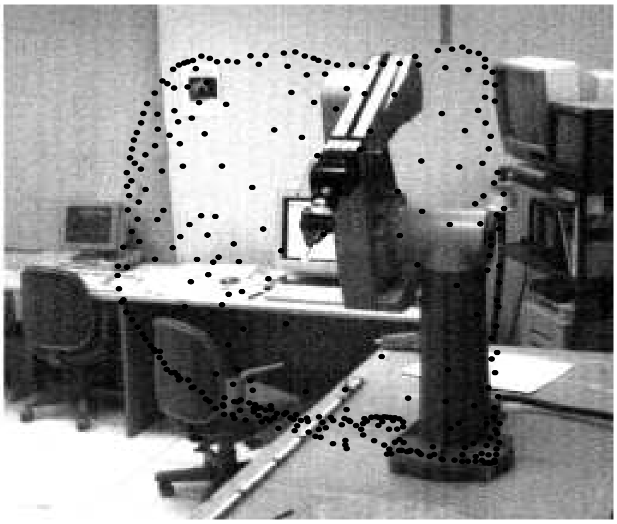
net.300.dat -- Kamera 1



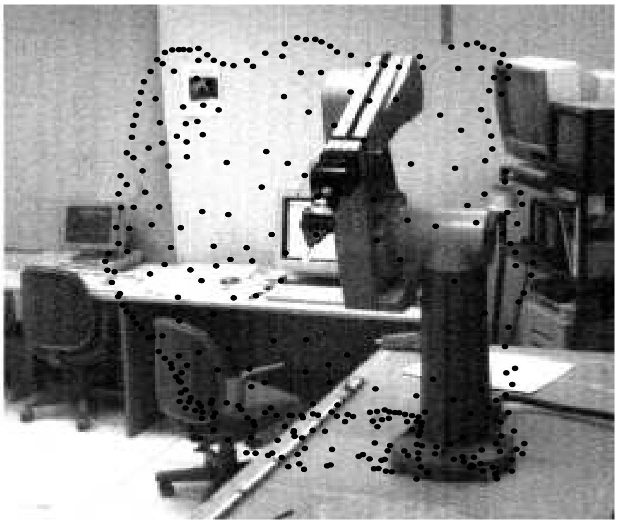
net.500.dat -- Kamera 1



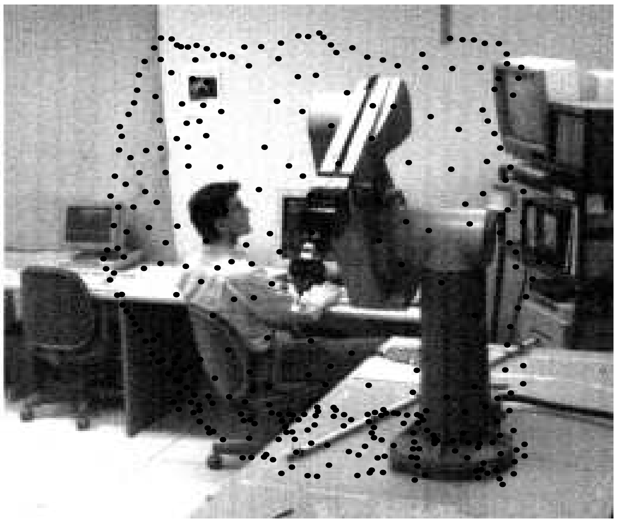
net.750.dat -- Kamera 1



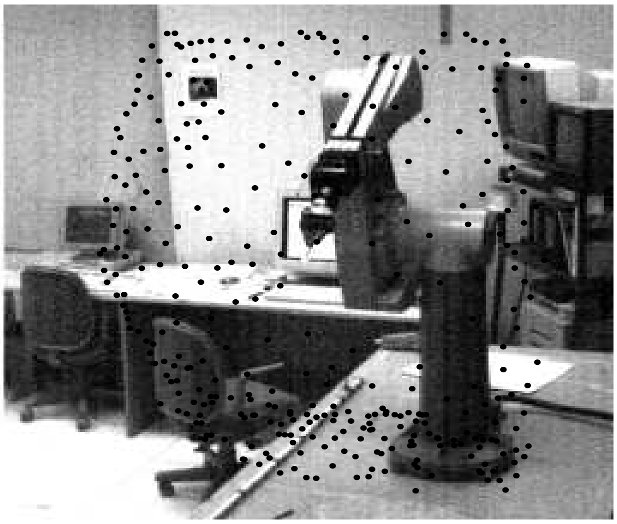
net.1000.dat -- Kamera 1



net.1500.dat -- Kamera 1



net.2000.dat -- Kamera 1



net.3000.dat -- Kamera 1

Wie man in der Bildsequenz deutlich erkennen kann, beginnt das Neuronengas seine Lernphase mit einer homogenen Zufallsverteilung („Sternenhimmel“ bei $t = 1$) und zieht sich aufgrund anfänglich hoher Plastizität (großes ϵ) und weitreichender Nachbarschaftswechselwirkung (λ) schnell auf eine kleine „Wolke“ um den Stimulationsschwerpunkt zusammen (siehe $t = 25$, 50, 100). Von dort streckt sich die Wolke, angetrieben von Zielvorgaben in den äußeren Bereichen der Arbeitssphäre, bis sie schließlich den ganzen Arbeitsbereich gleichmäßig abdeckt. Dabei kann man gut den allmählichen Übergang von anfangs schnellem, kollektivem Lernen zur asymptotischen Feinadaptation erkennen. Ein Vergleich des Zustandes nach $t = 3000$ Lernschritten mit dem Zustand am Ende der Lernphase nach $t_{max} = 4000$ in Abbildungen 7.2 zeigt kaum mehr eine Veränderung der Referenzvektoren $\mathbf{w}_\mu$.

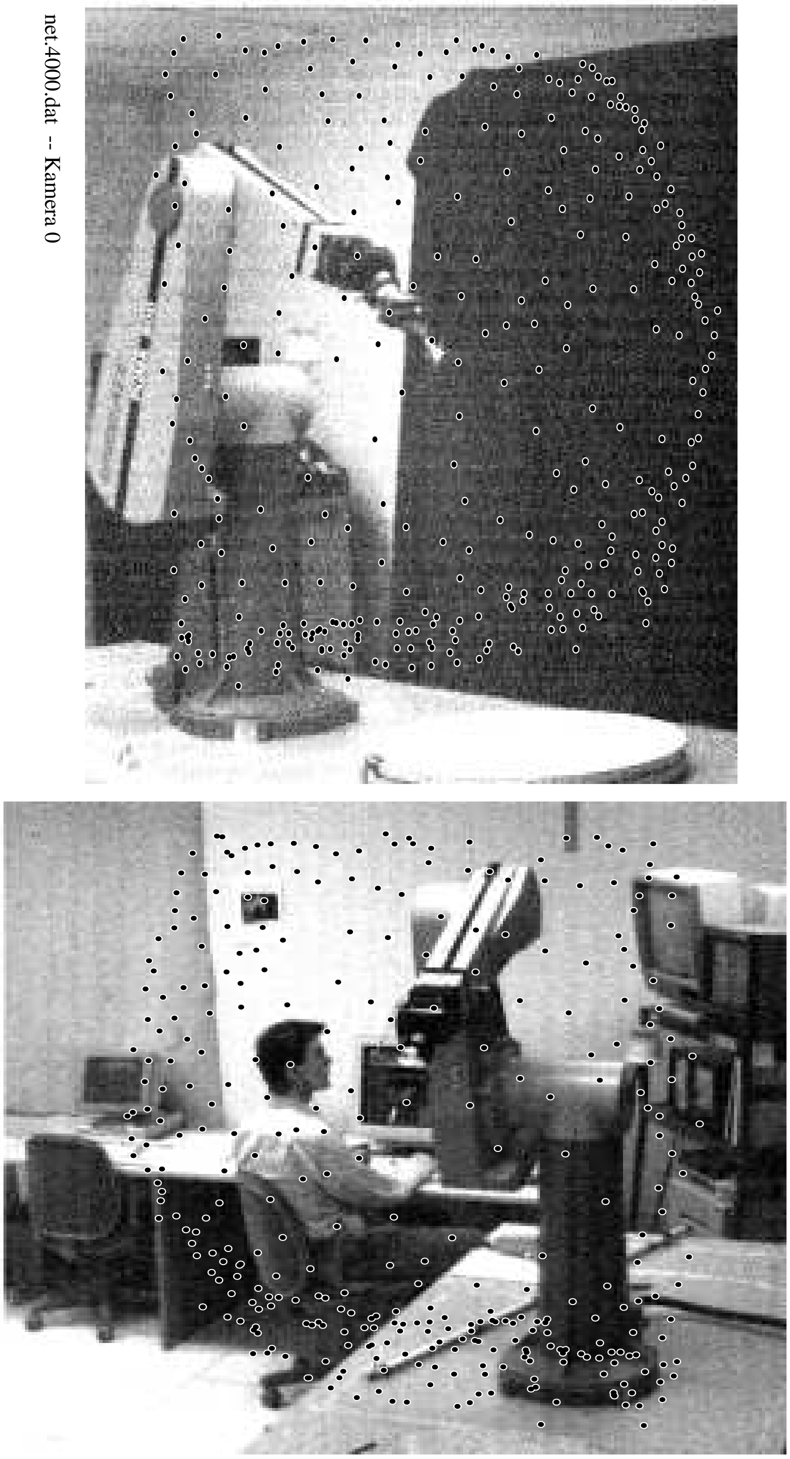
Der wesentliche Unterschied zu einer Implementation unter Zugrundelegung des Kohonennetzes liegt in der Verkürzung der Anfangsphase. Das Kohonennetz braucht Zeit, um sich bei beliebiger Zufallsinitialisierung der Referenzvektoren $\mathbf{w}_\mu$ zu entfalten und spätere Topologiedefekte zu vermeiden. Erst wenn sichergestellt ist, daß die Adaptionsverhältnisse der am Lernschritt beteiligten Neuronen deren Nachbarschaftsverhältnisse im Eingangsraum wiedergespiegeln, wird ein kooperatives Lernen sinnvoll und damit profitabel.

7.2 Die Lernkurve

Die Entwicklung der mittleren[1] Positioniergenauigkeit in Abhängigkeit der Anzahl der bereits erfahrenen Lernschritte zeigt Abbildung 7.3. Die obere, dünne Kurve zeigt das Positioniervermögen nach der Grobbewegung und die untere, dickere, mit „Feinbewegung“ markierte Kurve repräsentiert das Positioniervermögen nach dem Korrekturschritt. Der Vergleich beider Kurven macht den signifikanten Genauigkeitsgewinn deutlich, der durch das Ausnutzen der visuelle Rückkopplung (*feed back*) erreicht wird, und zeigt sehr schön den Unterschied zwischen einer offenen und einer (einmal) geschlossenen Kontrollschleife (*open/closed loop*). Nachdem der Fehler im Anfang der Lernphase rapide sinkt, erreicht des Netzwerk asymptotisch eine Positioniergenauigkeit von 2.2 mm.

[1] Gleitende Mittelwertbildung $\bar{F}(t) = \alpha F(t) + (1-\alpha)\bar{F}(t-1)$ mit $\alpha = 0.03$.

net.4000.dat -- Kamera 0



net.4000.dat -- Kamera 1

Abbildung 7.2: Die am Ende der Lernphase resultierende Verteilung der Referenzvektoren $\mathbf{w}_\mu$, projeziert auf die Bildebene der beiden Kameras *(oben und unten)*. Auf dem rollbaren Regalwagen *(im unteren Bild hinten rechts)* erkennt man die beiden Kontrollmonitore und darunter den Zentralrechner (teilweise vom Tisch verdeckt).

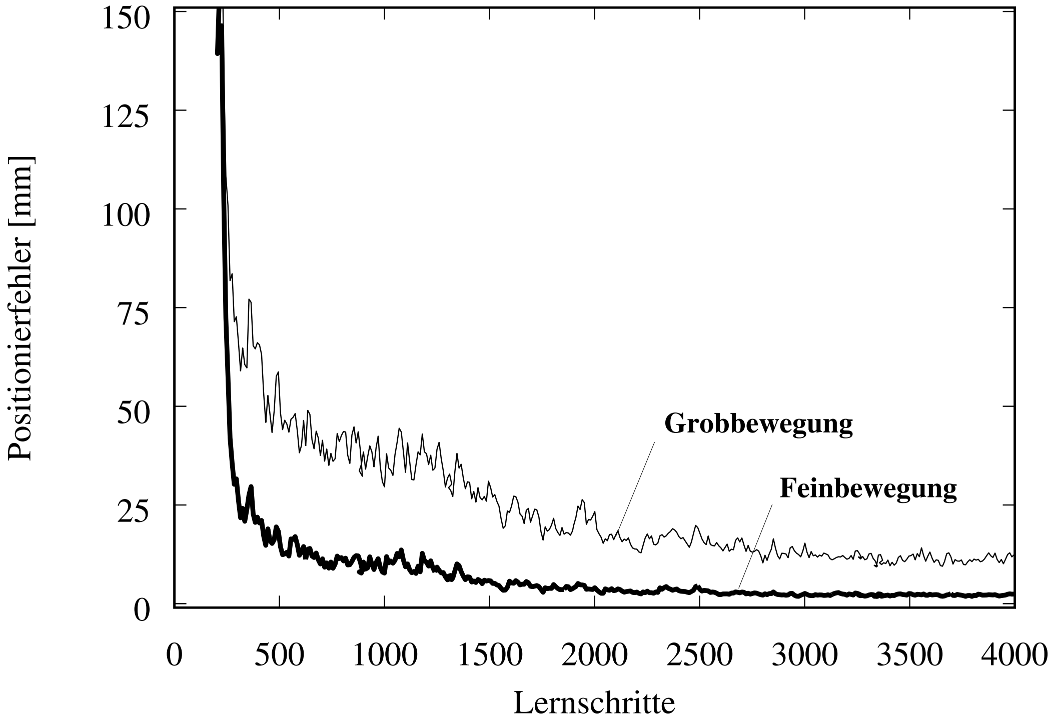


Abbildung 7.3: Die mittlere Abweichung des Endeffektors vom Zielpunkt in Abhängigkeit von der Zahl der Lernschritte. Die Korrekturbewegung aufgrund von visueller Rückkopplung verringert den Positionierfehler erheblich (Grobbewegung → Feinbewegung). Der durchschnittliche Restfehler nach 4000 Lernschritten liegt bei 2,2 mm.

Das Meßverfahren bestimmt den Euklischen Abstand zwischen dem gewünschten Zielpunkt und der erreichten Raumposition. Es macht sich dabei, wie in Abschnitt 7.5 näher erläutert, die robotereigenen Winkelmelder zu Nutze.

7.3 Anpassungsfähigkeit an veränderte Situationen

Ein entscheidender Vorteil von selbstlernenden Algorithmen ist zum einen ihre Fähigkeit zur Anpassung an verschiedene Umgebungen, *und* sich verändernde Umweltgegebenheiten. Der folgende Versuch soll dieses Adaptationsvermögen demonstrieren. Nach 3000 Lernschritten wird das Training unterbrochen und das letzte Armsegment um 100 mm verlängert ($\simeq 10\,\%$ des gesamten Arms).

Die Reaktion des Netzwerkes zeigt die Abbildung 7.4 in der zur vorherigen Abbildung analogen Weise. Nach der drastischen Geometrieänderung braucht das System nur 300 weitere Lernschritte um zu readaptieren und die frühere Präzision wiederzuerlangen. Auch hier wird nochmals der Einfluß der visuellen Rückkopplung im direkten Vergleich beider Kurven deutlich.

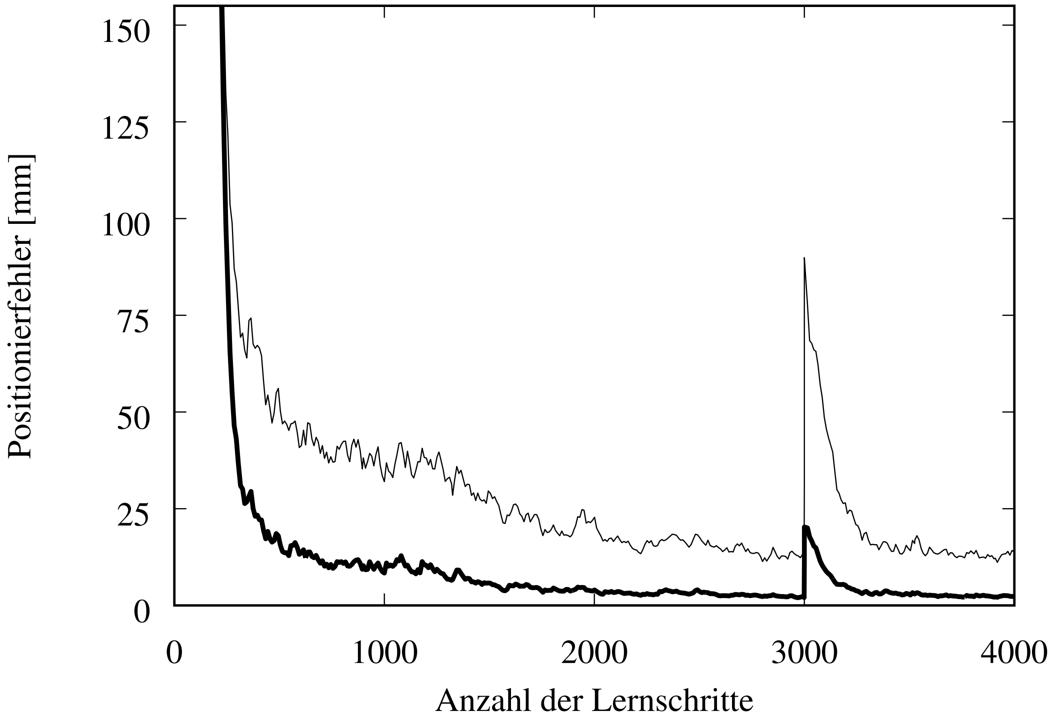


Abbildung 7.4: Der mittlere Positionierfehler *oben (dünne Linie)* ohne, und *unten (starke Linie)* mit visueller Rückkopplung. Nach 3000 Lernschritten wird der Arm um 100 mm ($\simeq 10\,\%$) verlängert. Schon nach 300 weiteren Trainingsschritten hat sich das Neuronengas fast vollständig der drastisch veränderten Situation angepaßt.

7.4 Der Einfluß der Nachbarschaftskooperation

Wir wollen nun die schon angedeutete Frage stellen, inwiefern kollektives Lernen (s. Seite 52) das Training der Positionierfähigkeit des Gesamtsystems verbessert. Um diese Frage zu beantworten, vergleichen wir zwei Lernzyklen, einen regulären Zyklus (entsprechend der Parameter in 6.5) und einen Zyklus mit einem Lernschema, das keine Nachbarneuronen am Lernen der Ausgangswerte teilnehmen läßt[2] ($\lambda' = 0$). Die Lernparameter ϵ, ϵ' und λ sind in beiden Lernzyklen identisch und erlauben in beiden Lernzyklen die Eingangsraumdiskretisierung (Referenzvektoren $\mathbf{w}_\mu$) durch kollektives

[2]Um $g'(\mu_0)$ nicht undefiniert zu lassen wird λ' tatsächlich zu $0 < \lambda' \ll 1/N$ gesetzt, was den beabsichtigten Effekt erzeugt (zur Klarheit $\lambda' = 0$ anstatt korrekterweise $\lim \lambda' \to 0$).

Lernen schnell zu erreichen. Beim Lernen der Ausgangswerte ist im zweiten Zyklus ($\lambda' = 0$) diese Kooperativität unterdrückt.

Solange die Positionierfähigkeit gering ist — wie zu Beginn der Lernphase, und, wie gezeigt wird, im zweiten Zyklus ohne Nachbarschaftskooperation auch bleibt — kann das Netzwerk Motorkommandos ausgeben, die den Roboterarm in verbotenes Gebiet schicken würden. Wie in Abschnitt 6.4.1 dargestellt, wird die Ausführung eines solchen Kommandos verhindert, mit der Konsequenz, daß kein absoluter Positionierfehler auftritt, den man sinnvoll mittlen könnte (vgl. Abb. 7.3 und 7.4). Um ein alternatives Leistungsmaß zu erhalten, das solche Fehlversuche mitberücksichtigt, betrachten wir nun die Wahrscheinlichkeit $P_t(R)$ eine vorgegebene Fehlertoleranz R zur Zeit t einzuhalten. $P_t(R)$ kann als

$$P_t(R) = \int_0^R p_t(r)dr, \qquad \text{mit } r \text{ als Euklidische Zielabweichung,} \tag{7.1}$$

geschrieben werden, wobei $p_t(r)dr$ die Wahrscheinlichkeit ist für einen Versuch zur Zeit t den Positionierfehler im Intervall $[r, r + dr]$ zu finden.

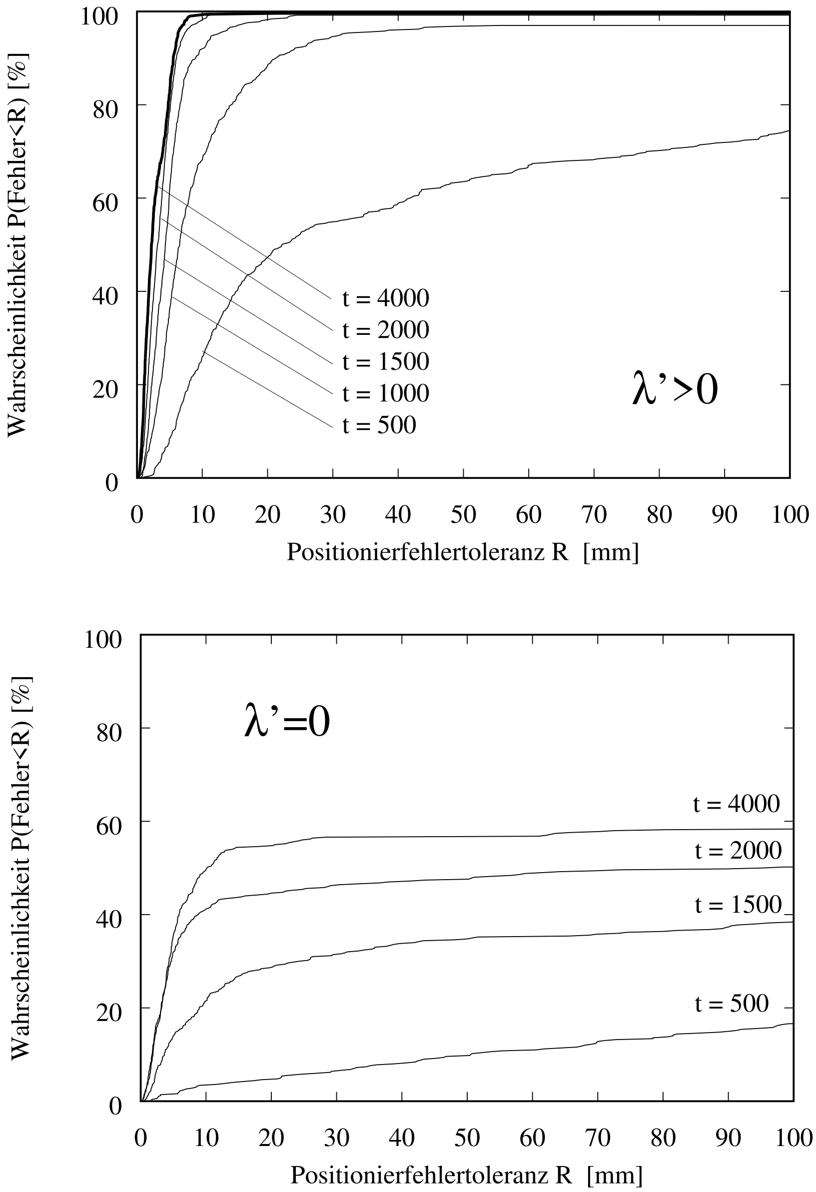


Abbildung 7.5: Effekt der Nachbarschaftkooperation auf die Positionierfähigkeit in verschiedenen Lernphasen. Die beiden Graphen zeigen die Wahrscheinlichkeit $P_t(R)$ eine bestimmte Zielfehlertoleranz R einzuhalten. *Oben:* Mit Nachbarschaftskooperation ($\lambda' > 0$) steigt die Positionierfähigkeit zunächst sehr schnell — dann asymptotisch langsam mit wachsender Zahl von Trainingsschritten t an. *Unten:* Bei *un*kooperativem Lernen der Ausgabewerte ($\lambda' = 0$) ist die Positionierfähigkeit deutlich geringer. Nach 4000 Lernschritten mißlingt noch jeder dritte Versuch, das Ziel zu erreichen.

Der untere Graph zeigt das System von Neuronen, deren Ausgabe nicht-kollektiv gelernt wird ($\lambda' = 0$). Das Positioniervermögen, das hier zu den Zeiten $t = 500$, $t = 1000$, $t = 2000$ und $t = t_{max} = 4000$ gezeigt ist, wächst relativ langsam. Das resultierende Positioniervermögen ist ungenügend, etwa jeder dritte Versuch führt zu einer groben Fehlpositionierung.

Der obere Graph zeigt ein System von Neuronen mit kollektivem Nachbarschaftslernen. Schon nach ein paar hundert Trainingsschritten zeigt es bessere Resultate als das andere System jemals erreicht. Die Ergebnisse für $t = 500$, $t = 1000$, $t = 1500$, $t = 2000$ und $t = t_{max} = 4000$ demonstrieren, wie das Positioniervermögen anfänglich sehr schnell wächst und asymptotisch sehr genaue Ergebnisse erzielt.

7.5 Die erreichte Positioniergenauigkeit

In diesem Abschnitt soll die erreichte Positioniergenauigkeit des Roboters diskutiert werden.

Im beschriebenen Experiment wurde nach dem Abschluß der Lernphase ein gemittelter Positionierfehler von 2.2 mm mit jeweils nur einer Korrekturbewegung erreicht. Dies läßt sich zu den Ausmaßen des Arbeitsbereiches oder denen des Roboterarmes in Relation setzen. Nehmen wir als Vergleichswert 1.0 m (das Trainingsvolumen ist etwas mehr als ein Kubikmeter), so ergibt dies eine relative Genauigkeit von $2.2 \cdot 10^{-3}$.

Dies ist um mehr als eine Größenordnung genauer als die Genauigkeit, die Kuperstein in seinen Arbeiten zur visuo-motorischen Kontrolle [28, 29] erzielt. Kuperstein [28, 29] verwendet einen ähnlichen Ansatz, der drei separate topographische Karten ausnutzt, um die visuelle Information der linken und rechten Kamera abzubilden. In der dritten Karte wird die Disparitätsinformation kodiert. Für jeden Armfreiheitsgrad steht eine eigene Ausgangszellenpopulation zur Verfügung, die dann die Bewegung bestimmt. Mittels eines geeigneten Lernverfahrens erzielt Kuperstein eine mittlere Positioniergenauigkeit von 4 %.

Die folgenden Einflußfaktoren auf den Positionierfehler kommen in Betracht:

1. Bildverarbeitung; Verzerrungen des optisches Systems, Pixelauflösung, ungenaue oder fehlerhafte Bildverarbeitung.

2. Neuronaler Kontrollalgorithmus; Restfehler durch lineare Approximation an endlich vielen diskreten Stützstellen.

3. Positionierwiederholgenauigkeit des Roboterarmes. (Gleiche $\vec{\theta}$ führen nicht an die selbe Raumposition; technische Daten s. S. 61).

4. Genauigkeit des Meßverfahrens zur Bestimmung des Fehlers.

Zunächst ist festzustellen, daß systematische Fehler der einzelnen Komponenten durch den adaptiven Kontrollalgorithmus kompensiert werden. Diese Eigenschaft ist eines der Hauptmerkmale des Verfahrens. Damit entfällt die Einflußmöglichkeit von Fehlern wie zum Beispiel Verzerrungen durch das optische System, Mißkalibrierung der Kameras oder des Roboters. Die von Martinetz erstellten Computersimulationen [38] mit 300 Neuronen zeigen für den Idealfall einen Restpositionierfehler von $0.8 \cdot 10^{-3}$. Dies bedeutet, daß in der Praxis, sofern nicht Konvergenzschwierigkeiten auftreten, noch „Kapazitätsreserven“ bleiben müßten. Die Lernkurve (Abbildung 7.3) legt nahe, daß dies nicht der Fall ist und der Hauptfehlerbeitrag daher anderweitig zu suchen ist.

Die Meßmethode des Positionierfehlers läßt sich durch die Wahl des *„Splitt Brain“*-- Lernverfahrens auf Differenzen von Endeffektorpositionsmessungen zurückführen. Die Endeffektorposition wird jeweils aus den eingebauten Winkelencodern mittels eines kinematischen Models bestimmt. Dies gelingt mit ausreichender Exaktheit, da die geometrischen Daten des Puma wohlbekannt sind und, da keine kurzzeitigen Lastwechsel auftreten, sich nicht signifikant verändern. Die 14-bit optischen Encoder wurden in einer Eichprozedur kalibriert (siehe 5.4.4.2). Somit ist der Fehlerbeitrag der Bestimmung benachbarter Raumpositionen zunächst vernachlässigbar. Nicht so der Einfluß des Getriebespiels, was die Liste möglicher Hauptverursacher auf Punkt 1 und 3 einschränkt.

Um die anderen Einflußfaktoren zu untersuchen, wurde das Bildvorverarbeitungssystem und zugleich die Wiederholgenauigkeit des Roboters mit einer Reihe von Endeffektorbewegungen getestet.

Die Testbewegungen des Endeffektors in x-, y- und z-Richtung sind rechts oben in Abbildung 7.6 als Dreibein illustriert. Aus den beiden Kamerabildern werden die Endeffektorkoordinaten u_1, u_2, u_3 und u_4 extrahiert. Sie werden für die drei orthogonalen Bewegungen in drei separaten Graphen mit willkürlich verschobenen Nullagen aufgezeichnet (Reihenfolge dem Dreibein entsprechend). Zunächst erkennt man die Treppenstufenstruktur, die von dem gewählten Bildextraktionsverfahren herrührt und die resultierenden Objektkoordinaten in 0.5 Pixelwerten diskretisiert (siehe 6.3). Die wiederholte Vorwärts- und Rückwärtsbewegung erzeugt verbreiterte „Flankenzonen“[3], in denen ein aufgezeichneter Raumpunkt nicht völlig mit den extrahierten Bildpunkten korrelliert. Ursache ist hier eine Mischung von Positionierfehlern und Helligkeitsfluktuationen. Das wesentliche Resultat jedoch ist, daß die Flankenzonen

[3]Als Flankenzonen sind die Bereiche gemeint, in denen die Kurve von einem Plateau zum nächsten springt.

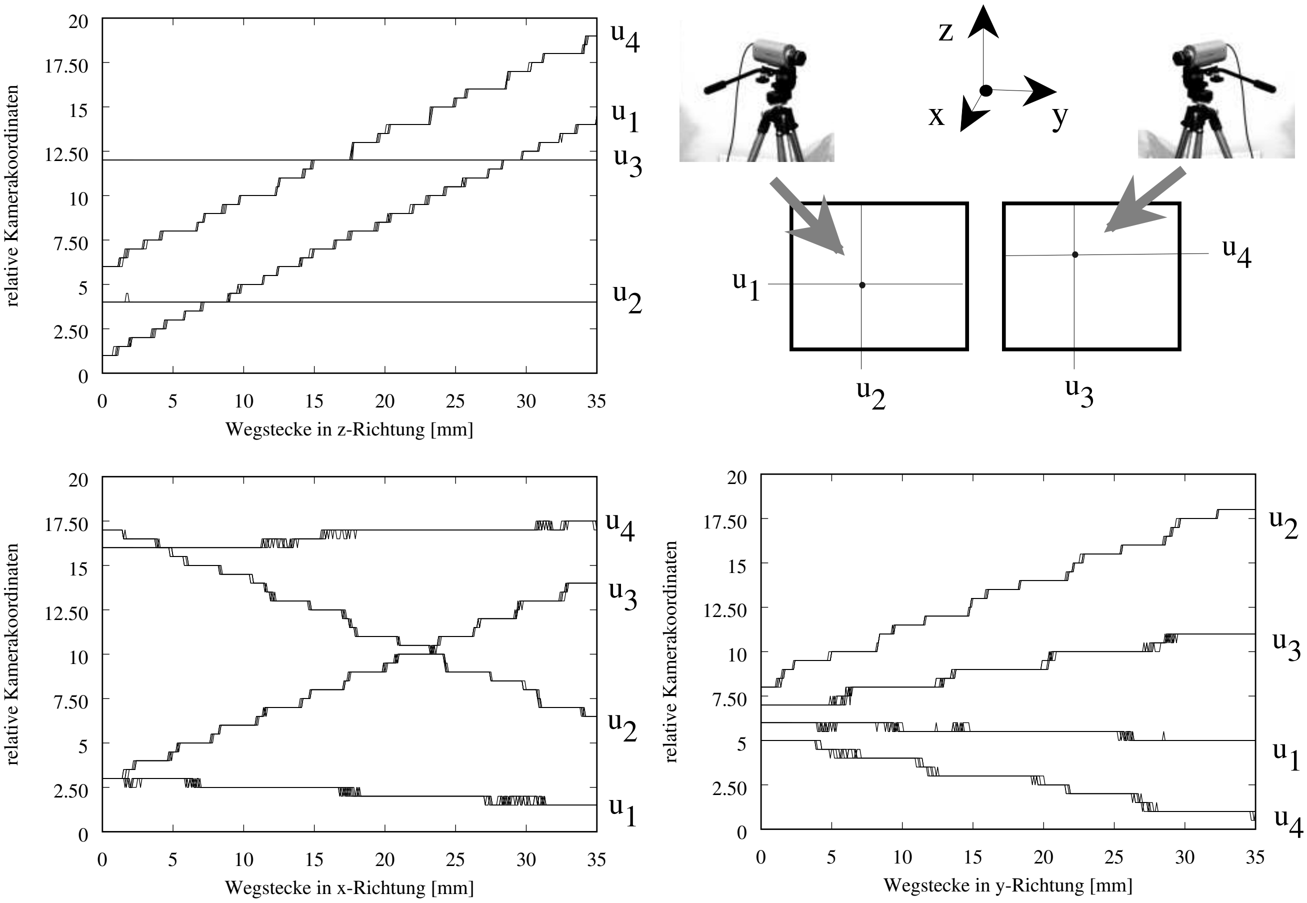


Abbildung 7.6: *(Rechts oben:)* Illustration der orthogonalen Bewegungen des Endeffektors und der aus den beiden Kamerabildern extrahierten Objektkoordinaten u_1, u_2 (linkes Kamerabild), und u_3, u_4 (rechtes Kamerabild). Die aus den Testbewegungen in x, y und z-Richtung resultierenden Kamerakoordinaten sind in den drei Graphen in dem Dreibein ensprechender Reihenfolge aufgezeichnet. Die Plateaus entstehen durch die Diskretisierung in 0.5 Pixel und die verbreiterten Flanken durch das fünfmalige Durchfahren der Wegintervalle (vor- und rückwärts). Aus dem Vergleich der Plateaubreite zur Flankenbreite kann man schließen, daß die Wiederholgenauigkeit des Roboterarmes besser ist als die Auflösung der Bildverarbeitung.

signifikant schmaler sind als die Plateaus. Dies identifiziert den *Bildverarbeitungsteil* als den im beschrieben Aufbau für die Genauigkeit des Gesamtsystems limitierenden Faktor.

Zusätzlich ergibt eine Analyse der jeweils steilsten (informationsreichsten) Kurve je Testrichtung eine Steigung von $\{3.0, 3.5, 2.5\}\,\frac{mm}{pixel}$ in $\{x, y, z\}$ Richtung (u_3, u_2, u_1). Daß heißt, in der Richtung der Winkelhalbierenden der beiden Kamerablickrichtungen, welche die am schlechtesten aufgelöste ist, ergibt sich der grob abgeschätzte Auflösungsfehler von $3.5\,\frac{mm}{pixel}\,0.5\,pixel\ =\ 1.8\,mm$. Dies enspricht ungefähr dem im Experiment gefundenen Positionierfehler.

Als Möglichkeiten zur Verbesserung der Genauigkeit, bieten sich an:

- Verwendung höher auflösender Kameras (relativ teure Lösung, da man damit den Bereich der Standardvideotechnik verläßt);
- Zusätzliche Kameras; die präzisionsrelevanten Bereiche könnten problemlos mit zusätzlichen stationären Kameras abgedeckt werden; die nötigen Algorithmusänderungen sind relativ simpel und würden auf eine dynamische Festlegung der Eingaberaumdimension der Kamerabildpositionsvektoren $\mathbf{u}_{target}$ und $\mathbf{v}$ hinauslaufen; auch eine am (z. B.) Unterarm montierte, mitbewegte Kamera wäre gut vorstellbar;
- Genauigkeitsgewinn durch Analyse der Grauwertbildinformation; man könnte die Kameraobjektive leicht defokusieren, so daß eine Punktlichtquelle eine glockenförmige Helligkeitsverteilung erzeugen würde; unter Berücksichtigung möglicher Übersteuerung (Glockenspitze abgeschnitten) lassen sich aus den Anstiegsflanken der Glockenkurven die gewünschten Bildkoordinaten in Subpixelauflösung bestimmen.

Der letzte Punkt ist sicherlich der Nächstliegende, um die Positioniergenaugkeit im momentanen Aufbau zu erhöhen. Schwierigkeiten könnten große Helligkeitsgradienten des Szenenhintergrundes bereiten, sofern sie nicht im Programm berücksichtig sind. Der Vorteil sind die nicht anfallenden Kosten für zusätzliche Hardware, der Preis ist eine Vermehrung der Rechenlast.

Kapitel 8

Zusammenfassung

Zwei Anwendungen für selbstlernende, neuronale Algorithmen standen im Vordergrund dieser Arbeit. Im ersten Teil wurde ein neuer, im Rahmen dieser Arbeit entstandener Ansatz für die Vorhersage von nicht-linearen Zeitserien untersucht. Die Idee besteht darin, einen Kohonen-Algorithmus bzw. einen „Neuronengas"-Algorithmus zu verwenden, um den Eingangsraum adaptiv zu diskretisieren. In jeder der dabei entstehenden Diskretisierungszellen wird ein separater Satz von linearen Prädiktionskoeffizienten angepaßt, mit deren Hilfe ein zukünftiger Datenpunkt vorhergesagt wird. Die Methode wurde sowohl an Teilchentrajektorien in einer Reihe von Potentialen als auch anhand einer *chaotischen* Zeitserie, der sog. Mackey–Glass Zeitserie, getestet. Hierbei wurde eine gute Vorhersagegüte schon für relativ kleine Netzwerke gefunden.

Der zweite Teil beschreibt die Implementierung eines visuo-motorischen Kontrollalgorithmus zur Endeffektorpositionierung eines Robotersystems. Ein PUMA 562, ein in der industriellen Fertigung häufig eingesetzter Roboterarm, lernte, ohne jeglichen Lehrer, durch bloßes Üben mit Hilfe von visueller Rückkopplung seinen Endeffektor richtig zu positionieren. Es wurde eine Positioniergenauigkeit von 2.2 mm, d. h. 0.2 % der linearen Abmessungen des Roboterarmes, erzielt. Trotz der geringen Anzahl von insgesamt 300 Neuronen, die für den 3-dimensionalen Arbeitsbereich zuständig waren (d. h. pro Dimension nur 6.7 Neuronen), war die Genauigkeit des Aufbaus durch die Bildvorverarbeitung und *nicht* durch den Kontrollalgorithmus begrenzt. Mittels Kamerarückkopplung gelang es dem System darüberhinaus, sich rasch an sowohl langsame als auch abrupte Änderungen seiner Umweltbedingungen anzupassen.

Ein wesentlicher Faktor für das schnelle Lernen ist ein kollektive Zusammenwirken von Neuronen, wobei jeweils eine Teilmenge der Neuronen am Lernschritt teilnimmt. Diese Teilmenge ergibt sich im Kohonenmodell aus der Nachbarschaftsrelation der Neuronen im *vorstrukturierten Gitter*. Im Gegensatz dazu wird im „Neuronengas"-Modell diese Teilmenge aus den Nachbarschaftsrelationen der *Referenzvektoren* im Eingangsraum bestimmt. Es zeigte sich, daß dieses kollektive Lernen die Leistungsfähigkeit der neuronalen Algorithmen sowohl in Bezug auf eine Robustheit gegenüber Anfangsbedingungen als auch in Bezug auf die Lerngeschwindigkeit stark verbessert.

Häufig verwendete Formelzeichen

E_{cost}	Kostenfunktion	$(S.\,11)$
$\langle x \rangle_{t'}$	Zeitmittelung des Argumentes $x(t')$	$(S.\,14)$
$P(\,)$	Wahrscheinlichkeitsverteilung des Argumentes ( )	$(S.\,11)$
F_{norm}	Normalisierter Vorhersagefehler	$(S.\,38)$
N	Anzahl der Neuronen	$(S.\,17)$
t	Zeit und auch Anzahl der Lernschritte (wenn $\Delta t = 1$)	$(S.\,10)$
$X \subseteq \mathbf{R}^m$	Eingangsraum	$(S.\,8)$
$\mathbf{w} \in \mathbf{R}^m$	Referenzvektor eines Neurons	$(S.\,8)$
m	Dimension des Eingaberaums	$(S.\,8)$
W	Menge der Referenzvektoren $\mathbf{w}$ aller Neuronen	$(S.\,10)$
$A \subseteq \mathbf{R}^n$	Ausgaberaum der Neuronen oder auch $A \subseteq \mathbf{R}^n \times \mathbf{R}^m$	$(S.\,12)$
n	Dimension des Ausgaberaums	$(S.\,12)$
$\mathbf{a} \in A$	Ausgabewert eines Neurons	$(S.\,12)$
$\mathbf{r}$	Index für Neuron am Gitterplatz $\mathbf{r}$	$(S.\,8)$
$\mathbf{s}$	Index des „Sieger"-Neurons, dessen Referenzvektor $\mathbf{w_s}$ am nächsten zum Eingangsvektor $\mathbf{x}_{stim}$ ist	$(S.\,8)$
ϑ_W	durch W bestimmte Abbildung $\vartheta_\mathbf{W} : \mathbf{x}_{stim} \mapsto \mathbf{s} = \vartheta(\mathbf{x}_{stim})$	$(S.\,8)$
$\Phi_{\vartheta\mathbf{a}}$	durch $\{\mathbf{w_r}\}$ und $\{\mathbf{a_r}\}$ bestimmte Abbildung $\Phi_{\mathbf{W},\mathbf{a}} : X \mapsto A$	$(S.\,12)$
μ	Index eines Neurons im „Gas" (entspricht $\mathbf{r}$ im Kohonenmodell)	$(S.\,17)$
μ_k	das Neuron mit dem insgesamt k-kleinsten $\lvert\lvert \mathbf{x}_{stim} - \mathbf{w}_\mu \rvert\rvert$ Wert; (μ_0 entspricht $\mathbf{s}$ im Kohonenmodell)	$(S.\,17)$
$\varepsilon^*(t)$	Schrittweite des t-ten Lernschrittes ($* \in \{(),(')\}$ siehe unten)	$(S.\,9)$
$h^*_{\mathbf{rs}}$	laterale Wechselwirkungsfunktionen im Kohonengitter	$(S.\,9)$
$\sigma^*(t)$	Reichweite von $h^*_{\mathbf{rs}}$ beim t-ten Lernschritt	$(S.\,10)$
$g^*(\mu)$	Wechselwirkungsfunktionen im „Neuronengas"	$(S.\,18)$
$\lambda^*(t)$	Reichweite von $g^*(\mu)$ beim t-ten Lernschritt	$(S.\,19)$
$* = ()$	$\varepsilon, h_{\mathbf{rs}}, \sigma, g(\mu), \lambda$ beziehen sich auf den Lernschritt der Referenzvektoren $\mathbf{w_r}$	$(S.\,18)$
$* = (')$	$\varepsilon', h'_{\mathbf{rs}}, \sigma', g'(\mu), \lambda'$ beziehen sich auf den Lernschritt der Ausgabewerte (z. B.) $\mathbf{a_r}$	$(S.\,18)$
$g^{mix}(\mu), \lambda^{mix}$	Gewichtungsfunktion g^{mix} für die (optionale) Mittelung des Ausgabewertes mit der Reichweite λ^{mix}	$(S.\,18)$

	in Kapitel 3	($S.$ 23)
$x(t)$	Zeitserienamplitude	($S.$ 23)
Δt	Diskretisierungszeit für eine Zeitserie	($S.$ 25)
l	Anzahl der Lernschritte (wenn $\Delta t \neq 1$)	($S.$ 10)
T	Vorhersagezeit in die Zukunft	($S.$ 27)
$\mathbf{x}_{state}(t)$	Vektor im Zustandsraum von $x(t)$	($S.$ 25)
$\mathbf{x}_{stim} \in X$	Eingabevektor für das neuronale Netzwerk	($S.$ 32)
$\mathcal{K}_i$	optionale Skalierungskonstanten für die X_{stim} Repräsentation	($S.$ 32)
$\mathbf{y}(\mathbf{x})$	zu lernender Funktionswert	($S.$ 13)
$\mathbf{x}_{hist} \in \mathbf{R}^n$	Vektor in einem Zustandsraum von $x(t)$	($S.$ 31)
d	Dimension der Eingangsraummannigfaltigkeit (Attraktor)	($S.$ 25)
c	Ordnung des 3D-Potentials	($S.$ 30)
τ	Verzögerungszeitparameter in der Mackey–Glass Gleichung	($S.$ 37)

	in Teil II	($S.$ 49)
$\mathbf{u}_{target} \in \mathbf{R}^4$	Zielposition in Kamerakoordinaten	($S.$ 50)
$\vec{\theta} \in \mathbf{R}^3$	Satz von Gelenkwinkeln	($S.$ 51)
$\mathbf{A} \in \mathbf{R}^3 \times \mathbf{R}^4$	Jakobimatrix	($S.$ 51)
$\vec{\theta}_\mu, \mathbf{A}_\mu$	Ausgabewerte des Neurons μ	($S.$ 51)
$\mathbf{v}_{initial} \in \mathbf{R}^4$	Endeffektorposition in Kamerakoordinaten nach „Grobbewegung" des Roboterarmes	($S.$ 51)
$\mathbf{v}_{final} \in \mathbf{R}^4$	Endeffektorposition in Kamerakoordinaten nach Korrekturbewegung („Feinbewegung") des Roboterarmes	($S.$ 51)

Danksagung

Die vorliegende Arbeit entstand in der Arbeitsgruppe *Theoretical Biophysics* von Professor Klaus Schulten am *Beckman Institute* und *Department of Physics* der University of Illinois at Urbana–Champaign, Illinois, USA.

Ich möchte Herrn Schulten ganz herzlich, für die ausgesprochen interessante und interdisziplinäre Aufgabenstellung, sowie die hervorragenden und anregenden Arbeitsbedingungen danken.Auch für das hohe Maß an Vertrauen, das mir Herr Schulten entgegenbrachte, als er die überdurchschnittlich hohen finanziellen Mittel zur Verfügung stellte, möchte ich ihm danken. Dieses Vertrauen und seine stete Zuversicht über das Gelingen des Projektes waren wichtiger Ansporn und Motivation.

Professor Helge Ritter möchte ich ganz besonders für seine herzliche und motivierende Betreuung, insbesondere in der Anfangsphases dieser Arbeit danken. Er hatte stets ein offenes Ohr für alle anstehenden Sorgen und Probleme, und wurde nie müde meine vielen Fragen als Neuling auf dem faszinierenden Gebiet der Neuroinformatik zu beantworten.

Thomas Martinetz danke ich für seine Vorarbeiten zum visuo-motorischen Kontrollalgorithmus. Ohne seine ausführlichen Computersimulationen wäre eine schnelle Implementierung des Netzwerkalgorithmus zur Robotersteuerung nicht möglich gewesen.

John Lloyd von der McGill University, Montréal, Canada trug durch seinen Einsatz beim Portieren und Installieren seines hervorragenden Softwarepakets RCI/RCCL entscheidend zum raschen Gelingen des Roboterprojektes bei. Auch nach dem offiziellen Installationsbesuch war er jederzeit ansprechbar und hilfsbereit, dieses oder jenes Problem aus der Welt zu schaffen.

Stan Berkovich gebührt Dank für seinen wesentlichen Programmierbeitrag zu den Ergebnissen der Zeitserienvorhersage mit dem „Neuronengas“-Modell.

Danken möchte ich auch Prof. Ahuja, der uns den PUMA 562 Roboterarm im Beckman Institute zugänglich machte, sowie seinen Mitarbeitern Chris Debrunner und Arun Krishnan die stets zuvorkommend waren, wenn es um Mitbenutzung von Laborraum und Computerperipherie ging.

Andreas Windemuth danke ich für etliche UNIX und Computer Tips, die mir das Arbeiten auf den verschiedenen Gruppenrechnern komfortabler machten.

Bei Markus van Almsick, Helmut Heller, Christian Kurrer, Thomas Martinetz, Klaus Obermayer, Benno Pütz und Andreas Windemuth möchte ich mich für das Lesen der Urfassung bedanken. Sie halfen durch ihre Korrekturen und Kritik, die Verständlichkeit der vorliegenden Arbeit deutlich zu verbessern.

Meinen Eltern möchte ich ganz herzlich danken, dafür, daß sie mir dieses Studium ermöglichten, und für ihre andauernde, liebevolle Beratung und Unterstützung bei allen meinen Vorhaben.

Diese Arbeit wurde vom Beckman Institute for Advanced Science and Technology of the University of Illinois at Urbana-Champaign, durch ein Grant (DIR-90-15561) der National Science Foundation und durch ein Stipendium der Firma Siemens unterstützt.

Literaturverzeichnis

[1] Androx Corp. DCN9000601 Revision 4.0: "Androx ICS Library Programmer's Reference Manual".

[2] S. Berkovich, J. Walter, T. Martinetz und K. Schulten: "Prediction of chaotic time series with "neural gas" algorithm", *(in Vorbereitung)*, (1991).

[3] M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Pérez und M. Mason, editors: "Robot Motion: Path Planning and Control". MIT–Press, 1982.

[4] D. Broomhead und G. King: "Extracting qualitative dynamics from experimental data", *Physica*, **20D**, 217–236 (1986).

[5] M. Casdagli: "Nonlinear predictions of chaotic time series", *Physica D*, **35**, 335–356 (1989).

[6] J. Craig: "Introduction to Robotics". Addison-Wesley, 1986.

[7] J. Cremers und A. Hübler: "Construction of differential equations from experimental data", *Zeitschrift der Naturforschung*, **42a**, 797–802 (1987).

[8] J. Crutchfield, J. Farmer, N. Packard und R. Shaw: "Chaos", *Scientific American*, **254**, 46–57 (1986).

[9] J. Crutchfield und B. McNamara: "Equations of motion from a data series", *Complex Systems*, **1**, 417–452 (1987).

[10] J. Denavit und R. Hartenberg: "A kinematic notation for lower-pair mechanisms based on matrices", *ASME Journal of Applied Mechanics*, Seiten 215–221215–221 (1955).

[11] J. Deppisch, H.-U. Bauer und T. Geisel: "Hierarchical training of neural networks and prediction of chaotic time series", *Phys. Lett. A*, Seite (1990), (preprint).

[12] J. F. Engelberger: "Robotics in Service". MIT Press, 1989.

[13] E. Erwin, K. Obermayer und K. Schulten: "Convergence properties of self-organizing maps", in "Proc. ICANN, Helsinki", 1991, (in press) sowie private Mitteilung.

[14] J. Farmer und J. Sidorowich: "Predicting chaotic time series", *Physical Review Letters*, **59**(8), 845–848 (1987).

[15] J. Farmer und J. Sidorowich: "Exploiting chaos to predict the future and reduce noise", Technical Report LA-UR-99-901, Theoretical Division and Center for Nonlinear Studies, Los Alamos, 1988.

[16] I. Fischler und O. Firschein, editors: "Computer Vision". Morgan Kaufmann Publishers Inc., 1987.

[17] A. Fraser und H. Swinney: "Independent coordinates for strage attractors from mutaual information", *Physical Review*, **33A**, 1134–1140 (1986).

[18] D. Gabor: "Communication theory and cybernetics", *Trans. of the Inst. of Radio Engineers*, **CT-1**(9), 9 (1954).

[19] D. Graph und W. LaLonde: "A neural controller for collision-free movement of general robot manipulators", in "IEEE International Conference on Neural Networks, San Diego", Seiten 77–84, 1988.

[20] P. Grassberger und I. Procaccia: "Characterization of stange attractors", *Physical Review Letters*, **50**, 346–349 (1983).

[21] V. Hayward und R. Paul: "Robot manipulator control under UNIX: RCCL, a Robot Control C Library", *International Journal of Robotics Reseach*, **5**(4), 94–111 .

[22] R. Hecht-Nielsen: "Neurocomputing". Addison-Wesley, 1989.

[23] J. Hertz, A. Krogh und R. Palmer: "Introduction to the Theory of Neural Computation, SFI Lecture Notes". Addison-Wesley, CA, 1991.

[24] J. J. Hopfield: "Neural networks and pysical systems with emergent collective computational abilities", *Proc Natl Acad Sci USA*, **79**, 2554–2558 (1982).

[25] J. J. Hopfield: "Neurons with graded response have collective computational properties like those of two-state neurons", *Proc Natl Acad Sci USA*, **81**, 3088–3092 (1984).

[26] T. Kohonen: "Self-organized formation of topographically correct feature maps", *Biol. Cybern.*, **43**, 59–69 (1982).

[27] T. Kohonen: "Self-organization and Associative Memory", Volume 8 of *Springer Series in Information Science*. Springer, Berlin, Heidelberg, New York, 1984.

[28] M. Kuperstein: "Adaptive visual-motor coordination in multijoint robots using parallel achitecture", in "IEEE Int. Automat. Robotics (Raleigh, NC)", Seiten 1596–1602, 1987.

[29] M. Kuperstein: "Neural model of adaptive hand-eye coordination for single postures", *Science*, **239**, 1301–1311 (1988).

[30] A. Lapedes und R. Farber: "Nonlinear signal processing using neural networks: Prediction and system modelling", Technical Report LA-UR-87-2662, Los Alamos, 1987.

[31] J. Lloyd, M. Parker und R. McClain: "Extending the RCCL programming environment to multiple robots and processors", in "IEEE Conf. Robotics and Automat. (Philadelphia, Pa)", Seiten 465–469, April 1988.

[32] J. Lloyd: "RCI/RCCL User's Guide and Reference Dokumentation", Computer Vision and Robotics Laboratory, McGill Research Centre for Intelligent Machines, McGill University, Montréal, Québec, Canada, (Proof Version Aug. 1990).

[33] M. Mackey und L. Glass: "Oscillation and chaos in physiological control systems", *Science*, **197**, 287–289 (1977).

[34] M. Mackey und L. Glass: "The Rhythms of Life". Princeton, 1988.

[35] T. Martinetz, H. Ritter und K. Schulten: "Three-dimensional neural net for learning visuomotor coordination of a robot arm", *IEEE Trans. Neural Networks*, **1**(1), 131–136 (1990).

[36] T. Martinetz und K. Schulten: "A "neural gas" network for vector quantization and learning of unknown topologies", *Neural Computation*, (1991).

[37] T. Martinetz und K. Schulten: "A "neural gas" network learns topologies", in "Proc. ICANN, Helsinki", 1991, (in press).

[38] T. Martinetz: "(1990)", Private Mitteilungen.

[39] T. Martinetz: "Selbstorganisierte visuo-motorische Kopplung", Diplomarbeit, Physik Department, Technische Universität München, Theoretische Physik, 1988.

[40] J. Moody und C. Darken: "Fast learning in networks of locally-tuned processing units", *Neural Computation*, **1**, 281–294 (1989).

[41] F. H. Netter, editor: "Nervensystem Neuroanatomie und Physiologie", Volume I. Band 5 of *Farbatlanten der Medizin*. Georg Thieme Verlag Stuttart, 1987.

[42] K. Obermayer, G. G. Blasdel und K. Schulten: "A neural network model for the formation and for the spatial structure of retinotopic maps, orientation- and ocular dominance columns", in "Proc. ICANN, Helsinki", June 1991, (in press).

[43] K. Obermayer, H. Ritter und K. Schulten: "A prinziple for the formation of the spatial structure of cortical feature maps", *Proc Natl Acad Sci USA*, **87**, 8345–8349 (1990).

[44] R. Paul: "Robot Manipulators: Mathematics, Programming, and Control". MIT Press, 1981.

[45] M. Powell: "Radial Basis Function Approximations to Polynomials". Univ. of Cambridge, 1987.

[46] M. Priestley: "State dependent models: a general approach to nonlinear time series analysis", *Journal of Time Series Analysis*, **1**, 47–71 (1990).

[47] I. Rhodes: "A tutorial introduction to estimations and filtering", *IEEE Trans. Autom. Control*, **6**, 688–706 (1971).

[48] H. Ritter, T. Martinetz und K. Schulten: "Neuronale Netze". Addison Wesley, Bonn, 2. Auflage, 1990.

[49] H. Ritter, K. Obermayer und K. S. (ed. L. v. Hemmen): "Physics in Neural Networks", Volume 2, chapter Self-organizing Maps and Adaptive Filters, Springer, New York, 1991, in press.

[50] H. Ritter und K. Schulten: "Topology conserving maps for learning motor tasks", in J. Denker, editor, "Neural Networks for Computing", Seiten 376–380. AIP Conf Proc 151, Snowbird, Utah, 1986.

[51] H. Ritter: "Asymptotic level density for a class of vector quantization processes", Report A9, Helsinki Unversity of Technology, Laboratory of Computer and Information Science, Finland, March 1989.

[52] D. Rumelhart, G. Hinton und R. Williams: "Learning representations by backpropagagation errors", *Nature.*, **323**, 533–536 (1986).

[53] G. Sugihara und R. M. May: "Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series", *Nature.*, **344**, 734–741 (1990).

[54] F. Takens: "Dynamical systems and turbulence", in D. Rand und L. Young, editors, "Detecting Strange Attraktors in Turbulence", Seiten 366–381. Springer, Berlin, 1981.

[55] C. v.d. Malsburg: "Self-organization of orientation sentive cells in the striata cortex", *Kybernetik*, **14**, 85–100 (1973).

[56] J. Walter, T. Martinetz und K. Schulten: "Industrial robot learns visuo-moto coordination by means of "neural-gas" network", in "Proc. Int. Conf. On Artificial Neural Networks, Helsinki". Elsevier, Amsterdam, June 1991, (in press).

[57] J. Walter, H. Ritter und K. Schulten: "Non-linear prediction with self-organizing maps", in "International Joint Conference On Neural Networks, San Diego", Volume 1, Seiten 587–592, June 1990.

[58] Westinghouse Unimation Inc. CT: "Unimate Programming Manual Users's Guide to VAL II, (Version 2.0) Manual 398AG1".

[59] Westinghouse Unimation Inc. CT: "Unimate PUMA Mark III Robot; 500 Series (Models 552/562); Equipment Manual 398AH1", (1990: AEG Westinghouse).

[60] B. Widrow und M. Hoff: "Adaptive switching circuits", *WESCON Conv Record*, **4**, 96–104 (1960).

[61] D. Willshaw und C. v.d. Malsburg: "How patterned neural connections be set up by self-organization", *Proc R Soc London B*, **194**, 431–445 (1976).

Abbildungsverzeichnis