Neuronale Netze
zur
Musterklassifizierung
und
Exploration des Raumes

Zur Erlangung des akademischen Grades

DOKTOR - INGENIEUR

vom Fachbereich Elektrotechnik der

Universität – GH Paderborn

genehmigte Dissertation

vorgelegt von

Dipl.-Inf. Martin Busemann

aus Madfeld / Hochsauerland

Referrent: Prof. Dr. rer. nat G.Hartmann

Kooreferent: Prof. Dr. - Ing. F. Dörrscheidt

Tag der mündlichen Prüfung: 11.09.2000

D14-149

Inhaltsverzeichnis

1. Einleitung

Die Leistungsfähigkeit des menschlichen Gehirns ist sehr beeindruckend. Betrachtet man nur die Fähigkeit Objekte zu finden und zu erkennen, so ist diese bemerkenswert. Zwar sind uns die Computer immer überlegen, wenn es darum geht, komplexe mathematische Probleme zu lösen, doch bei der für uns vergleichbar einfachen Aufgabe, Ziffern oder Buchstaben zu erkennen, erreichen sie sehr schnell das Ende ihrer Leistungsfähigkeit.

Durch das Abschauen und Nachbauen von bekannten Strukturen und Verarbeitungsschritten des Gehirns versucht man, entsprechend leistungsfähige Systeme zu schaffen. Außerdem wird durch nachgebildete Verschaltungen untersucht, wie einzelne Teile des Gehirns arbeiten könnten. Die so erzeugten Systeme werden mit dem Begriff der *neuronalen Netze* bezeichnet. Von einem vollständigen Verstehen des Gehirns ist man aber noch weit entfernt.

Die Haupteinsatzgebiete für neuronale Netze sind zur Zeit:

- Mustererkennung,
- Spracherkennung,
- Optimierungsprobleme,
- assoziative Speicher und
- Selbstorganisation.

In dieser Arbeit werden zwei neuronale Netzwerke unter Verwendung von pulscodierten Neuronen vorgestellt. Das erste dient als Musterklassifikator in einem Erkennungssystem, das Objekte, die auf einem Tisch vor einem Roboter liegen, findet und erkennt. Bei der Konstruktion des Erkennungssystems hielt man sich sehr nah an biologische Vorbilder. Die Objekte können an beliebiger Position und in beliebiger Orientierung aufgenommen werden. Das Erkennungssystem kann die entsprechenden Bilder verarbeiten und bestimmt, wo und in welcher Orientierung die Teile auf dem Tisch liegen. Um einTeil zu erkennen, muß es dem neuronalen Netz nur einmal gezeigt werden. Anschließend ist es in der Lage, das Teil wiederzuerkennen. Es braucht nicht expliziet von Lernen auf Erkennen umgeschaltet werden.

Das zweite neuronale Netz repräsentiert die Umgebung eines Roboters in einem kamerafesten Koordinatensystem. Objekte, die sich im Raum befinden, werden durch *Aktivitätswolken* dargestellt. Bewegt sich die Kamera, dann verschieben sich die Aktivitätswolken in die entgegengesetzte Richtung. Dies geschieht ausschließlich durch Aktivieren und Deaktivieren

von Neuronen. Die Kamera kann in sechs Freiheitsgraden bewegt werden. Die Objekte bleiben stabil durch die Wolken repräsentiert.

Zur Simulation des Verhaltens von neuronalen Netzen wird eine große Rechenleistung benötigt. Erst mit den heutigen Computern ist es möglich, Simulationen in annehmbarer Zeit durchzuführen. Es wurde untersucht, inwieweit es sinnvoll ist, die beiden vorgestellten Netzwerke zu parallelisieren und auf einen Parallelrechner zu implementieren.

Um die beiden neuronalen Netze in bestehende Systeme zu integrieren, war es notwendig, die Modelle zu abstrahieren. Es werden die angewandten Abstraktionsmethoden vorgestellt. Als Abschluß werden Parallelitäten zum Cerebellums aufzeigt.

Eingebettet war die Arbeit in ein vom Bundesministerium für Forschung und Technik gefördertes Verbundprojekt zur **SEN**sorischen Exploration des Greifraums eines **ROB**oters (*SENROB*) (Fig 1.1). An diesem Verbundprojekt waren folgende Partner beteiligt:

- Prof. R. Eckmiller, Universität Bonn
- Prof. Dr.-Ing. E. Freund, Institut für Roboterforschung in Dortmund
- Prof. G. Hartmann, Universität-Gesamthochschule Paderborn
- Prof. B. Hostika, Ph.D. Fraunhofer-Institut für Mikroelektronische Schaltungen und Systeme in Duisburg
- Firma Pietch, Ettlingen.

Ziel des Projektes war es, einen sensorisch geführten Roboter zu erstellen. Hierbei wurde untersucht, inwieweit einzelne Komponenten sich durch neuronale Netze realisieren lassen.

Verschiedene Montageteile liegen auf einem Tisch vor dem Roboter und werden von einem Visionsystem, das in Paderborn erstellt wurde, als bekannt oder unbekannt eingestuft. Außerdem werden vom Visionsystem die Position und die Orientierung der Teile bestimmt. Diese Daten werden an das von der Dortmunder Arbeitsgruppe entwickelte wissensbasierte Montageplanungssystem übermittelt, das die Daten in ein „Umweltmodell“ einträgt. Dort stehen die für den Greifvorgang benötigten Parameter. Durch eine übergeordnete KI-Schale erfolgt dann eine Montageplanung, um die Objekte zu greifen und zusammenzubauen.

Die Bonner Arbeitsgruppe erstellte neuronale Netze zur Kinematik und Bahnvorgabe des Roboters. Außerdem wurde die konventionelle Regelung des Roboters durch ein neuronales Netz ersetzt. Zur Beschleunigung der neuronalen Komponenten erstellte die Arbeitsgruppe in Duisburg eine spezielle neuronale Hardware.

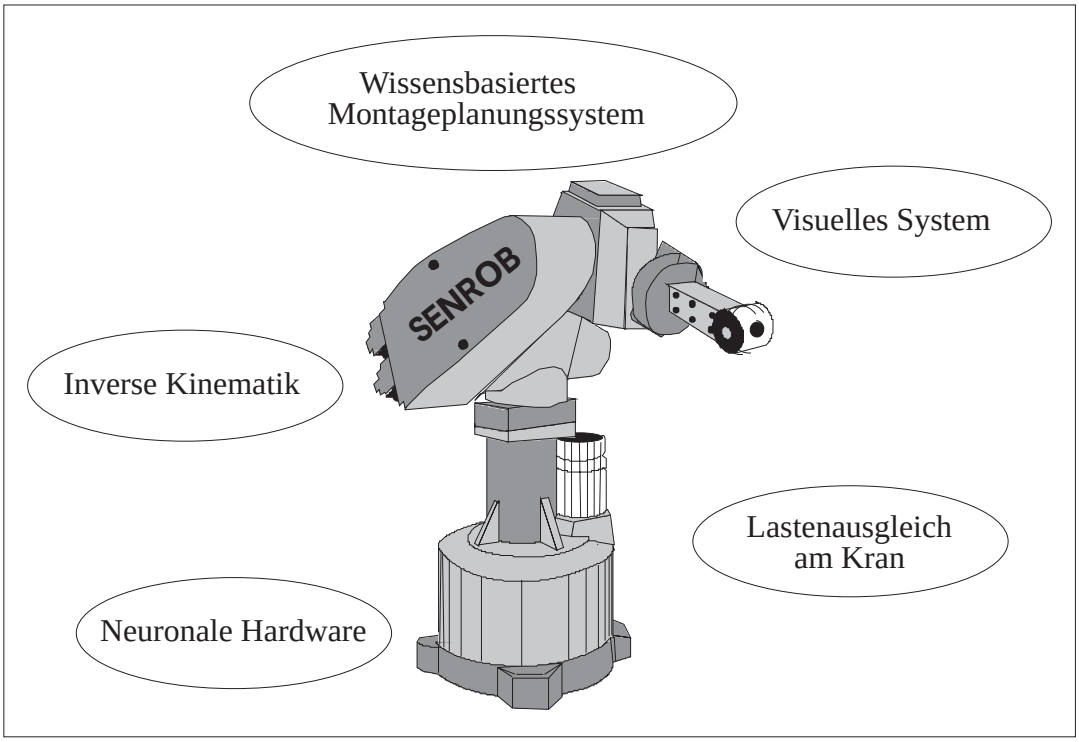


Fig. 1.1: Die einzelnen Komponenten des SENROB-Projektes.

Als privates Wirtschaftsunternehmen beschäftigte sich die Firma Pietch mit dem Lastenausgleich an einem Kran, wodurch die Nutzungsmöglichkeit von neuronalen Netzen für diese Aufgabe untersucht wurde.

2. Erkennungssystem PDIS

Der Arbeitsgruppe in Paderborn obliegt die visuelle Erfassung der einzelnen zu montierenden Teile durch ein *perspektiven-* und *distanzinvariantes System* (*PDIS*) [Busemann 1993], [Hartmann 1993] und die dynamische räumliche Repräsentation vorhandener Objekte durch ein *Raumrepräsentationsnetzwerk* (*RRN*) [Hartmann 1992a].

Die einzelnen Leistungen des PDIS sind in Fig. 2.1 zu sehen. Die Objekte liegen in beliebiger Lage und Orientierung auf einem Tisch und werden mit einer Farbkamera aufgenommen. Zuerst wird die gesamte Szene erfaßt, und alle Objekte auf dem Tisch werden lokalisiert [Wiemers 1994]. Um dann die einzelnen Objekte aufnehmen zu können, ist die Kamera an der sechsten Achse des Roboters neben dem Greifer befestigt. So kann mit Hilfe des Roboters die Kamera nacheinander über jedes Objekt bewegt und so jedes Objekt einzeln aufgenommen werden. Eine der Retina höherer Lebewesen entsprechende Abtastung ermöglicht es, daß entfernte Objekte die gleiche Auflösung haben wie naheliegende [Kräuter 1995].

Die Distanzinvarianz wird durch neuronale Abbildungen, die mittels Entfernungsmessung gesteuert werden, erreicht. Um die Orientierungsinvarianz zu erhalten, wurden zu jedem Teil eine oder mehrere Vorzugsorientierungen bestimmt, in die das Objekt in seiner internen Repräsentation ebenfalls durch neuronale Abbildungen gedreht wird [Busemann 1993], [Hartmann 1993]. Eine dem visuellen Cortex entsprechende neuronale Repräsentation erlaubt es, geringfügige Änderungen von Lage, Orientierung und Größe der Objekte zu tolerieren. Das Lernen von Objekten bzw. deren neuronale Repräsentation erfolgt durch ein neuronales Netzwerk in einem Schritt [Hartmann 1991a].

2.1 Künstliche Retina

Das Bild der Farbkamera wird so verarbeitet, daß die Bildpunkte in ihrer Anordnung und Dichte der Rezeptorenverteilung der Retina (Netzhaut) entsprechen. Die Abtastung ist eine logarithmisch-polare Abtastung. Hierdurch ist es möglich, die besonderen Eigenschaften dieser Abtastung auszunutzen, die auch schon von Reitböck [Reitböck 1984] und Schwarz [Schwarz 1981] erkannt wurden. So wird z.B. eine Drehung des Objektes um das Abtastzentrum durch eine Translation auf dem polaren Abtastgitter in tangentialer Richtung kompensiert und eine Vergrößerung bzw. Verkleinerung des Bildes aufgrund sich ändernder Entfernungen zwischen Objekt und Kamera durch eine Verschiebung in radialer Richtung realisiert werden.

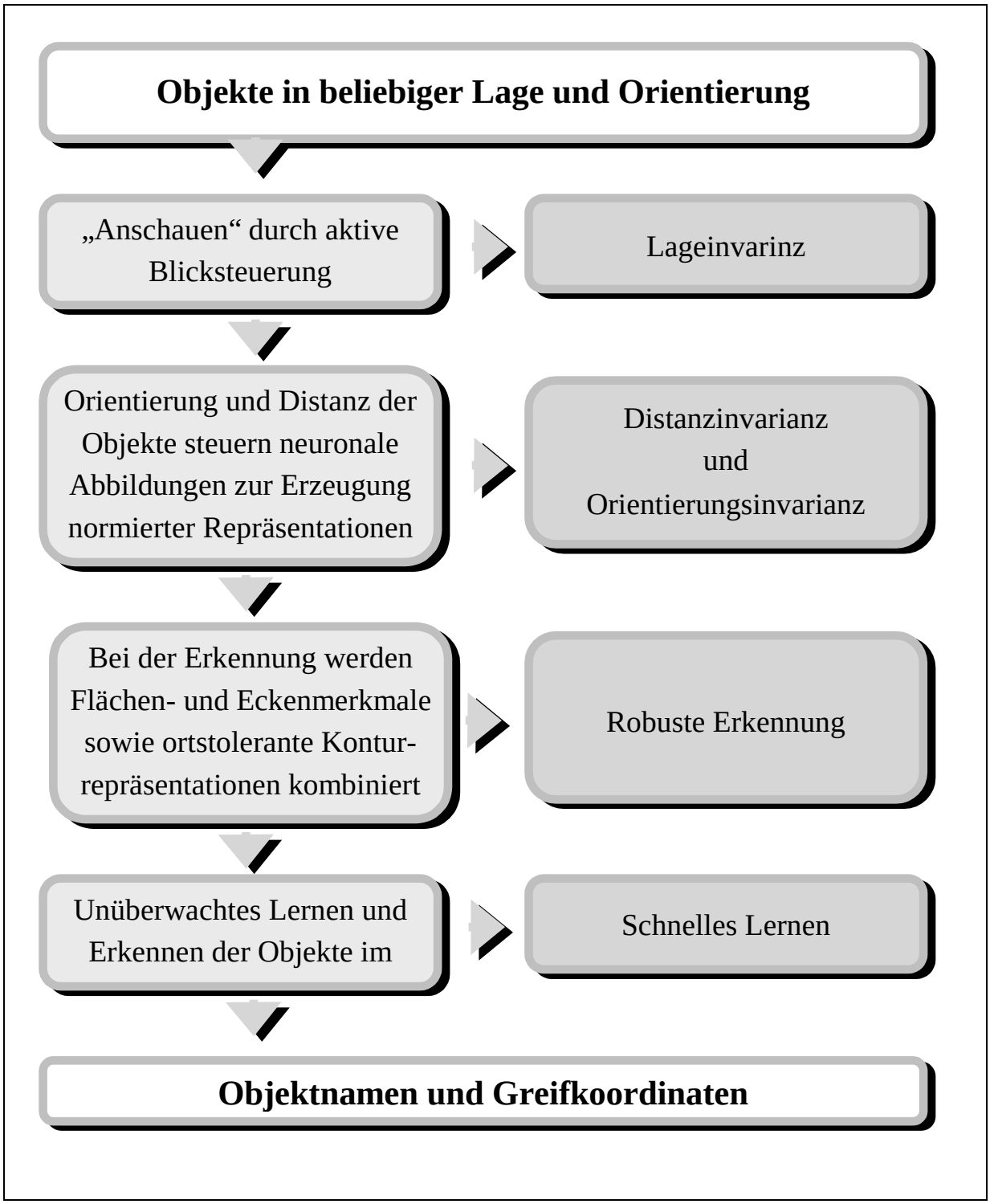


Fig. 2.1: Die einzelnen Leistungsmerkmale des PDIS.

Doch hat die logarithmisch-polare Abtastung den großen Nachteil, daß ein Objekt in der Mitte am Fovealisierungspunkt sehr dicht abgetastet wird, während die Information im peripheren Bereich sehr spärlich ist. Doch gerade dort befinden sich oft die für die Erkennung relevanten Strukturen. Dies läßt sich nicht dadurch lösen, daß eine Retina mit dichterer Abtastung benutzt wird. Aus diesem Grund wurde eine künstliche Retina konstruiert, die diesen Nachteil nicht aufweist, aber noch die Vorteile der herkömmlichen Retina besitzt [Wiemers 1994].

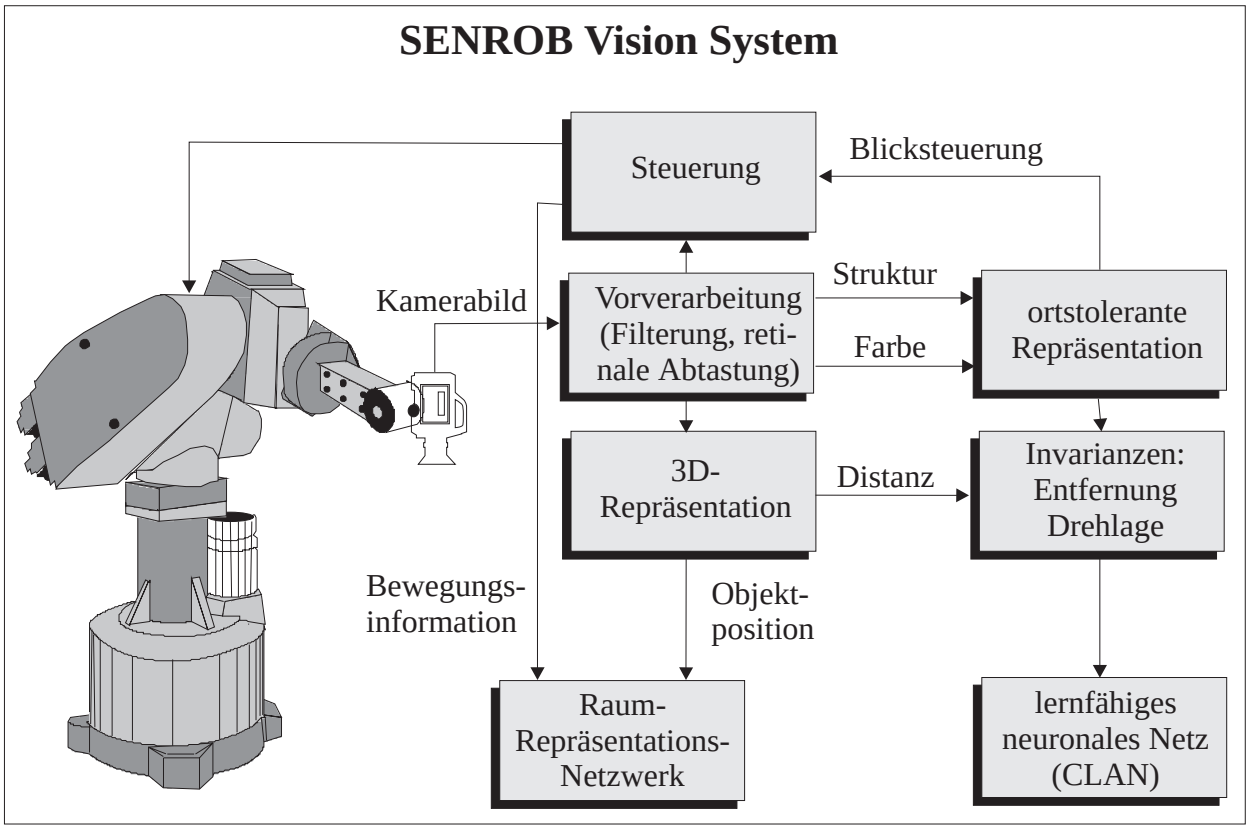


Fig. 2.2: Eine Gesamtübersicht über die einzelnen Module des Erkennungssystems.

Um diese zu erstellen, wird zuerst die herkömmliche Retina erzeugt und aus ihr der Teil herausgeschnitten, in dem die Auflösung dicht genug ist. Anschließend wird eine zweite dichtere Retina gebildet. Aus dieser wird wiederum derjenige Teilbereich betrachtet, in dem das Abtastraster dicht genug ist. Die beiden Raster werden dann überlagert.

Dieses Verfahren wird insgesamt fünfmal durchgeführt. Anschließend erhält man bei geeigneter Parameterwahl eine künstliche Retina mit den Ringen R_0 - R_4, die noch die ursprünglichen Eigenschaften der herkömmlichen logarithmischen Retina aufweist (Fig. 2.3) und bei welcher außerdem die Abtastung auch in den peripheren Bereichen dicht genug ist. Als wei-

teres werden kleinere Bilder entfernter Objekte mit gleicher Auflösung repräsentiert wie retinafüllende Bilder naheliegender Objekte.

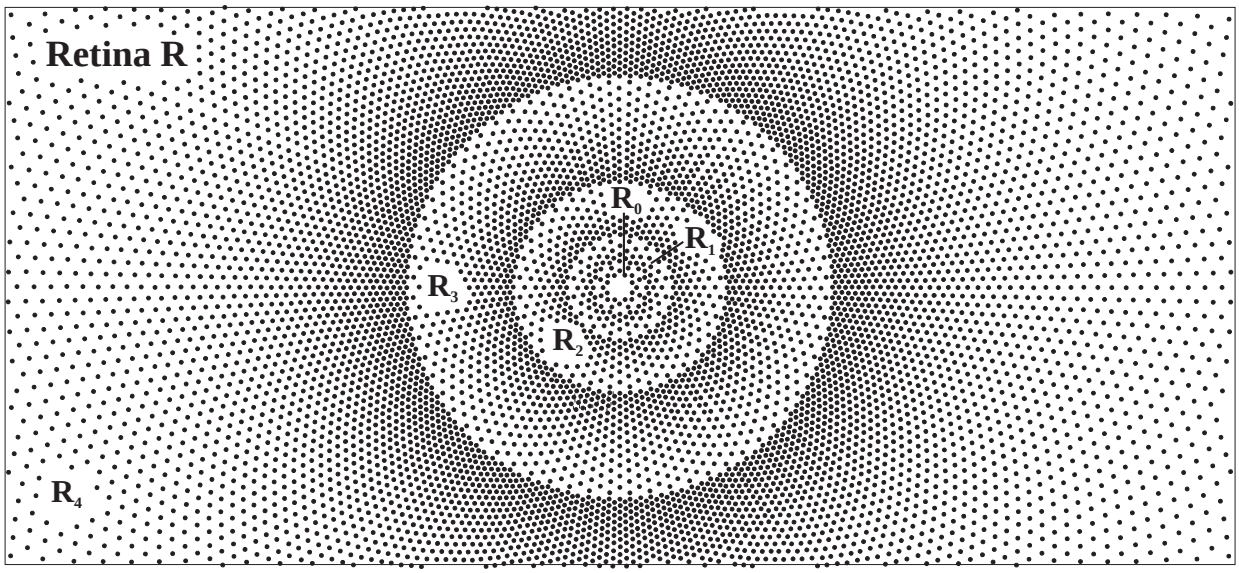


Fig. 2.3: Die künstliche Retina ist aus den Ringen R_0 bis R_4, die Teilbereiche der herkömmlichen logarithmischen Retina sind, aufgebaut.

2.2 Ortstolerante Repräsentation und Invarianzen

Das retinale Bild wird durch Neuronen mit kreisförmigen rezeptiven Feldern repräsentiert, die auf Helligkeitsdifferenzen zwischen Zentrum und Peripherie reagieren. Diese ermitteln den Konturverlauf des Objekts. Mit dem verwendeten Detektorensatz ist eine Auflösung der Kantenorientierungen von 15 Grad möglich. Damit nicht schon durch eine Verschiebung des Bildes um ein Pixel die Konturverläufe als nicht übereinstimmend angesehen werden, wird eine Kante durch eine "Orientierungswolke" von benachbarten Detektoren repräsentiert. Hierdurch wird bei einer Verschiebung um ein Pixel zwar nicht mehr eine hundertprozentige Übereinstimmung erzielt, aufgrund der Überlappung der entsprechenden Codierungen stimmen die beiden Bilder aber zu einem gewissen Prozentsatz noch überein.

Neben der Kanteninformation werden flächenbasierte Merkmale (helle/dunkle Flächen, Farben) codiert. Außerdem werden als weiteres konturbasiertes Merkmal die Ecken eines Objektes extrahiert und zur Erkennung mit herangezogen. Die Ecken sind notwendig, um z.B. einen Kreis von einem Achteck zu unterscheiden. Durch die drei beschriebenen Komponenten kann eine robuste Erkennung durchgeführt werden.

Als nächster Verarbeitungsblock schließt sich die Invarianzenbildung an. Über den Roboter wird jeweils die Entfernung des Objektes zur Kamera ermittelt. Durch gesteuerte Abbildungen (parametrische Mappings) mit Hilfe präsynaptischer Steuerungen werden die codierten Bilder in ihrer internen Repräsentation immer wieder in eine vordefinierte Entfernung geschoben.

In ähnlicher Weise werden die Bilder der Objekte in eine objekteigene Vorzugsorientierung verschoben. Der Parameter für diese Verschiebung ist der Winkel, um den das Objekt unabhängig von der Größe aus seiner Vorzugsorientierung gedreht ist. Die Vorzugsorientierung entspricht der lokalen Kantenorientierung, die für das entsprechende Objekt am häufigsten vorkommt. Sie läßt sich leicht durch ein Histogramm der vorhandenen Orientierungen finden. Obwohl die lokalen Kantendetektoren nur mit einer Winkelauflösung von ±15° arbeiten, kann die Orientierung eines der angebotenen Objekts auf ±1,4° genau bestimmt werden.

2.3 Lernfähiges neuronales Netz

Am Ende der Verarbeitungsstufen steht das *CLAN* (**C**losed **L**oop **A**ntagonistic **N**etwork), welches für jede Position im Raster der internen Repräsentation so viele binäre Eingänge hat, wie es mögliche Merkmale gibt. Ein CLAN-Eingang ist genau dann aktiv, wenn das entsprechende Merkmal an der Position vorhanden ist. Das CLAN teilt die Eingabemuster nach einmaliger Präsentation in verschiedene Musterklassen ein, wobei, falls erforderlich, eine neue Musterklasse gebildet wird. Da das CLAN Gegenstand der vorliegenden Arbeit ist, wird es in Kapitel 4 noch ausführlicher beschrieben.

Parallel zum Erkennungsvorgang arbeitet das Raumrepräsentationsnetzwerk (RRN). Objekte, die innerhalb des Greifraumes liegen, werden durch aktive Neuronen in einem dreidimensionalen Gitter repräsentiert. Die Kameraposition liegt im Mittelpunkt des Gitters. Bei Bewegungen des Roboters gibt die Steuerung die Bewegungsinformation an das RRN weiter. Hierdurch wird die Repräsentation der Objekte entgegengesetzt zu der Bewegungsrichtung im Gitter verschoben, so daß zu jedem Zeitpunkt die Position und die Orientierung der Objekte in Bezug auf die Kameraposition festgehalten wird. Diese Angaben sind für das Greifen von Werkstücken und bei der Kollisionsvermeidung von Bedeutung. Kapitel 6 und darauffolgende Kapital geht ausführlich auf das RRN ein.

3. Biologische und künstliche Neuronen

Grundlage der neuronalen Netze bilden die künstlichen bzw. technischen Neuronen, die in einfachster Form die biologischen Neuronen nachbilden. Um die Arbeitsweise des in dieser Arbeit benutzten technischen Neurons zu verstehen, soll es, ausgehend vom biologischen Neuron, vorgestellt werden.

3.1 Biologische Neuronen

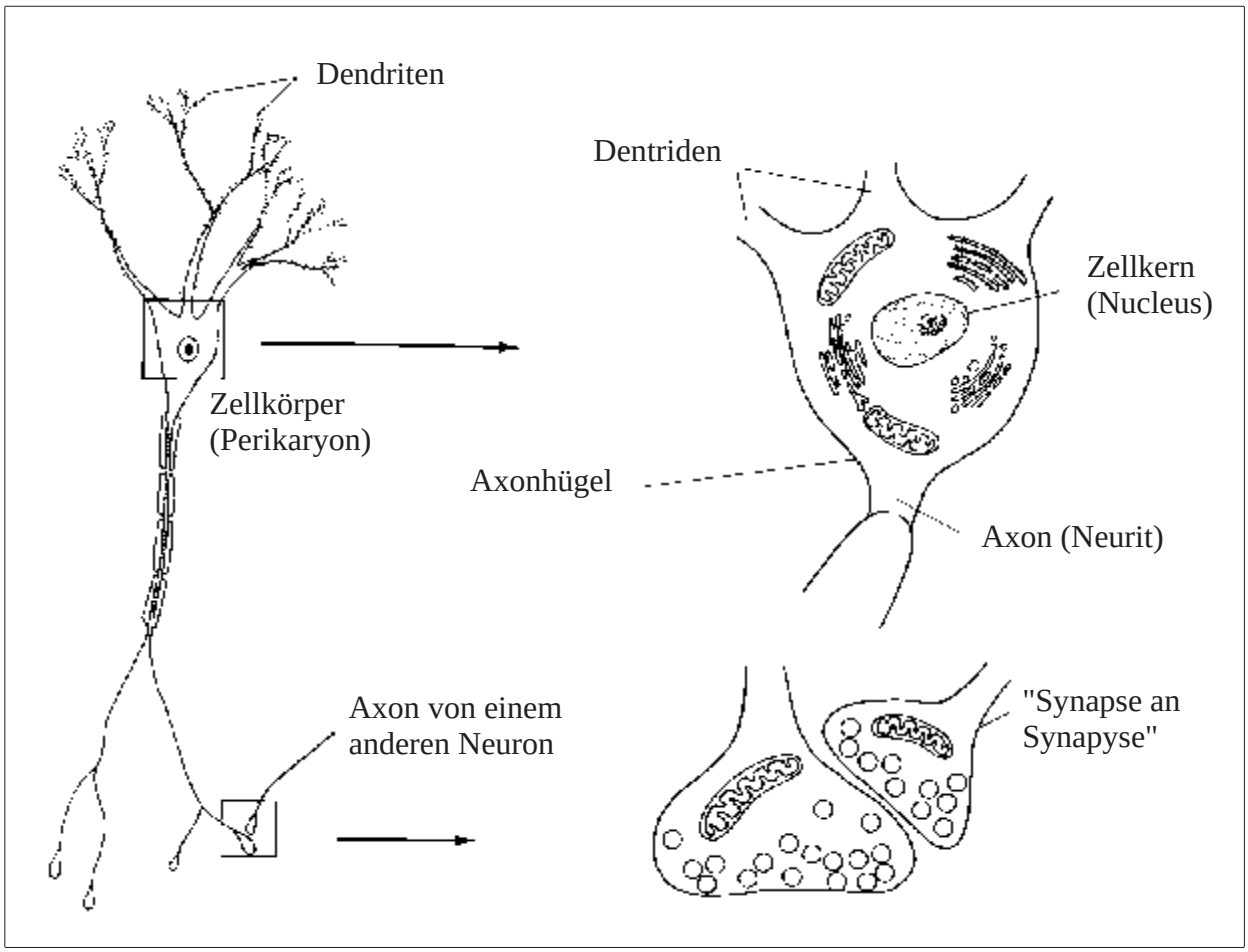


Fig. 3.1: Aufbau eines biologischen Neurons [Thompson 1990].

Ein Gehirn ist aus einzelnen Nervenzellen oder Neuronen aufgebaut (Fig. 3.1). Man schätzt, daß das menschliche Gehirn aus ungefähr 100 Milliarden (10^{11}) Neuronen besteht ([Thompson 1990]). Sie übertragen Informationen auf andere Neuronen oder auf Muskel- oder Drüsenzellen. Die Leistung des Gehirns beruht auf seiner strukturellen und funktionellen Organi-

sation, also darauf, wie die Neuronen im Schaltplan des Gehirns vernetzt sind, wie sie zusammenarbeiten und sich gegenseitig beeinflussen.

Über die *Dendriten* und den *Zellkörper* empfängt ein Neuron Informationen, und über ihr *Axon* leitet es Signale an andere Nervenzellen oder sonstige Zellen weiter. Das Axon spaltet sich am Ende in astähnlichen Verzweigungen auf, die *Synapsen*. Die Synapsen stellen die funktionelle Verbindung zwischen einem Axon und anderen Neuronen her. Hier wird die Information von einer Nervenzelle auf die nächste übertragen. Die Informationsleitung geschieht nur in diese Richtung. Ein einzelnes Neuron kann mehrere tausend Synapsen aufweisen.

Normalerweise hält ein Neuron eine Potentialdifferenz (*Ruhepotential*) zwischen Innen- und Außenseite. Die Spannung innerhalb des Neurons beträgt relativ zur Außenseite ca. –75 mV. Durch die ankommenden Informationen von anderen Neuronen wird das Potential, das auch als *Membranpotential* bezeichnet wird, verändert. Grundsätzlich wird zwischen *erregenden* und *hemmenden* Eingangsinformationen unterschieden. Erregende Informationen erhöhen das Potential, während hemmende das Potential erniedrigen. Übersteigt das Potential eine *Schwelle* (ungefähr –60 mV), dann wird das Neuron *aktiv*. Es kommt zu einem plötzlichen Anstieg der Spannung bis auf einen Höchstwert von +30 bis +40 mV. Anschließend erfolgt innerhalb 1 ms ein Spannungsabfall bis unter das Ruhepotential. Das Ruhepotential wird nach ca. einer weiteren msec wieder erreicht. Der Bereich vom Anstieg bis zum Abfall des Membranpotentials wird als *Aktionspotential* oder *Spike* bezeichnet und die darauffolgende Phase, in der die Spannung unter das Ruhepotential abfällt und sich dann langsam wieder dem Ruhepotential annähert, als *Nachpotential* (Fig. 3.2).

In der Zeit vom Beginn des Spikes bis zu dessen Ende kann ein Neuron kein weiteres Aktionspotential erzeugen (*absolute Refraktärzeit*). Während der Phase des Nachpotentials kann das Neuron zwar wieder aktiv werden, doch ist eine stärkere Reizung notwendig (*relative Refraktärzeit*). Danach reicht dann ein "normaler" Input aus.

Durch ein einzelnes Eingangssignal (auch *Impuls* genannt) kann ein Neuron nicht aktiviert werden. Damit das Membranpotential größer als die Schwelle wird, müssen mehrere Impulse von verschiedenen Neuronen gleichzeitig eintreffen, oder mehrere Impulse müssen kurzzeitig hintereinander eintreffen.

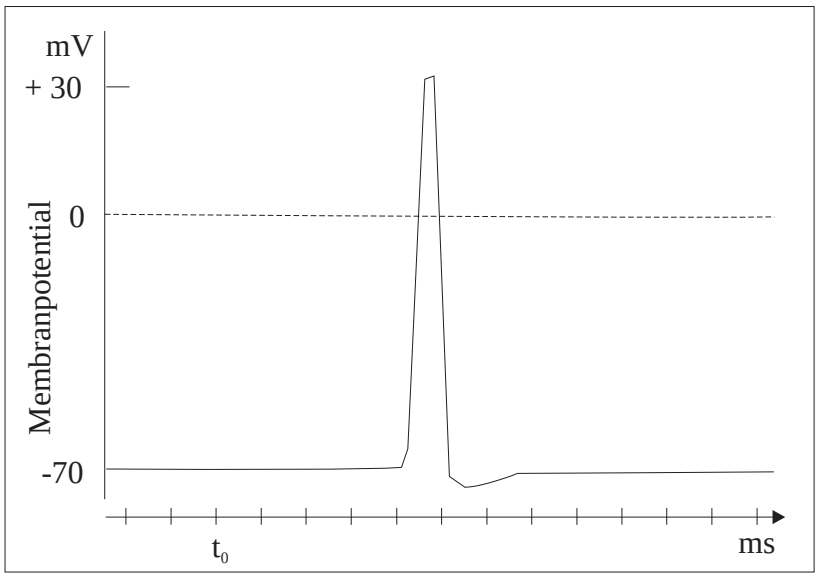


Fig. 3.2: Verlauf des Aktionspotentials [Thompson 1990].

3.2 Biologische präsynaptische Verbindungen

Die Information von den Synapsen wird nicht nur auf die Dendriten oder Zellkörper von Neuronen übertragen. Einige Synapsen, die sogenannten *präsynaptische Synapsen,* übermitteln die Informationen an andere, den *postsynaptischen Synapsen*. (Fig. 3.1 unten rechts). Dies wird als *präsynaptische Verbindung* bezeichnet. Die präsynaptische Synapse wirkt erregend oder hemmend auf die postsynaptische.

Die Wirkung der präsynaptischen Verbindung sei beispielhaft an der hemmenden präsynaptischen Synapse erklärt. In dem Beispiel wird davon ausgegangen, daß eine einzelne Synapse mit einem Impuls das Neuron so anregen kann, daß hierdurch ein Aktionspotential erzeugt wird.

Ist die präsynaptische Synapse nicht aktiv, beeinflußt sie die postsynaptische Synapse nicht, so daß diese in gewohnter Form ein Signal an das Neuron sendet, und das Neuron ein Aktionspotential erzeugt (Fig. 3.3). Im anderen Fall wird das Signal der postsynaptischen Synapse gehemmt. Das Neuron wird dadurch nicht mehr so stark oder gar nicht erregt und erzeugt kein Aktionspotential (Fig. 3.4). In einer ganz einfachen Form kann der präsynaptische Vorgang mit einem elektrischen Schalter verglichen werden, der eine Verbindung durchschaltet oder unterbricht.

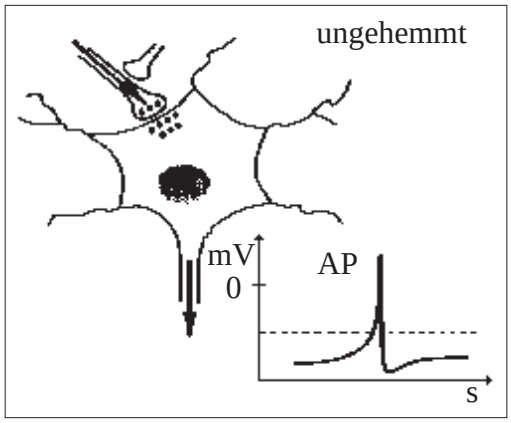


Fig. 3.3: Die nicht aktive hemmende präsynatische Synapse.

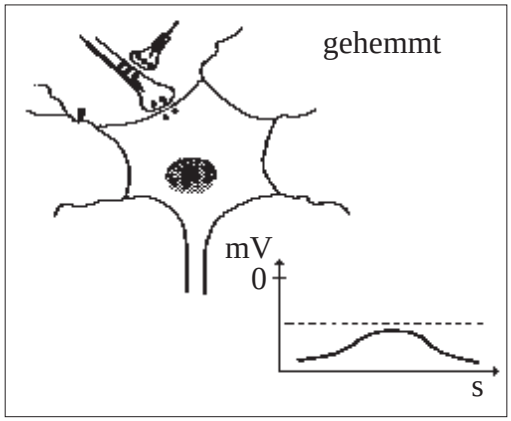


Fig. 3.4: Die aktive präsynaptische Synapse.

3.3 Technische Neuronen

Mit einem technischen Neuron wird versucht, ein biologisches Neuron in einfachster Form nachzubilden. Wenn im folgenden vom Neuron gesprochen wird, dann ist immer das technische Neuron gemeint. In Fig. 3.5 sehen wir eine entsprechende *Unit*, wie das künstliche Neuron auch bezeichnet wird. Von ihr werden ebenfalls Signale empfangen, verarbeitet und an andere Neuronen weitergegeben.

Die Signale von anderen Neuronen enden an den *synaptischen Gewichten* w_{ij}. Durch die Gewichte w_{ij} wird festgelegt, wie stark das Eingangssignal auf das Neuron wirkt. Positive Gewichte sind erregende und negative hemmende Verbindungen. In Grafiken werden positive Gewichte durch einen Kreis symbolisiert (alle Gewichte in Fig. 3.5), während negative Gewichte durch ein Rechteck dargestellt werden. Der gesamte Input wird zur *Eingangsfunktion*

$$p_i = \Sigma w_{ij} \cdot u_j \qquad (3.1)$$

aufsummiert.

Über die Aktivierungsfunktion

$$a_i = F(p_j) \tag{3.2}$$

wird der Zustand des Neurons i bestimmt und die Ausgangsfunktion

$$o_i = G(a_i) \tag{3.3}$$

berechnet.

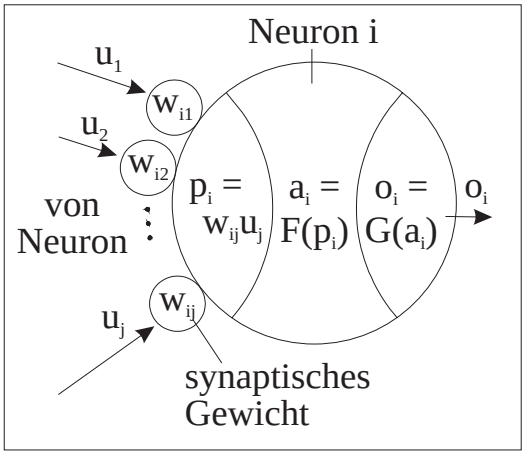


Fig. 3.5: Ersatzschaltbild des technischen Neurons.

Die einzelnen Units unterscheiden sich im wesentlichen durch ihre Ausgangsfunktion. Einige der bekanntesten sind

- einfache lineare Einheit,
- lineare Schwellwert-Einheit,
- thermodynamische Einheit,
- sigmoide Einheit und
- sigma-Pi Einheit.

3.4 Aufbau der neuronalen Netze

Aus den Einheiten werden die neuronalen Netze aufgebaut, die allgemein in *hierarchische* und *nicht-hierarchische* Netzwerktypen aufgeteilt sind. In einem hierarchischen Netzwerk gibt es nur Verbindungen von einer Neuronenschicht zur nächsten. Es gibt keine Rückkop-

pelungen. Alle anderen Netzwerke zählen zu den nicht-hierarchischen, in denen die Neuronen beliebig miteinander verschaltet sein können.

In den neuronalen Netzen wird Wissen in den Gewichten w_{ij} und im Aktivierungszustand gespeichert. Das Wissen wird durch die Netze erlernt, wobei das Lernen zumeist bedeutet, daß die Gewichte richtig eingestellt (*trainiert*) werden. Man unterscheidet das Lernen mit Lehrer (*supervised)* und ohne Lehrer (*unsupervised*). Beim Lernen mit Lehrer wird der erzielte mit dem gewünschten Output verglichen, und die Gewichte werden so eingestellt, daß sich allmählich eine minimale Differenz ergibt. Beim *unsupervised Learning* werden dem Netz nur *Muster*, wie das zu Lernende oft bezeichnet wird, präsentiert. Es gibt keinen vordefinierten Output. Das Netz soll, aufgrund der in den Mustern vorhandenen Gemeinsamkeiten, diese selbständig in Klassen einteilen. Das Finden und Bilden von Klassen ist Aufgabe des Netzes.

3.5 Technische präsynaptische Hemmung

Mit der oben bereits aufgelisteten Sigma-Pi Unit wird die präsynaptische Hemmung simuliert. Das synaptische Gewicht eines Neurons endet nicht auf dem Neuron i, sondern auf dem synaptischen Gewicht eines anderen Neurons j. Dadurch wird das Gewicht w_{ij} verändert und somit auch die Eingangsfunktions des Neurons i (Fig. 3.6). Sie berechnet sich nun wie folgt:

$$p_i = \Sigma w_{ij} \cdot u_j \cdot \pi_{jk}.$$

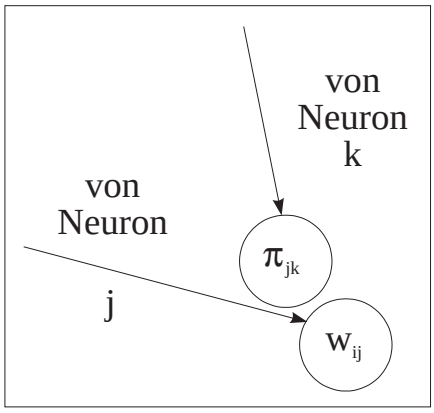


Fig. 3.6: Modellierung der präsynaptische Hemmung.

Die Aktivierungs- und Ausgangsfunktion ändert sich hierdurch nicht.

3.6 Pulscodierte Neuronen

In der Simulationen benutzen wir das pulscodierte Neuron, wie es von French und Stein 1970 als Ersatzschaltbild vorgestellt wurde [French 1970]. Dieses kommt dem biologischen Neuron in seinem Verhalten ziemlich nahe. Die einzelnen Neuronen kommunizieren über Impulsfolgen miteinander. So muß man den Output eines Neurons über einen längeren Zeitraum betrachten, damit man eine Aussage über die Aktivität des Neurons erhält.

Die Ausgangssignale von Neuronen 1 bis j werden mit dem Gewichtsfaktor w_{ij} verknüpft. Dann wird der gesamte Input zum Membranpotential M_i des Neurons i aufsummiert und in einem Diskriminator mit einem *statischen Schwellwert* T_S verglichen. Ist M_i größer als T_S, dann erzeugt der Pulsgenerator einen Impuls o_i, der wiederum als Eingabe für andere Neuronen dient. Wird ein Impuls ausgelöst, dann erhöht sich gleichzeitig die *dynamische Schwelle* T_D auf ihren Maximalwert und wirkt negativ auf den Eingang des Diskriminators. Die dynamische Schwelle sinkt mit einer Verzögerungszeit τ_T wieder gegen Null (Fig. 3.7).

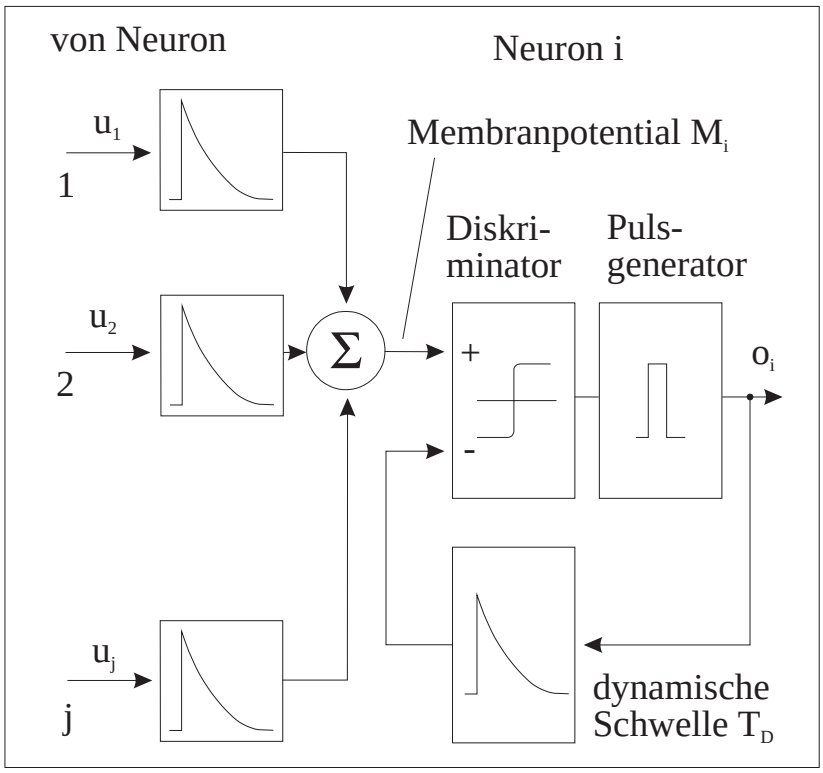


Fig. 3.7: Das Ersatzschaltbild des pulscodierten Neurons.

Das Membranpotential eines Neurons ist auf einen Maximalwert M_{max} begrenzt, den es nicht übersteigen kann. Außerdem sinkt es mit einer Verzögerungszeit τ_M wieder gegen Null. Wird

nun die dynamische Schwelle nach einem Impuls erhöht, dann liegt der Gesamtwert der statischen und dynamischen Schwelle T_{ges} so hoch, daß er den Maximalwert des Membranpotentials übersteigt. In dieser Zeit kann kein weiteres Ausgangssignal erzeugt werden. In Abhängigkeit von der Zeit sinkt T_{ges} unter M_{max}, dann erzeugt ein hohes Membranpotential einen Impuls. Erst wenn die dynamische Schwelle wieder bei Null liegt, reicht ein "normales" Membranpotential zur Erzeugung eines Signals aus. Hierdurch werden die beiden Refraktärzeiten eines biologischen Neurons nachgebildet.

4. Closed Loop Antagonistic Network

Wie schon beschrieben, werden die extrahierten Merkmale der aufgenommenen Objekte, die im folgenden auch als Muster bezeichnet werden, durch den neuronalen Musterklassifizierer ***C**losed **L**oop **A**ntagonistic **N**etwork* (*CLAN*) gelernt bzw. wiedererkannt.

Das CLAN ist in der Lage, Muster in wohldefinierte Musterklassen einzuteilen. Hierbei entscheidet es ohne Lehrer, ob für ein Muster bereits eine Klasse existiert und das Muster somit wiedererkannt wird, oder ob das Muster gänzlich unbekannt ist und somit eine neue Musterklasse eröffnet werden muß. Um eine neue Musterklasse zu erzeugen, braucht das CLAN nicht von Erkennen auf Lernen umgeschaltet zu werden, sondern es lernt das Muster automatisch. Nach der einmaligen Repräsentation eines Musters ist dieses gelernt, und es kann bei der nächsten Repräsentation wiedererkannt werden. Es sind keine langwierigen Trainingssequenzen notwendig, wie sie sonst bei neuronalen Netzen (z.B. bei Netzen mit „Backpropagation“ [RUMELHART 1986], [LE CUN 1986] und [PARKER 1985]) üblich sind. „Competitive Learning“ ist im Vergleich zum CLAN gewöhnlich sehr langsam [CARPENTER 1987]. Wenn auch die „Adaptive Resonance Theory“ (ART) zwar ähnliche Komponenten wie das CLAN besitzt, so ist die Implementation des CLANs nicht so umfangreich [GROSSBERG 1976].

An dieser Stelle sei aber auch erwähnt, daß einzelne Prinzipien des CLANs bereits bekannt sind. Die antagonistische Repräsentation wurde schon von Willshaw [WILLSHAW 1972] und das Gegenspiel von positiven und negativen Gewichten bereits von Kohonen [KOHONEN 1984] benutzt. Auch der Wettbewerb mit eingestreuten inhibitorischen Neuronen wurde von Malsburg bereits erwähnt [MALSBURG 1973], und die „closed loop“-Architektur wurde von Palm untersucht. Der Vorteil und die Besonderheit des CLANs resultieren aus der Kombination dieser doch sehr unterschiedlichen Komponenten.

4.1 *Qualitätsmaß* des CLANs

Um eine Klassifizierung vornehmen zu können, ist es zuerst notwendig, ein Maß für die Qualität der Übereinstimmung zu definieren, mit dem verschiedene Muster in Klassen aufgeteilt werden können. Die Qualität der Übereinstimmung von zwei Mustern L und P wird als gut bezeichnet, wenn ein hoher Prozentsatz ihrer Komponenten überlappt. Diese Tatsache wird im folgenden in eine mathematische Formel umgesetzt. Außerdem wird darauf geachtet, daß sich das so ermittelte Qualitätsmaß durch ein neuronales Netz realisieren läßt.

Es sei $f = (f_1, f_2, \ldots f_i \ldots f_f)$ der Vektor eines Merkmalraums, in dem f verschiedene Merkmale beschrieben werden. Die Menge $F = \{f_i \mid 1 \leq i \leq f\}$ beschreibt die Menge der Merkmale. Jede Komponente f_i wird durch ein Neuron der Eingangsschicht in einem neuronalen Netz beschrieben. Zur Vereinfachung werden nur binäre Neuronen betrachtet. Dies heißt, daß die Merkmale $f_i \in \{0, 1\}$ sind und ein entsprechendes Neuron aktiv oder passiv ist.

Ein präsentiertes Muster P mit p aktiven Neuronen wird durch die Menge

$$P = \{f_i^{(P)} \mid f_i^{(P)} = 1;\ 1 \leq i \leq f\} \tag{4.1}$$

beschrieben. In gleicher Weise wird ein gelerntes Muster L von l aktiven Neuronen durch

$$L = \{f_i^{(L)} \mid f_i^{(L)} = 1;\ 1 \leq i \leq f\} \tag{4.2}$$

definiert.

Zur Verdeutlichung der nachfolgenden Betrachtung werden die Muster als zweidimensionale Mengen dargestellt (Fig. 4.1). Es seien P und L zwei Eingabemuster, die Untermengen von F sind. Die Schnittmenge $M = P \cap L$ ist die Übereinstimmung der beiden Muster. Sie enthält m Elemente.

Es wäre möglich, m als ein Maß der Übereinstimmung beider Muster anzusehen. Doch m ist noch kein gutes Maß für die Gleichheit von zwei verschiedenen Mustern, denn die Nichtübereinstimmung beider Muster L\M und P\M wird nicht berücksichtigt. Dies ist in Fig. 4.1 zu sehen: Obwohl die Muster L und P eine gleich große Schnittmenge M wie die Muster L^* und P^* haben, würde der Beobachter bei der Betrachtung der Muster sofort sagen, daß die Muster in Fig. 4.1d größere Ähnlichkeit miteinander haben. Dies liegt daran, daß die Nichtübereinstimmung L^* \M und P^* \M kleiner ist als bei den ersten beiden Mustern.

Für ein gutes Qualitätsmaß gilt somit, daß nicht nur die Übereinstimmung m beider Muster zu berücksichtigen ist. Vielmehr muß auch die Nichtübereinstimmung der aktiven Komponenten $|L\backslash M| = (l-m)$ und $|P\backslash M| = (p-m)$ mit beachtet werden. Dies führt zu dem Qualitätsmaß:

$$\begin{aligned} q(P,L) &= m - [(p - m) + (l - m)] \\ &= 3m - p - l \end{aligned} \tag{4.3}$$

Dieses Qualitätsmaß ist hoch, wenn ein großer Prozentsatz von Muster L mit Muster P übereinstimmt.

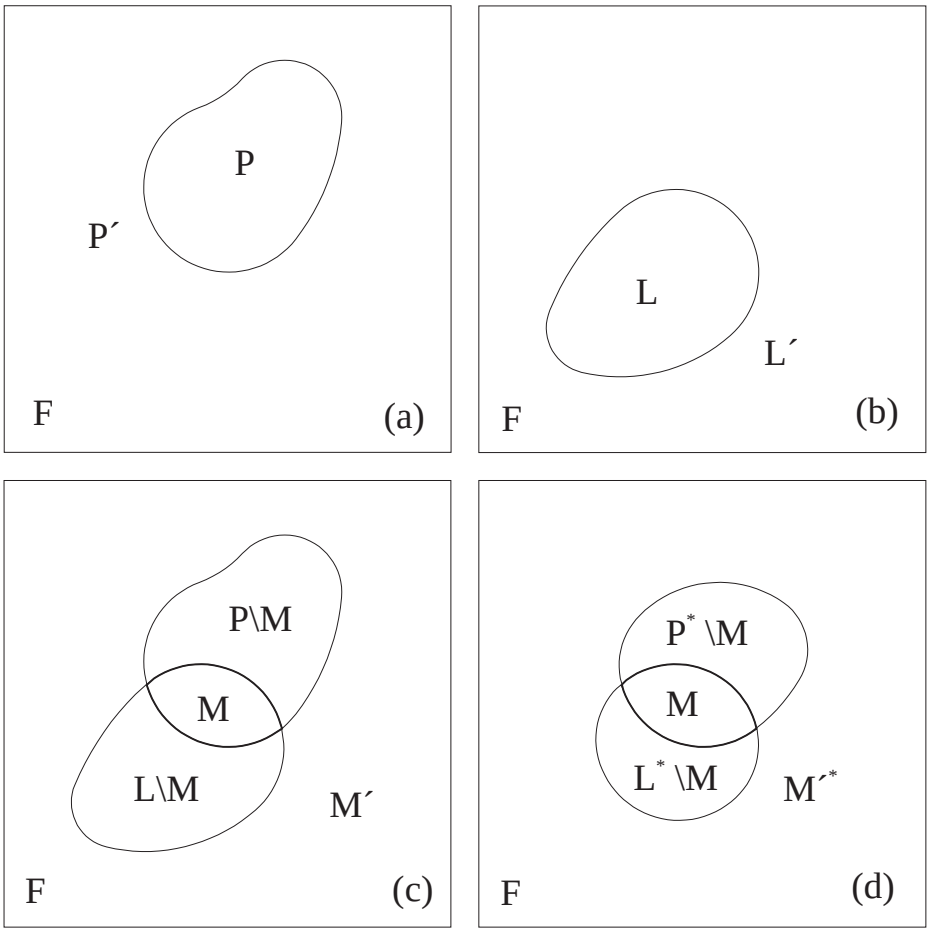


Fig. 4.1: Veranschaulichung der verschiedenen Übereinstimmungen.

4.2 Neuronale Architektur

Das Qualitätsmaß läßt sich durch ein neuronales Netz berechnen. Ein einzelnes Neuron kann in Abhängigkeit von q(P,L) aktiviert werden, wenn es in einer speziellen Architektur eingebettet ist.

Zunächst wird von einem einfachen neuronalen Netz ausgegangen, wie es in Fig. 4.2 zu sehen ist. Die Gruppe der F-Neuronen repräsentiert den Merkmalsraum, wie er beschrieben wurde. Ein Neuron ist aktiv (grau in Fig. 4.2), wenn dessen entsprechendes Merkmal vorhanden ist; ansonsten ist es passiv. Alle F-Neuronen sind mit allen A-Neuronen der assoziativen Schicht A verbunden. Jede Verbindung zwischen einem F-Neuron F_i und einem A-Neuron A_j hat den Gewichtsfaktor w_{ji}, und alle Gewichte zusammen ergeben den Gewichtsvektor w_{AF}.

Die Gewichte werden am Anfang auf einen Wert w gesetzt. Wird nun ein Muster L präsentiert, verändern sich die Gewichte, sobald ein Neuron A_j der Schicht A dieses Muster lernt. Beim Lernen des Musters durch ein Neuron A_j werden alle Eingangsgewichte des Neurons A_j, die mit einem aktiven Neuron der Eingabeschicht F verbunden sind, bis auf einen Maximalwert $w_{ji} = W > w$ erhöht. Alle anderen Gewichtsfaktoren des Neurons A_j werden immer kleiner, bis die entsprechenden Gewichte $w_{ji} = 0$ sind.

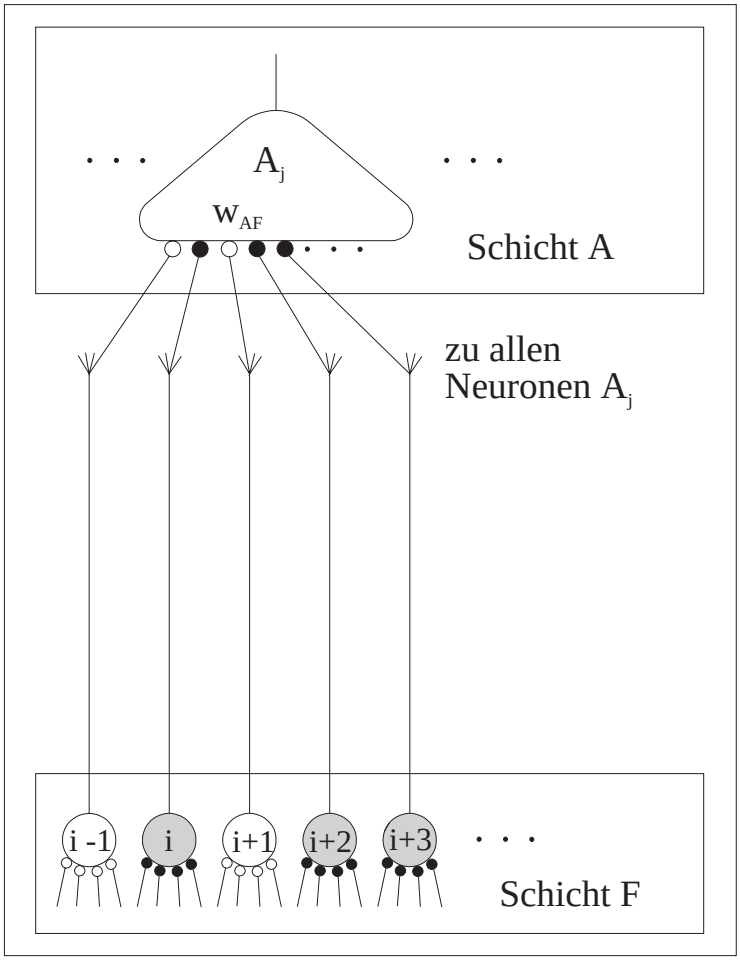


Fig. 4.2: Eine zweischichtige Architektur, die die Übereinstimmung zweier Muster berechnet.

Somit besitzen nur die Gewichte derjenigen F-Neuronen zu dem adaptierten Neuron A_j, an deren Eingang eine positive Komponente des Vektors f anliegt, einen Faktor größer Null. Dies entspricht genau den Elementen von L. Der gelernte Gewichtsvektor ergibt sich zu

$$w_{AF}^{(L)} = W \cdot f^{(L)}. \qquad (4.4)$$

Hieraus folgt, daß das Neuron A_j auf ein neu angelegtes Muster P mit einem Wert, der als *Aktivität* bezeichnet wird, von

$$a_j(P|L) = w_{AF}^{(L)} \cdot f^{(P)} = W \cdot f^{(L)} \cdot f^{(P)} = W \cdot m \qquad (4.5)$$

antwortet.

In dieser Architektur wird also nur die Übereinstimmung, der Match, m zwischen zwei Mustern bestimmt. Die Nichtübereinstimmung beider Muster findet keine Berücksichtigung. Durch die Architektur des CLANs läßt sich dieses Problem lösen.

4.3 Architektur des CLANs

Es wird eine zusätzliche Eingangsschicht von Neuronen, die F'-Neuronen, eingeführt. Die neue Eingangsschicht hat die gleiche Größe wie die Schicht F. Jedes Neuron in der F'-Schicht ist einem Neuron in der F-Schicht zugeordnet. Sie weisen ein antagonistisches Verhalten zu den F-Neuronen auf, d. h. ist ein F-Neuron aktiv, ist dessen Gegenpart in der F'-Schicht nicht aktiv, und umgekehrt.

Hierdurch wird ein Muster L nicht nur durch die positiven Komponenten in $f^{(L)}$ repräsentiert, sondern zusätzlich durch eine Menge L' = F\L positiver Komponenten in $f'^{(L)}$. Ein Muster besteht nun aus den beiden Vektoren $f^{(L)}$ und $f'^{(L)}$ mit insgesamt 2f Komponenten.

Analog zur Schicht F gibt es für die Gewichte eines Neurons A_j zu allen F'-Neuronen den Gewichtsvektor $w_{F'}$. Die Anfangsgewichte erhalten ebenfalls den Initialwert w. Wird ein Muster L durch das Neuron A_j gelernt, vergrößern sich alle Gewichte schrittweise auf den Maximalwert $w_{ji} = W > w$. Alle anderen Gewichte zu dem Neuron A_j verkleinern sich langsam, bis $w_{ji} = 0$ ist. Es ergibt sich der Gewichtsvektor

$$w_{AF'}^{'(L)} = W \cdot f'^{(L)}. \qquad (4.6)$$

Bevor nun die beiden Vektoren kombiniert werden, wird das Gewicht w_{AF} verdoppelt. Ein Neuron A_j, das das Muster L gelernt hat, reagiert mit folgender Aktivität auf ein präsentiertes Muster P:

$$\begin{aligned} a_j(P|L) &= 2w_{AF}^{(L)} \cdot f^{(P)} + w_{AF'}^{'(L)} \cdot f'^{(P)} \\ &= 2W \cdot f^{(L)} \cdot f^{(P)} + W \cdot f'^{(L)} \cdot f'^{(P)} \\ &= 2W \cdot m + W \cdot m' \end{aligned} \qquad (4.7)$$

Der Term 2W·m berechnet die Überlappung der beiden Muster L und P, während der Term $W \cdot m'$ die Überlappung der aktiven Komponenten, die nicht in P und L liegen, beschreibt. Für die Bestimmung von m' ergibt sich die folgende Betrachtung:

$$\begin{aligned} M' &= (F \backslash P) \cap (F \backslash L) \\ &= (F \backslash (P \cup L)) \\ &= (F \backslash (P \cup (L \backslash M)) \end{aligned} \tag{4.8}$$

Es gilt somit m' = f - (p + (l - m)). Ersetzt man nun m', so erhält man den Term

$$\begin{aligned} a_j(P|L) &= 2W \cdot m + W \cdot (f - (p + (l - m))) \\ &= W \cdot (3m - p - l) + W \cdot f \end{aligned} \tag{4.9}$$

Mit Formel 4.3 ergibt sich:

$$a_j(P|L) = W \cdot q(P|L) + W \cdot f \tag{4.10}$$

Diese Formel berechnet die gewünschte Aktivität bis auf den zusätzlichen Term W · f. Bei der Betrachtung von binären Neuronen ist dieser zusätzliche Faktor konstant. Er kann deshalb durch eine entsprechende Schwelle oder durch Anpassung der Gewichtsfaktoren kompensiert werden. Es ist aber auch möglich, den Term W · f durch eine zusätzliche Schicht von hemmenden I-Neuronen abzuziehen. In Fig. 4.3 ist ein solches I-Neuron eingezeichnet. Es erhält seinen Input von den F- und F'-Neuronen. Da entweder ein Neuron in der Schicht F oder sein Partner in der F'-Schicht aktiv ist, ist die Eingangsaktivierung der I-Neuronen proportional zu f.

Der benötigte hemmende Input der A-Neuronen kann aber nicht von einem einzelnen Neuron erzeugt werden. Deshalb wird eine Menge von I-Neuronen verwendet, die durch den Vektor $s = (s_1, \ldots, s_k, \ldots)$ beschrieben werden. Auf der Eingangsseite sind sie vollständig mit der F- und F'-Schicht verschaltet und erhalten alle den gleichen Input.

Auf der Ausgangsseite werden sie durch die Gewichtsvektoren w_{AI} der Gewichte w_{jk} von jedem Neuron I_k zu jedem A-Neuron beschrieben. Die Gewichte w_{jk} werden am Anfang auf den Initialwert w gesetzt. Beim Lernen durch das Neuron A_j erhöhen sich die Gewichte von allen I-Neuronen zu dem lernenden Neuron auf den Maximalwert $w_{jk} = W > w$. Es werden alle Gewichte erhöht, da alle I-Neuronen aktiv sind. Die Gewichte wirken negativ auf das A-Neuron.

Wählt man nun

$$w_{AI}^{(L)} = w \cdot s^{(L+L')} = W \cdot f, \qquad (4.11)$$

wird der zu große Term kompensiert. Dadurch erzeugt das Neuron A_j im beschriebenen Netzwerk eine Aktivität, die proportional zum Qualitätsmaß

$$\begin{aligned} a_j(P|L) &= 2w_{AF}^{(L)} \cdot f^{(P)} + w'^{(L)}_{AF'} \cdot f'^{(P)} - w_{AI}^{(L)} \cdot s \\ &= W \cdot q(P,L) \end{aligned} \qquad (4.12)$$

ist.

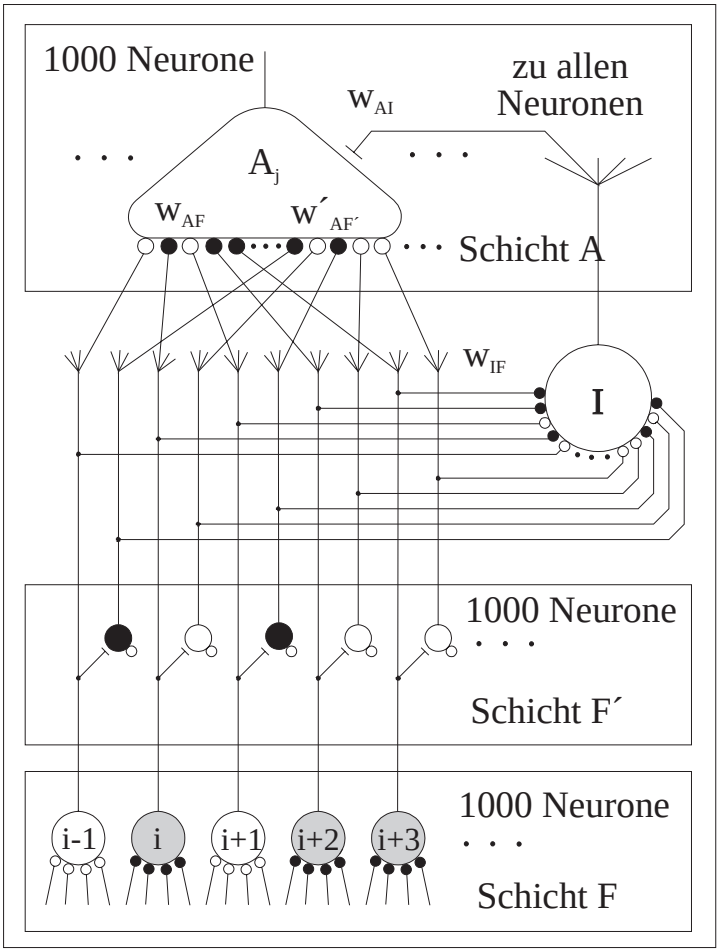


Fig. 4.3: Die Architektur des CLANs.

4.4 Aktivität aller A-Neuronen

Nun gilt es, die Aktivität eines A-Neurons, das ein Muster gelernt hat, mit einem A-Neuron zu vergleichen, das kein Muster gelernt hat. Nach der bisherigen Betrachtung ergibt sich die Aktivität eines Neuron A_j zu

$$\begin{aligned} a_j(P|L) &= W \cdot q(P,L). \\ &= W \cdot [m - (p - m) - (l - m)] \end{aligned} \qquad (4.13)$$

Im Spezialfall, d. h. wenn das Muster L vom Neuron A_j gelernt und das gleiche Muster nochmals als Muster P präsentiert wurde, gilt l = p = m. Das Neuron A_j reagiert mit einer Aktivität von

$$a_j(P|P) = W \cdot p \qquad (4.14)$$

Wurde vom Neuron A_j noch kein Muster gelernt, so sind alle Anfangsgewichte zu den Eingangsschichten F, F' und I noch um den Initialwert gestreut, und es ergibt sich die Aktivierung

$$\begin{aligned} a_j(P|0) &= 2w_{AF} \cdot f^{(P)} + w'_{AF'} \cdot f'^{(P)} + w_{AI} \cdot s \\ &= 2w \cdot p + w \cdot (f - p) - w \cdot f \\ &= w \cdot p \end{aligned} \qquad (4.15)$$

Um die Aktivitäten zu vergleichen, wird das normierte Qualitätsmaß

$$Q(P|L) = q(P|L)/p \qquad (4.16)$$

eingeführt.

Wenn im folgenden von Qualitätsmaß gesprochen wird, ist immer das normierte Qualitätsmaß gemeint.

Falls das gelernte Muster L mit dem präsentierten Muster P übereinstimmt, ist das Qualitätsmaß

$$\begin{aligned} Q(P|P) &= q(P|P)/p \\ &= p/p \\ &= 1 \end{aligned} \qquad (4.17)$$

Für den Fall, daß noch kein Muster gelernt wurde, ergibt sich

$$\begin{aligned} Q(P,0) &= q(P|0)/p \\ &= a_j(P|0)/W \cdot 1/p \\ &= w \cdot p/(W \cdot p) \\ &= w/W. \end{aligned} \qquad (4.18)$$

Sind die Muster sehr unterschiedlich, wird das Qualitätsmaß negativ.

4.5 Lernschwelle

Um Muster mit dem vorgestellten Netzwerk zu klassifizieren, muß ein Muster wiedererkannt werden, wenn es mit einem gelernten gut übereinstimmt. Auf der anderen Seite muß ein unbekanntes Muster als ein neues Muster erkannt werden. Hierzu ist ein Umschaltmechanismus notwendig, der sich auf folgende Weise realisieren läßt: Falls ein Neuron mit einem Qualitätsmaß

$$Q(P,L) > Q(P,0) = w/W \tag{4.19}$$

existiert, wird das Muster wiedererkannt. Existiert aber kein Neuron mit

$$Q(P,L) > w/W, \tag{4.20}$$

wird das neue Muster gelernt. Der Wert Q(P,0) = w/W wird als *Lernschwelle* bezeichnet.

Bei diesem Verfahren ist kein explizites Umschalten des Netzwerkes zwischen Lernen und Wiedererkennen notwendig. Das Netz erkennt selbständig, ob die Aktivität eines A-Neurons größer als die Lernschwelle ist, und kann dann entscheiden, ob das präsentierte Muster bekannt ist oder neu gelernt werden muß.

4.6 Rückkopplung

In dem beschriebenen Netzwerk reagieren die Neuronen der assoziativen Schicht auf ein präsentiertes Muster. Ohne eine zusätzliche Maßnahme würden aber nach kurzer Zeit alle Neuronen aktiv und dann mit ihrer maximalen Frequenz feuern, so daß es keinen Unterschied mehr zwischen den Aktivitäten der einzelnen Neuronen gäbe. Es könnte also kein Gewinnerneuron ermittelt werden, das das Muster lernt bzw. wiedererkennt. Um dies zu verhindern, wird eine zusätzliche Neuronenschicht C in das Netz integriert, die eingangsseitig vollständig mit den A-Neuronen vernetzt ist, während der Output hemmend auf alle A-Neuronen wirkt (Fig. 4.4). Alle Verbindungen zwischen den beiden Schichten sind immer gleich gewichtet, d.h. sie werden beim Lernen nicht verändert.

Um die Funktion dieser Rückkopplungsschleife zu erklären, wird angenommen, daß das Neuron A_j die höchste Aktivität besitzt, wenn das Muster P präsentiert wird. Das Membranpotential V_j des Neurons A_j berechnet sich aus dem Input der Schichten F, F' und I (p_j) und dem negativen Input der Rückkopplungsschicht C (p_f). Es gilt:

$$V_j = p_j - p_f \tag{4.21}$$

Übersteigt das Membranpotential V_j die dynamische Schwelle T, wird das Neuron A_j überschwellig und erzeugt einen Output der Größe $a_j(P|L) \sim (V_j - T)$. Ein Neuron, dessen Potential $V < T$ ist, erzeugt keinen Output.

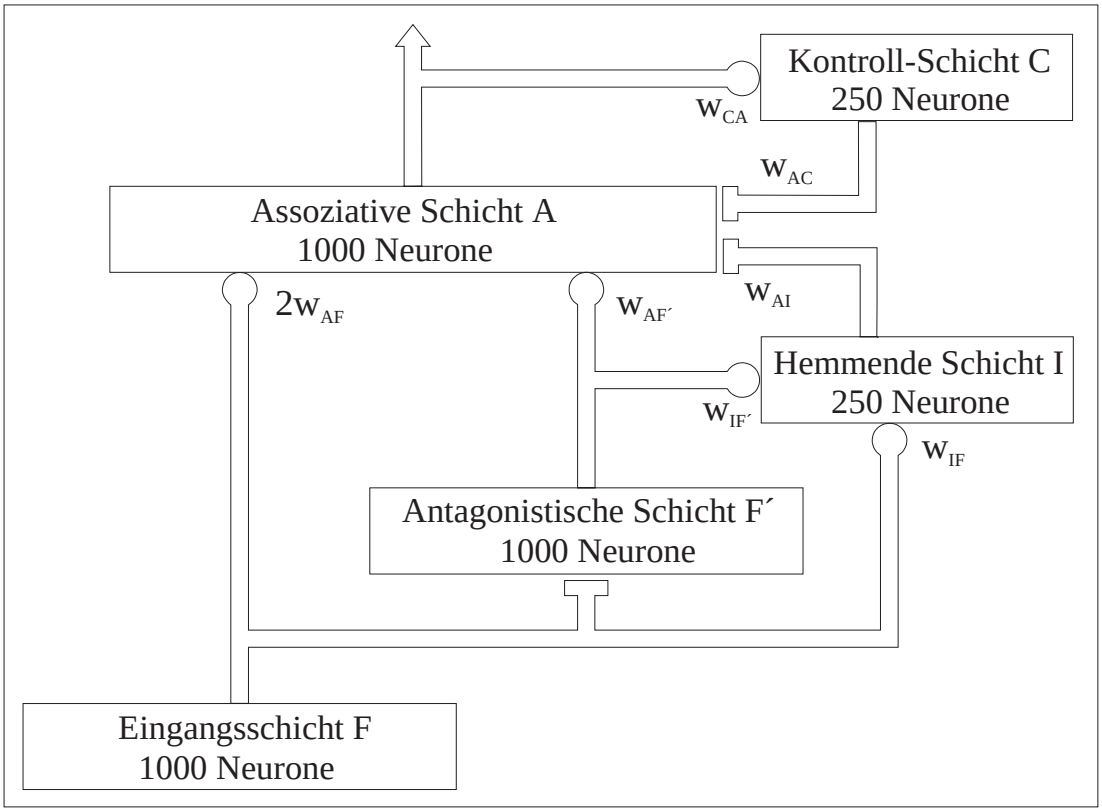


Fig. 4.4: Blockschaltbild des CLANs

Anfangs gibt es mehrere Neuronen, die überschwellig werden. Die C-Neuronen werden kräftig angeregt und erzeugen eine hohe Spikerate. Diese läßt das Membranpotential aller A-Neuronen wieder unter die Schwelle sinken. Hierdurch werden die C-Neuronen wiederum nicht mehr so stark angeregt, so daß nach kurzer Zeit das Potential der C-Neuronen sinkt und die A-Neuronen nicht mehr so stark gehemmt werden. Nun werden die A-Neuronen wieder überschwellig und der Vorgang beginnt von neuem. Allmählich kristallisiert sich ein einzelnes A-Neuron heraus, dessen Aktivität größer als die Aktivität aller anderen A-Neuronen ist, und nach einer Lernverzögerung beginnt dieses A-Neuron das Muster zu lernen. Hierdurch ist es in der Lage, die C-Neuronen so zu stimulieren, daß die Membranpotentiale alle anderen Neuronen unter der Schwelle bleiben und sogar gegen Null sinken. Es selber besitzt aber ein Membranpotential, das größer als die Schwelle ist. Es gilt dann:

$$V_j > T > V_i \quad \text{für alle } i \neq j \,. \tag{4.22}$$

Die beschriebene Theorie wurde in ein Programm umgesetzt. Die Versuchsreihen bestätigten die Annahmen. In folgenden sind die Potentialverläufe beim Lernen und Wiedererkennen an Hand von drei Beispielen aufgezeichnet.

In Fig. 4.5 ist beispielhaft der Potentialverlauf und die Anzahl der Spikes von drei A-Neuronen beim Lernen eines Musters zu sehen. Zu Beginn wächst das Membranpotential aller drei Neuronen stetig an. Neuron 20 und 8 werden fast gleichzeitig aktiv und feuern, während das Potential von Neuron 67 unter der Schwelle bleibt. Durch die Erregung der C-Neuronen sinkt das Potential aller Neuronen, um nach kurzer Zeit wieder anzusteigen. Während Neuron 20 danach konstant überschwellig bleibt und das Muster lernt, feuert Neuron 8 noch fünf mal, bevor sein Potential gegen Null sinkt. Neuron 67 wird überhaupt nicht überschwellig, und sein Potential sinkt sehr schnell gegen Null, wenn Neuron 20 anfängt zu lernen.

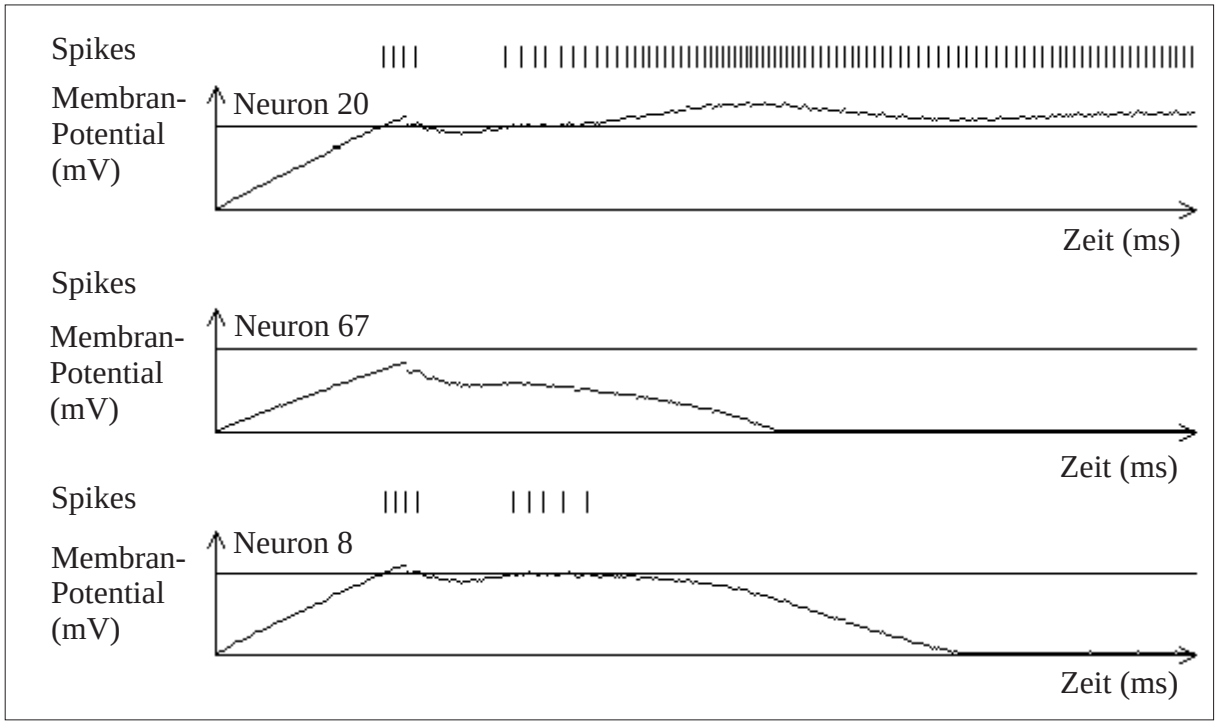


Fig. 4.5: Potentialverlauf und Anzahl der Spikes von drei A-Neuronen beim Lernen eines unbekannten Musters.

Legen wir nun ein neues unbekanntes Muster P an, so verändert sich das normierte Qualitätsmaß des vorher aktiven Neurons A_j von Q(L,L) zu Q (P,L). Das Maß Q(P,L) ist sehr klein, wenn beide Muster sehr verschieden sind. Es gibt andere Neuronen im Netzwerk, die noch kein Muster gelernt haben und deren Aktivität Q(P,0) wesentlich größer ist. Aus diesem

Grund verringert sich das Membranpotential des Neurons A_j schlagartig und alle Neuronen, die noch kein Muster gelernt haben, treten in einen Wettkampf um das neue Muster. Das Neuron A_i, das die beste Anfangsinitialisierung Q(P,0) hat, wird als erstes aktiv. Nach einer kurzen Einschwingphase, wie bei dem ersten Lernvorgang, bleibt nur ein Neuron aktiv, welches dann das Muster lernt.

Dieses Verhalten ist in Fig. 4.6 zu beobachten. Neuron 60 hat zu Beginn das angelegte Muster adaptiert, bis ein neues unbekanntes Muster angelegt wird. Dann sinkt das Membranpotential von Neuron 60 sehr schnell gegen Null, und das Membranpotential der anderen beiden Neuronen steigt. Nach einer kurzen Einschwingphase lernt Neuron 90 das Muster.

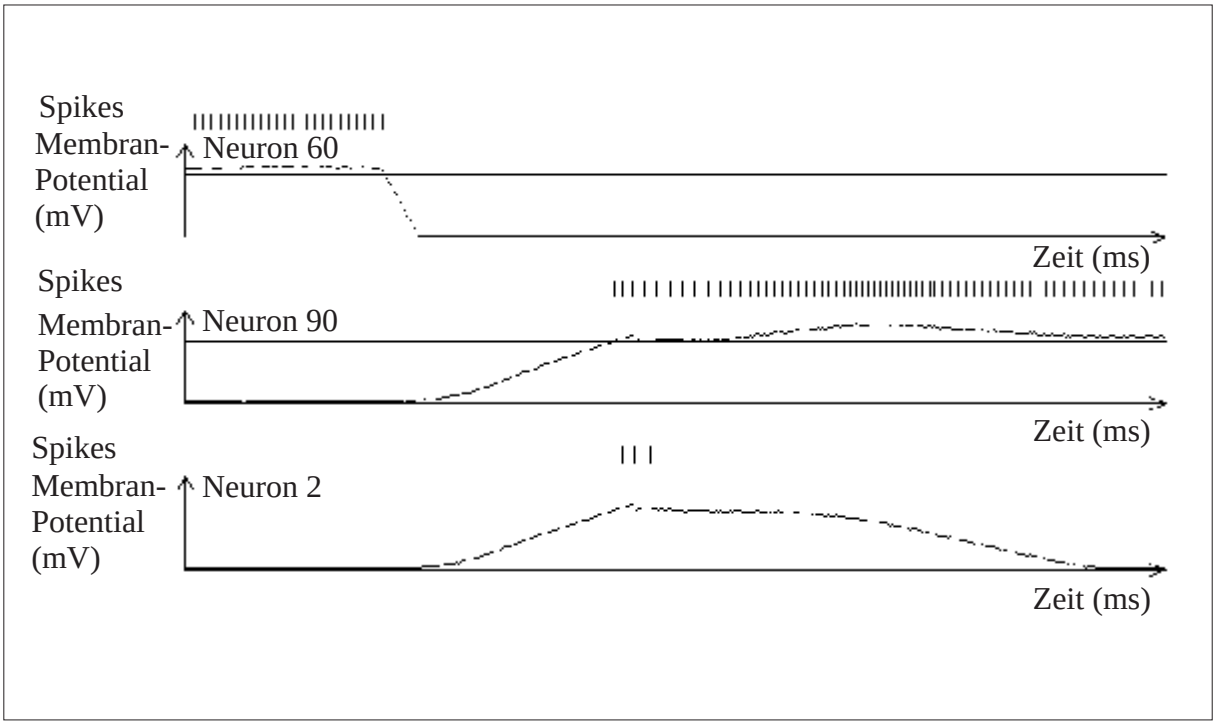


Fig. 4.6: Potentialverlauf und Anzahl der Spikes von drei A-Neuronen beim Lernen eines zweiten unbekannten Musters.

Wird dem Netzwerk ein Muster angeboten, welches es bereits durch das Neuron A_j gelernt hat, steigt das Membranpotential von A_j schneller an als das Membranpotential aller anderen Neuronen. Nach kurzer Zeit wird es überschwellig und feuert mit einer hohen Frequenz. Dadurch wird die C-Schicht sehr stark erregt, und das Potential aller Neuronen der A-Schicht wird um den Faktor net_f reduziert. Die Hemmung ist so hoch, daß nur Neuron A_j aktiv bleibt. Das Membranpotential aller anderen Neuronen sinkt gegen Null. Das Muster wird wiedererkannt.

Der Potentialverlauf beim Wiedererkennen ist in Fig. 4.7 zu sehen. Zuerst wird durch Neuron 6 ein Muster gelernt. Nun wird ein bekanntes Muster gezeigt. Schlagartig sinkt das Potential von Neuron 6, während das Membranpotential von Neuron 82 sehr stark ansteigt. Das Potential von Neuron 65, das bisher noch kein Muster gelernt hat, steigt hingegen nur langsam. So wird das Neuron 82 schnell überschwellig und erkennt das gezeigte Muster. Das Potential von Neuron 65 erreicht die Schwelle nicht und sinkt wieder gegen Null, nachdem Neuron 82 sehr stark gefeuert hat.

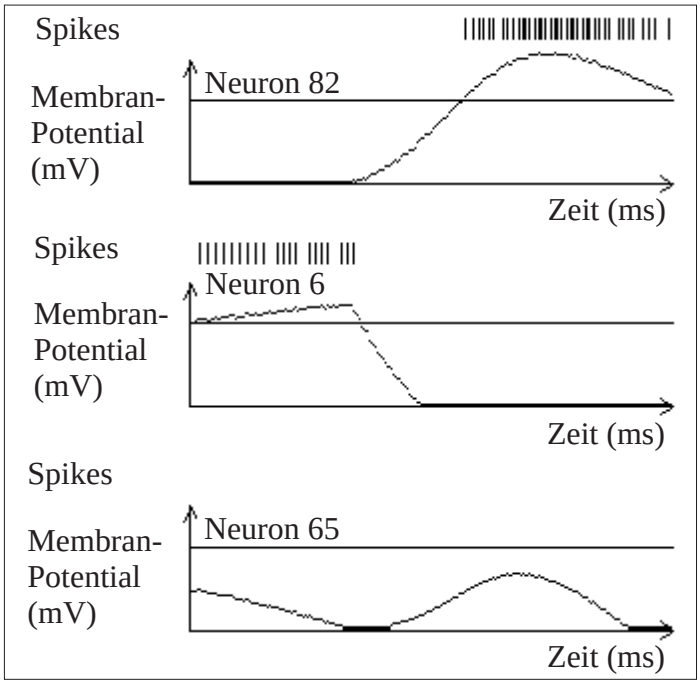


Fig. 4.7: Potentialverlauf und Anzahl der Spikes von drei A-Neuronen beim Wiedererkennen des Musters durch Neuron 82.

4.7 Lernen im CLAN

In der vorliegenden Arbeit wurde schon mehrmals darauf hingewiesen, daß ein Neuron ein Muster lernt. Nachfolgend werden alle Aussagen zusammengefaßt und die genaue Lernregel definiert.

Ein Muster soll durch das A-Neuron mit der größten Aktivität gelernt werden. Doch da alle A-Neuronen unabhängig voneinander sind, kann ein einzelnes Neuron in der vorgestellten Architektur nicht erkennen, ob es das aktivste Neuron ist. Ein Neuron weiß nur, ob es aktiv oder passiv ist. Auf diese Weise wird das Lernen gesteuert.

Finge gleich jedes überschwellige A-Neuron an, das Muster zu lernen, würde das Muster durch mehrere Neuronen gleichzeitig gelernt, da in der Initialisierungsphase erst mehrere A-Neuronen aktiv sein müssen, um die Kontrollschicht C überhaupt zu erregen. Aus diesem

Grund beginnt ein Neuron A_j nicht gleich zu lernen, wenn es überschwellig wird. Es wurde eine Lernverzögerung eingebaut. Ein Neuron der A-Schicht muß erst eine Zeit lang aktiv sein, bevor es lernen kann. Durch diesen Mechanismus wird erreicht, daß ein einzelnes Neuron das Muster lernt. Nur in wenigen Ausnahmefällen, wenn die zufällige Initialisierung zweier A-Neuronen fast gleich ist, wird ein Muster durch zwei Neuronen gelernt.

Wenn nun das Neuron A_j ein Muster lernt, werden die Gewichte aller Neuronen der Schichten F, F' und I zu dem Neuron A_j verändert, während alle Verbindungen von den Neuronen der Schichten F, F' und I zu allen A-Neuronen sowie die Verbindungsgewichte zwischen der A- und C-Schicht unverändert bleiben.

Das Lernen erfolgt mit einer leicht abgeänderten *Hebbschen Regel*. Diese besagt, daß das Gewicht zwischen zwei Neuronen vergrößert wird, wenn sie zugleich wiederholt stark aktiv sind [HEB1949]. Im CLAN werden zusätzlich alle passiven Verbindungen zum aktiven Neuron A_j verlernt. Die Lernregel sieht folgendermaßen aus:

$$\Delta w_{ij} = \begin{cases} -\alpha & \text{wenn } 0 < w_{ij} < W \text{ und } V_j > T_k \text{ und Neuron i feuert nicht} \\ \beta & \text{wenn } 0 < w_{ij} < W \text{ und } V_j > T_k \text{ und Neuron i feuert} \\ 0 & \text{sonst} \end{cases}$$

Solange das Membranpotential eines Neurons A_j größer als die konstante Schwelle T_k = -40 mV ist, werden alle Synapsengewichte, die mit einem aktiven F-, F'- oder I-Neuron verbunden sind, erhöht. Alle anderen Gewichte zu den passiven Eingangsneuronen werden verringert. Liegt das Potential von A_j unter T_k, werden die Gewichte nicht verändert.

5. Implementierung des Closed Loop Antagonistic Networks

Das beschriebene neuronale Netz wurde auf einer Sun-Workstation implementiert, wobei folgende Konfiguration gewählt wurde:

- F-Schicht 1000 Neuronen
- F'-Schicht 1000 Neuronen
- I-Schicht 250 Neuronen
- C-Schicht 250 Neuronen
- A-Schicht 500 Neuronen

Die Reduzierung des Modells ergab sich aus Platz- und Zeitgründen. Parametersätze wurden ermittelt, und es zeigte sich, daß die Theorie in einen Algorithmus umsetzbar ist. Die folgenden Ausführungen sollen nur eine grobe Übersicht geben. Eine detaillierte Beschreibung des Algorithmus und der gefundenen Parameter findet man in [Kalthoff 1990].

5.1 Simulation

Ein *Simulationsschritt* t_i, der im folgenden auch als *Zeitschritt* bezeichnet wird, modelliert die einmalige Modifizierung aller Neuronen, wodurch jeweils eine Millisekunde (ms) simuliert wird. Die gesamte Simulation arbeitet eine einstellbare endliche Anzahl von Simulationsschritten ab. Als Eingaben werden nur binäre Eingabemuster betrachtet.

Für die Neuronen der A- und C-Schicht werden pulscodierte Neuronen benutzt (vgl. Kapitel 3.4). Jedes dieser Neuronen besitzt Variablen für das Membranpotential M_i und die dynamische Schwelle T_D, außerdem Konstanten für das maximale Membranpotential M_{max}, die statische Schwelle T_S und die beiden Zeitkonstanten τ_M und τ_T. Für alle synaptischen Gewichte zwischen je zwei Layern gibt es variable zweidimensionale Arrays.

In jedem Zeitschritt t_i sinkt das Membranpotential jedes Neurons in Abhängigkeit von der Zeit τ_M. Danach werden die Gewichte aller Eingänge, an denen ein Signal anliegt, zu einer Gesamtpotentialveränderung aufsummiert und zum aktuellen Membranpotential addiert. An einem Eingang liegt ein Signal an, wenn im Zeitschritt t_{i-1} das vorgeschaltete Neuron ein solches versendet hat. Übersteigt das Potential die Summe aus der statischen und dynamischen Schwelle, sendet das betrachtete Neuron ein Signal über seinen Ausgang an alle ver-

bundenen Eingänge. Außerdem wird die dynamische Schwelle dieses Neurons auf ihren Maximalwert gesetzt. Abschließend sinkt die dynamische Schwelle bei allen Neuronen in Abhängigkeit von der Zeit τ_T.

Die Neuronen der Eingangsschichten F, F' und I werden in einer etwas vereinfachten Form als pulscodierte Neuronen simuliert. Das Membranpotential aller F- und F'-Neuronen, deren Komponente im Eingangsvektor auf eins gesetzt ist, sowie das aller I-Neuronen liegt auf einem konstanten Wert über der statischen Schwelle. Die anderen Neuronen der Schichten F und F' haben ein Potential in Höhe des Ruhepotentials. Es wird nur das Zusammenspiel des Membranpotentials und der beiden Schwellen simuliert. Übersteigt also das Membranpotential die Summe aus statischer und dynamischer Schwelle, dann sendet das Neuron ein Signal aus, und die dynamische Schwelle wird auf ihren Maximalwert gesetzt. Außerdem sinkt der Wert aller dynamischen Schwellen in Abhängigkeit von der Zeitkonstanten τ_T.

Die Gewichte aller Verbindungen sind anfangs um den Wert w = W/2 gestreut. Beginnt das Neuron zu lernen, werden die Gewichte langsam nach der vorgestellten Lernregel verändert (vgl. Kapitel 4.6).

Um ein Muster zu lernen, sind ca. 800 bis 1000 Simulationsschritte notwendig. Somit benötigt das Netz ca. 1000 Zeitschritte zum Lernen. Wesentlich weniger Zeit ist erforderlich, um ein Muster wiederzuerkennen. Innerhalb von 300 Schritten kann eindeutig ein Muster wiedererkannt werden.

Auf einer Sparc10 erfordert die Simulation eines Lernvorgangs eines Musters zwischen 4 und 5 Minuten. Dies bedeutet, daß eine Zeitschritt ungefähr in 0,3 s Rechenzeit simuliert werden kann. Die Zeit, um ein Muster zu lernen, liegt unter den Lernzeiten üblicher neuronaler Netze. Dies liegt daran, daß ein Muster nur einmal präsentiert werden muß. Es sind keine aufwendigen Trainingssequenzen notwendig.

5.2 Ergebnisse

Die folgenden Figuren zeigen Grafiken, in denen das Membranpotential aller A-Neuronen in mV aufgetragen ist. In den Grafiken wird das Ruhepotential der Neuronen mit 0 mV angenommen, und die statische Schwelle liegt bei 40 mV.

Wird dem CLAN ein unbekanntes Muster gezeigt, steigt zuerst das Membranpotential aller A-Neuronen an (Fig. 5.1). Nach ca. 110 Zeitschritten übersteigt das Potential einiger Neuro-

nen die statische Schwelle (Fig. 5.2). Durch die anfangs zufällige Streuung der Gewichte ist das Membranpotential der einzelnen Neuronen leicht unterschiedlich. Nun werden die Neuronen der C-Schicht erregt, und diese wirken hemmend auf die A-Neuronen. Hierdurch sinkt und steigt das Potential der A-Neuronen, und die einzelnen Potentialunterschiede vergrößert sich.

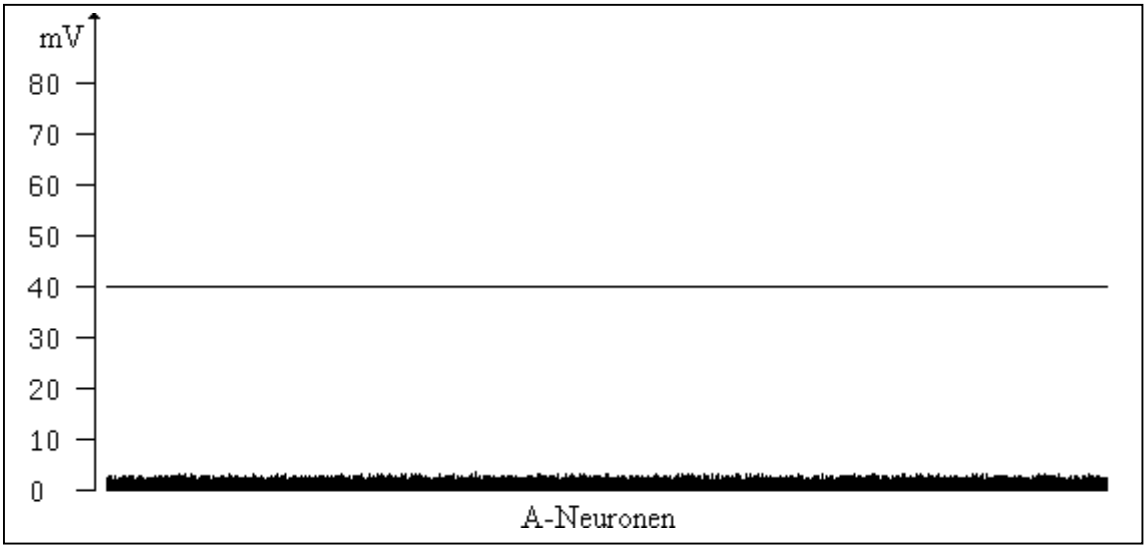


Fig. 5.1: Lernen eines unbekannten Musters. Dargestellt ist das Membranpotential der A-Neuronen nach 10 Zeitschritten.

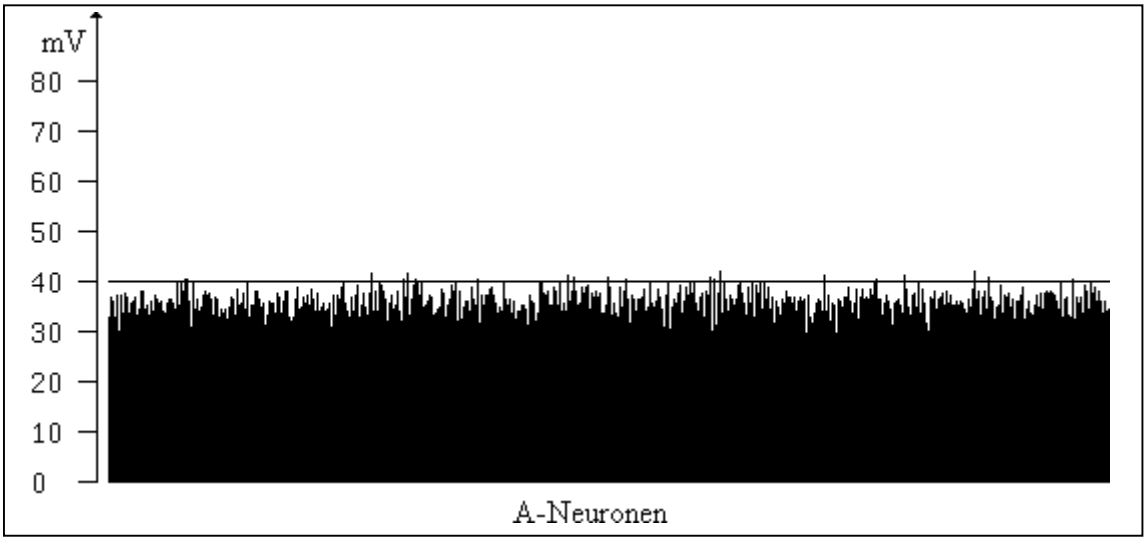


Fig. 5.2: Lernen eines unbekannten Musters. Dargestellt ist das Membranpotential der A-Neuronen nach 100 Zeitschritten.

Nach ca. 300 Zeitschritten ist nur noch das Membranpotential eines Neurons größer als die Schwelle (Fig. 5.3). Dieses Neuron lernt nun das präsentierte Muster. Nach ca. 600 Zeitschritten ist das Membranpotential aller anderen Neuronen fast Null. Es ist nur noch das eine Neuron aktiv (Fig. 5.4). Um das Muster vollständig zu lernen, d. h. seine Gewichte optimal einzustellen, braucht es nun noch zwischen 200 und 400 Schritte.

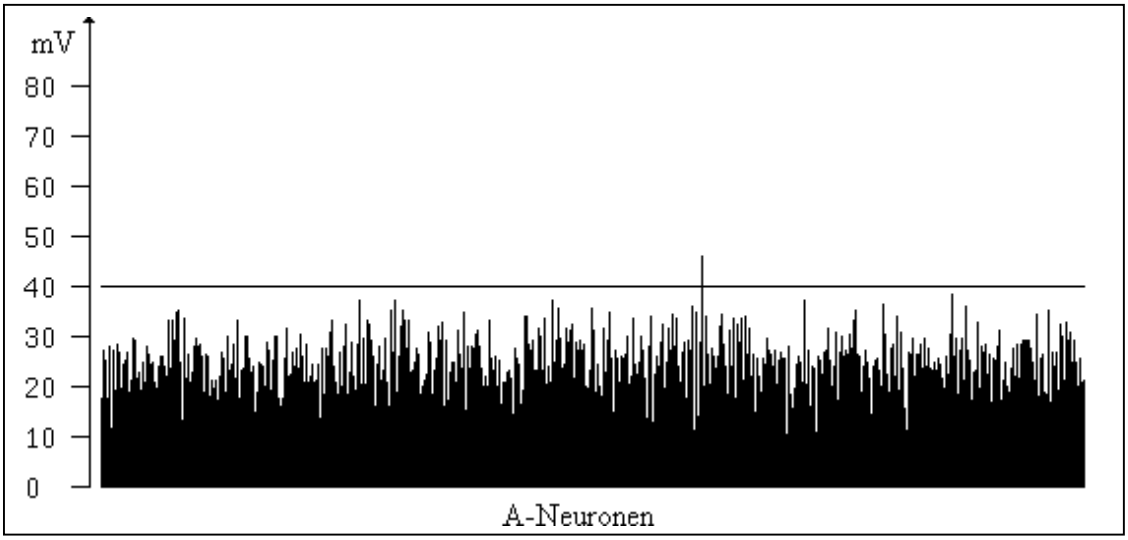


Fig. 5.3: Lernen eines unbekannten Musters. Dargestellt ist das Membranpotential der A-Neuronen nach 300 Zeitschritten.

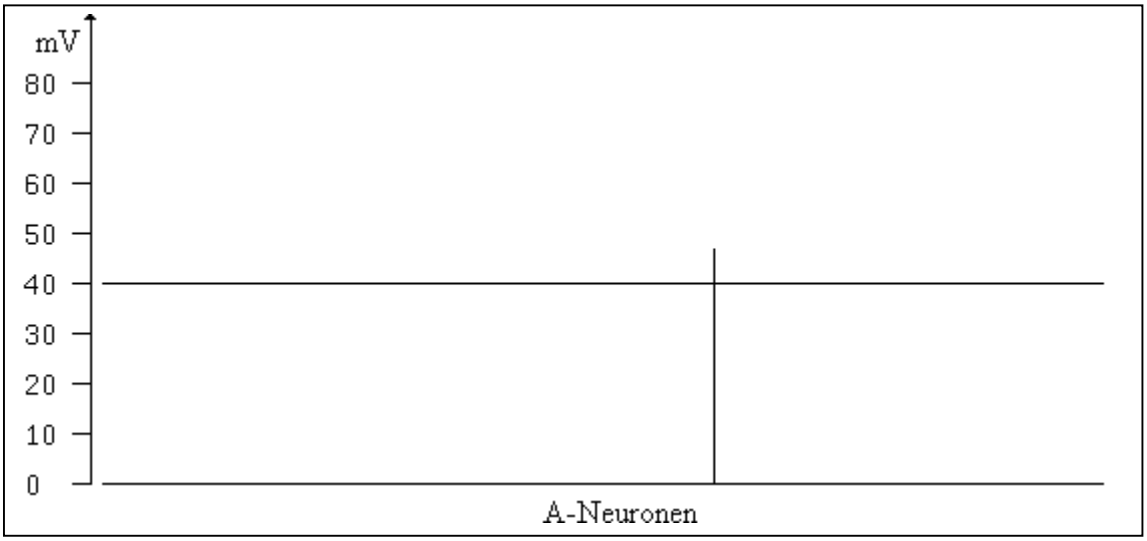


Fig. 5.4: Lernen eines unbekannten Musters. Dargestellt ist das Membranpotential der A-Neuronen nach 600 Zeitschritten.

Wird dem CLAN nun ein bekanntes Muster gezeigt und ist vorher kein A-Neuron aktiv, steigt das Potential aller A-Neuronen langsam an. Nach 10 Schritten ist noch kein gravierender Unterschied zum Lernen eines unbekannten Musters zu sehen (Fig. 5.5). Doch schon nach 110 Zeitschritten ist nur noch das A-Neuron, das das Muster gelernt hat, aktiv. Das Potential aller anderen Neuronen bleibt unter der Schwelle (Fig. 5.6). Nach 300 Schritten ist nur noch das Membranpotential des Neurons, das das Muster gelernt hat, über der Schwelle, während alle anderen Potentiale bei Null liegen (Fig. 5.7). Das Muster ist somit wiedererkannt worden.

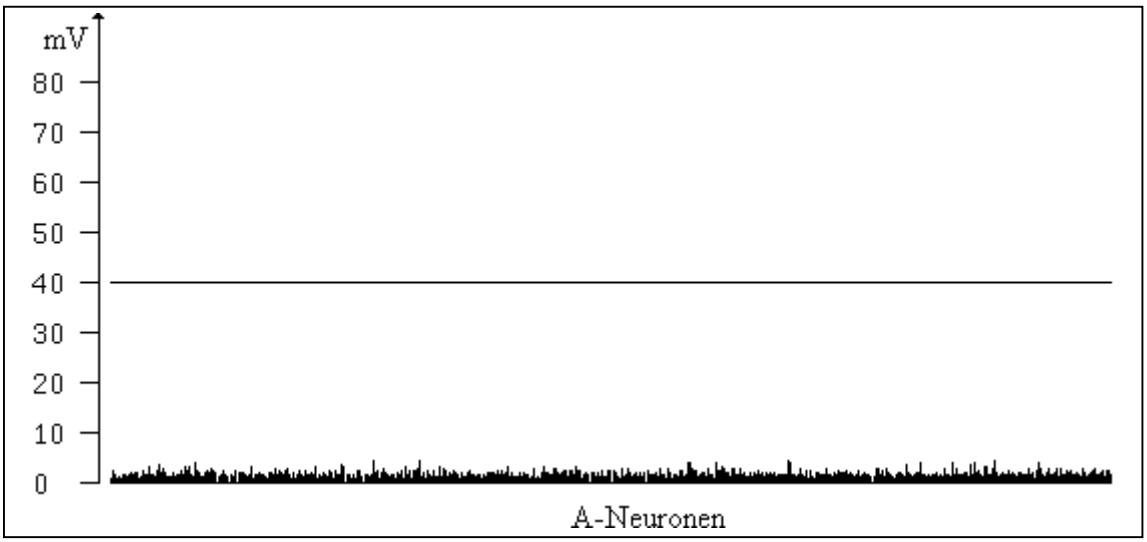


Fig. 5.5: Wiedererkennen eines bekannten Musters. Dargestellt ist das Membranpotential aller A-Neuronen nach 10 Zeitschritten.

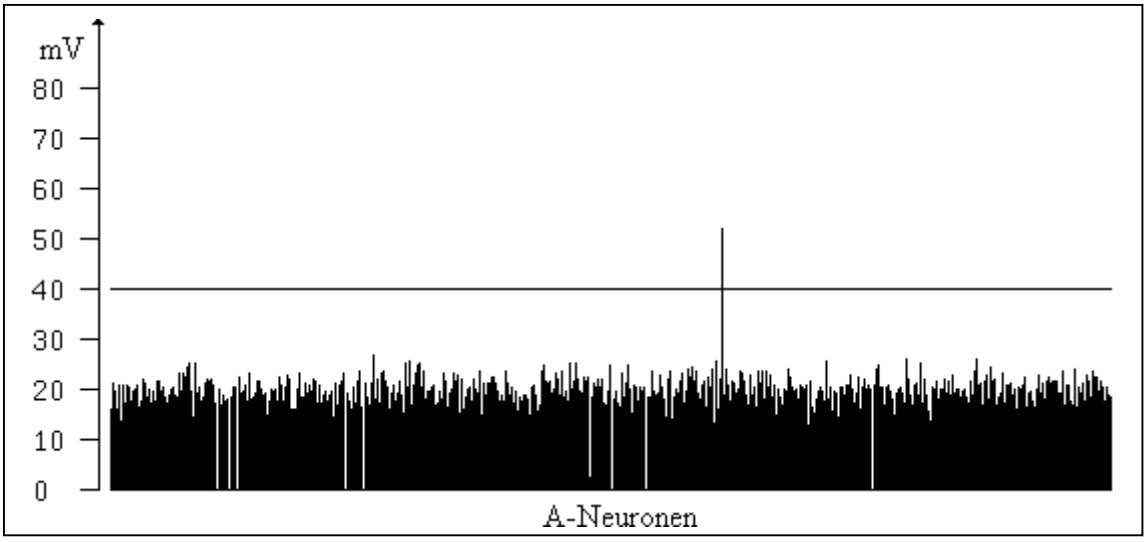


Fig. 5.6: Wiedererkennen eines bekannten Musters. Dargestellt ist das Membranpotential aller A-Neuronen nach 110 Zeitschritten.

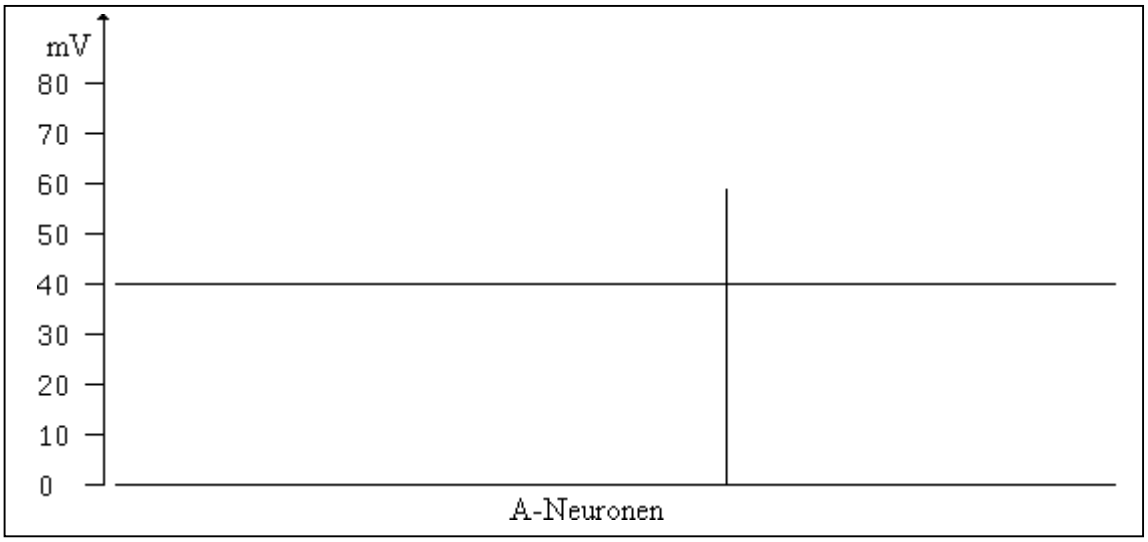


Fig. 5.7: Wiedererkennen eines bekannten Musters. Dargestellt ist das Membranpotential aller A-Neuronen nach 300 Zeitschritten.

5.3 Betrachtung modifizierter Lernstrategien

Hat nun ein Neuron ein Muster gelernt und wird ein ähnliches Muster präsentiert, das in die gleiche Musterklasse gehört, dann erkennt das Neuron das Muster. Beim anschließenden Lernen werden alle Gewichtsfaktoren an das neue Muster angepaßt. Die Struktur des zuerst gelernten Musters geht an den Stellen verloren, an denen sich die beiden Muster unterscheiden. Dadurch verschiebt sich die Musterklasse, die durch das Neuron repräsentiert wird. Geschieht dies mehrere Male für das gleiche Neuron, kann sich die Musterklasse sehr weit von ihrer ursprünglichen Klasse entfernen. Dies kann soweit gehen, daß das erste Muster nicht mehr von dem entsprechenden Neuron erkannt wird.

Durch eine Aufsplittung der Gewichtsfaktoren in ein *langsam* und ein *schnell lernendes* Gewicht wird dieses Verhalten abgeschwächt. Das schnelle Gewicht arbeitet wie das bisherige Gewicht. Die entsprechenden Gewichtsfaktoren werden schon bei der einmaligen Präsentation des Musters (1000 Zeitschritte) auf ihre Endwerte modifiziert.

Die langsamen Gewichte werden nach der gleichen Lernregel verändert, aber die Faktoren α und β werden durch die wesentlich kleiner Werte α' und β' ersetzt. Somit verändern sich die Gewichte nur langsam auf ihre Endwerte. Es reicht nicht mehr aus, ein Muster einmal zu zeigen, sondern das gleiche Muster muß mehrere Male präsentiert werden, um die Gewichte auf die Endwerte zu modifizieren. Wählt man

α' $= \alpha / 8$ und
β' $= \beta / 8$,

dann erreichen die Gewichte für ein Muster erst nach acht Präsentationen ihre Endwerte.

Werden diesem Netzwerk nun mehrere Muster präsentiert, die zwar der gleichen Musterklasse angehören, aber im ersten Fall zu einer Verschiebung der Musterklasse geführt hatten, adaptiert der schnelle Gewichtsvektor zwar immer wieder das aktuelle Muster, doch der langsame Gewichtsvektor ist zu träge, um die Verschiebung mitzumachen. Er bildet einen Gewichtsvektor, in dem von allen präsentierten Mustern ein Teil vorhanden ist. Hierdurch ist es möglich, eine Verschiebung der Musterklassen zu erschweren. Durch diese Vorgehensweise ist es aber nicht möglich, eine generelle Verschiebung der Musterklassen zu verhindern. Durch eine unglückliche Folge von ähnlichen Mustern, die aber dann viel öfter gezeigt werden müßten, kann es trotzdem zu einer Verschiebung kommen [Projektgruppe 1990].

5.4 Feuernde Neuronen als Input

In der ersten Version wurden nur binäre Eingaben für das CLAN betrachtet. Es wurde aber auch untersucht, inwieweit es möglich ist, die Eingabeschichten direkt durch feuernde Neuronen zu aktivieren. Zunächst wird die Dynamik des Membranpotentials der Neuronen der Eingabeschichten F, F' und I ebenfalls implementiert. Der Algorithmus wurde von den A-Neuronen übernommen (siehe Kapitel 5.1).

Es wurde ein Pulsgenerator (*Spikegenerator)* erzeugt, der als Eingabe das angebotene Muster erhält. Er generiert für aktive Musterkomponenten eine Spikerate, die bei ungefähr 100 Hz liegt, und für alle passiven Komponenten eine Spikerate von ca. 10 Hz. Die erzeugten Spikeraten werden auf die Eingangsschicht F gegeben.

Hierdurch werden die Neuronen der F-Schicht angeregt. Neuronen, die mit einem Ausgang des Spikegenerators verbunden sind, der mit einer Frequenz von 100 Hz feuert, werden nach einer kurzen Verzögerungszeit aktiv und feuern dann ebenfalls mit einer Frequenz von ca. 100 Hz. Der Input für die anderen F-Neuronen reicht nicht aus, um sie zu aktivieren, somit erzeugen sie keinen Output.

Alle Neuronen der F'-Schicht sind durch einen zusätzlichen Input während der Initialisierungsphase aktiv. Die aktiven Neuronen der F-Schicht wirken hemmend auf ihren Gegenpart

in der F'-Schicht, so daß diese Neuronen passiv werden. In der F'-Schicht sind anschließend nur noch die Neuronen aktiv, deren Gegenpart in der F-Schicht nicht aktiv ist.

Durch die aktiven Neuronen der F- und F'-Schicht werden die Neuronen der I-Schicht immer konstant aktiviert. Alle drei Schichten erzeugen somit die gleiche Aktivität wie das vorher vorgestellte Netzwerk. Bis auf eine zeitliche Verzögerung zur Aktivierung des gesamten Netzwerks zeigen beide Versionen ein äquivalentes Verhalten [Projektgruppe 1990].

5.5 Parallelisierung des CLANs

Um die Rechenzeit zu verringern, wurde das System parallelisiert. Hierbei wurde auf die erste Version des CLANs zurückgegriffen. Es wurde untersucht, welche Parallelisierungsstrategie sich am besten eignet. Ein großes Problem bei der Parallelisierung von neuronalen Netzen besteht in der komplexen Verbindungstruktur. Selbst wenn für jedes einzelne Neuron ein eigener Prozessor vorhanden wäre, könnte zwar die maximal mögliche Parallelität aus dem neuronalen Netz gewonnen werden, doch aufgrund der komplexen Verbindungstruktur ist der Aufbau eines solchen Systems mit der heutigen Technik nicht denkbar.

Für das CLAN gilt, daß alle Schichten mit der assoziativen Schicht A vollständig vernetzt sind. Innerhalb der einzelnen Schichten des CLANs gibt es keine Verbindungen. Daher ist eine Verteilung eines kompletten Layers auf je einen einzelnen Transputer bzw. ein Transputercluster nicht sinnvoll, denn die gesamte Information, die zwischen den Neuronen ausgetauscht wird, müßte über die Links gesendet werden, und innerhalb eines Transputers fände keinerlei Datenaustausch zwischen den einzelnen Elementen statt. Außerdem wäre es sehr schwierig, die Transputer gleichmäßig auszulasten, da die einzelnen Layers sehr unterschiedliche Rechenleistung benötigen und diese außerdem noch vom betrachteten Zeitpunkt abhängig ist.

Eine alternative Möglichkeit der Parallelisierung besteht darin, die Schichten gleichmäßig auf die einzelnen Transputer zu verteilen. Es wurde untersucht, inwieweit eine Beschleunigung zu erreichen ist. Der große Nachteil ist hierbei, daß nach jedem Simulationsschritt alle Gewichtsfaktoren der aktiven Neuronen gesammelt und an die A-Neuronen gesendet werden müssen. Fig. 5.8 zeigt, daß nur bis zu einer kleinen Transputerzahl eine Beschleunigung zu erreicht wurde. Die Werte gelten für ein Transputernetz, in dem die Transputer als Ring verbunden sind.

Bei der Wahl von anderen Topologien ergibt sich keine wesentliche Verbesserung der Simulationszeit. Werden die Transputer in Würfelform verbunden, dann läßt sich nochmals eine Reduzierung der Rechenzeit um 20-30% erzielen. Die Simulationszeit konnte für einen Lernvorgang von 5 Minuten auf 1,5 Minuten reduziert werden. Insgesamt kann aber gesagt werden, daß sich das CLAN für die beschriebene Art der Parallelisierung nur in geringem Maße eignet.

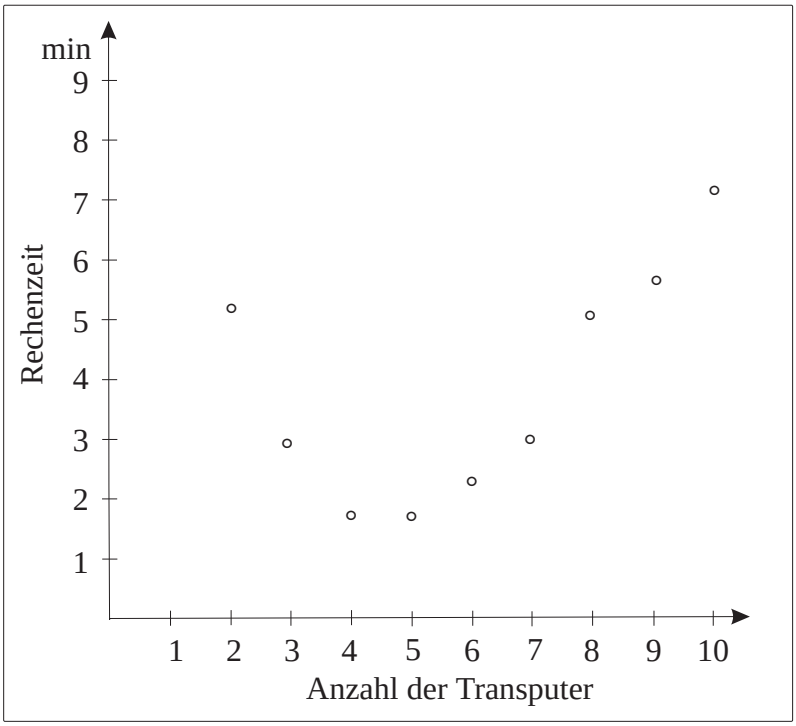


Fig. 5.8: Rechenzeit bei einer gleichmäßigen Verteilung auf die Transputer.

5.5.1 Aufteilung auf mehrere CLANs

Dies führt zu der Überlegung, auf jedem Transputer ein komplettes CLAN zu implementieren [Busemann 1991]. Hierzu wird der Merkmalsvektor des Eingabemusters in einzelne Teile zerlegt, und jeder Transputer erhält als Eingabe einen Teil des Bildes, den er zu bearbeiten hat. Diese Vorgehensweise erscheint auch biologisch sinnvoll, denn der gesamte Eingangsvektor des CLANs im SENROB Projekt hat mehrere zehntausend Einträge. Dies hieße, daß z.B. die F-Schicht ebenso viele Neuronen benötigte und ein Neuron somit über hundertausend Synapsen hätte, während sein biologisches Vorbild nur über einige Tausend Synapsen verfügt. Durch die Benutzung mehrerer CLANs verringert sich die Zahl der Synapsen auf eine in der Biologisch vorkommende Anzahl.

Um ein CLAN auf einem T800 Transputer zu implementieren, war es notwendig, die Größe des CLANs anzupassen. Zum einen habt ein T800-Transputer nur einen begrenzten Speicherplatz,

andererseits sollte die Zahl der Transputer noch einigermaßen überschaubar sein. Deshalb wurde die assoziative Schicht eines CLANs auf 100 Neuronen verkleinert, während die Anzahl der F- und F'-Neuronen pro CLAN auf 1536 erhöht wurde. Hierdurch konnte das gesamte CLAN auf 32 Transputern simuliert werden. Die I-Schicht erhielt 250 Neuronen.

Bei der Simulation werden auf allen Transputern die gleichen Prozesse gestartet (Fig. 5.9). Neben dem Prozeß, der jeweils ein CLAN simuliert, sind noch weitere Prozesse für die Ein- und Ausgabe zuständig. Durch eine Grafikoberfläche, die vergleichbar mit der Grafikoberfläche in der sequentiellen Version ist, werden die einzelnen CLANs angesteuert. Zusätzlich visualisiert sie die Muster und Ergebnisse.

Die einzelnen CLAN-Prozesse auf den verschiedenen Transputern tauschen keine Informationen aus. Es werden keinerlei Daten untereinander ausgetauscht. Auch die Ein- und Ausgabeprozesse auf den verschiedenen Transputern kommunizieren nicht untereinander. Es werden nur Daten durchgereicht, für den internen CLAN-Prozeß entgegengenommen oder von diesem weitergeleitet. Dies ist eine ideale Voraussetzung für die Parallelisierung auf einem Transputernetz.

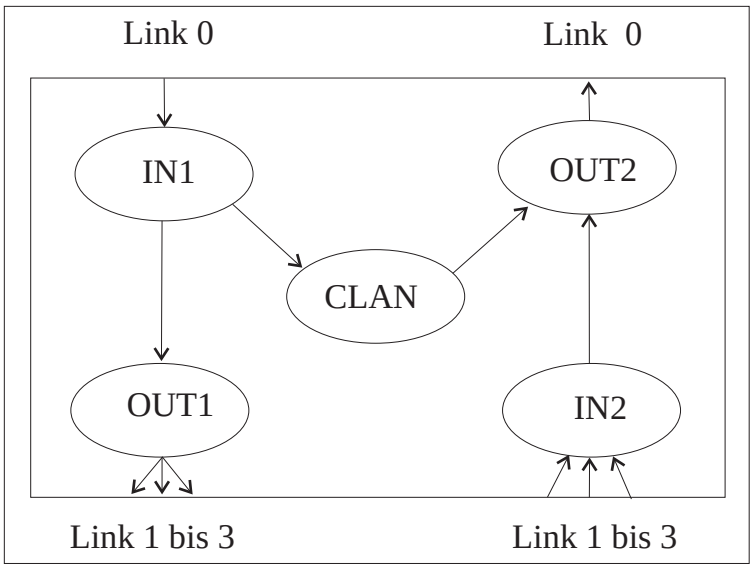


Fig. 5.9: Auf jedem Transputer laufen identische Prozesse ab.

Als Topologie für das Transputernetz wurde ein Baum gewählt, da so die Information auf kürzestem Wege versendet werden kann. Dadurch ist gewährleistet, daß eine Nachricht höchstens durch log(n) Transputer geleitet werden muß, wobei n die Anzahl der einzelnen CLANs ist.

Nach dem Laden des Codes auf die Transputer werden alle Variablen auf den Transputern initialisiert. Zusätzlich muß jeder Transputer bestimmen, über welche Links weitere CLANs erreichbar sind. Grundsätzlich gilt, daß über Link 0 ein Transputer die Eingabe- und Steuerwerte empfängt und die Ausgaben versendet. Diese Verbindung ist bei jedem Transputer geschaltet. Somit muß nur kontrolliert werden, ob über die Links 1 bis 3 eine Verbindung zu einem anderen im Baum tiefer liegenden CLAN besteht. Dies wird dadurch erreicht, daß jeder Transputer zu Beginn über den Link 0 ein *Aktivsignal* an seinen Vorgänger sendet. Dieser überwacht die Links 1 bis 3 am Anfang und merkt sich, über welche Links ein Aktivsignal ankommt. Nur diese Verbindungen sind aktiv und erhalten während des Programmablaufs die ankommenden Steuersignale.

Der CLAN-Prozeß entspricht zum größten Teil dem Algorithmus, wie er schon in der sequentiellen Version angewendet wurde (vgl. Kapitel 5.1). Er mußte lediglich nach OCCAM portiert werden. Der Eingabeprozeß IN1 wartet darauf, daß am Link 0 eine Information eintrifft. Diese empfängt er und verarbeitet sie. Die Eingaben, die für das interne CLAN bestimmt sind, werden an dieses weitergegeben. Alle anderen Eingaben werden parallel über die aktiven Ausgabelinks an die anderen Transputer weitergesendet.

Die Ergebnisse der einzelnen Transputer durchlaufen den umgekehrten Weg. Über die Links 1 bis 3 empfangen die IN2-Prozesse die entsprechenden Daten und geben sie an den OUT2-Prozeß weiter. Außerdem übergibt der CLAN-Prozeß seine Ergebnisse an den OUT2-Prozeß, welcher diese dann über den Link 0 versendet.

Von der Steuereinheit muß immer der gleiche Teilbereich eines Musters an den gleichen Transputer gesendet werden. Um eine eindeutige Zuordnung zu erhalten, müssen die Teilmuster und die Transputer gekennzeichnet werden. Die einfachste Möglichkeit bestünde darin, die Transputer durchzunumerieren. Doch dann müßte ein Transputer, der das Teilmuster nur weiterreichen soll, genau wissen, welche Transputer im Baum unter ihm über welchen Link zu erreichen sind. Dies hieße aber, daß auf jedem Transputer eine spezielle auf ihn zugeschnittene Tabelle vorhanden sein müßte und eine Änderung in der Größe des Netzwerks womöglich eine Änderung in allen Tabellen nach sich zöge.

Deshalb wurde eine Kennzeichnung gewählt, die es erlaubt, auf allen Transputern identische Programme ablaufen zu lassen. In der Kennzeichnung wird der zu benutzende Weg kodiert. Dies geschieht folgendermaßen: Der gesamte Weg wird durch eine Integerzahl kodiert. Das letzte Byte gibt immer an, über welchen Link die Nachricht als nächstes weiterzusenden ist. Nachdem dieses Byte gelesen wurde, wird die gesamte Zahl um ein Byte nach rechts verschoben, so daß nun das vorletzte Byte an die letzte Position rückt. Dieses Byte gibt dann an,

über welchen Link der nächste Transputer die Nachricht weitersenden muß. Ist das Byte gleich Null, sind die Daten für den internen CLAN-Prozeß bestimmt. Dies ist zulässig, da über Link 0 selbst nur Daten empfangen, aber nicht weiterverschickt werden.

Für das Zurücksenden von Nachrichten gilt die umgekehrte Vorgehensweise. Jeder Transputer verschiebt das letzte Byte in die vorletzte Position und schreibt in das letzte Byte, über welchen Link die Nachricht kommt. So wird für jede Nachricht die gleiche eindeutige Kennung aufgebaut, die in einer Tabelle in der Grafikoberfläche festgehalten ist.

Diese Vorgehensweise erlaubt es auch, die Verteilung der Transputer sehr flexibel zu halten. Bei einer Änderung muß nur in der Tabelle der Grafikoberfläche die Änderung vollzogen werden. Am Code auf den einzelnen Transputern ändert sich nichts.

Beim Auftreten von sehr vielen Ausgaben kann es geschehen, daß Nachrichten von tiefer liegenden Transputern nicht von den anderen Transputern abgenommen werden, da diese zu sehr mit ihrer eigenen Ausgabe beschäftigt sind. Um dies zu verhindern, werden die Transputer bei jeder Ausgabe synchronisiert.

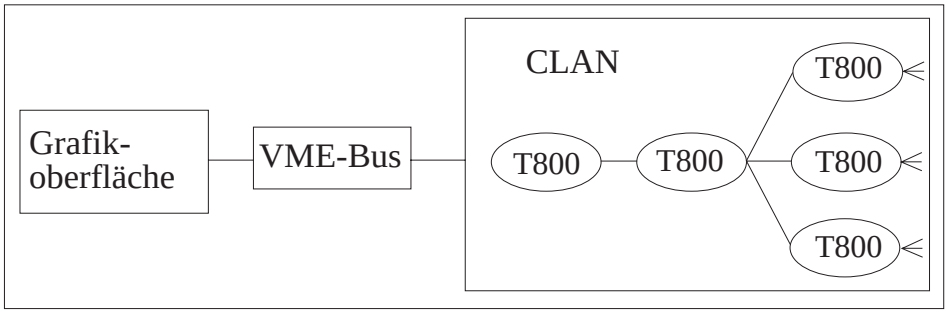


Fig. 5.10: Mit Hilfe einer Grafikoberfläche, die direkt über dem VME-Bus arbeitet, wird auf die Transputer zugegriffen.

Die Transputer verfügen über keine grafische Ausgabe. Deshalb wurde eine Grafikoberfläche erstellt, die unabhängig von den Transputern auf einem Sun/Sparc-Rechner läuft. Sie kommuniziert direkt über den VME-Bus mit den Transputern (Fig. 5.10). Da alle Informationen über diese Verbindung gesendet werden müssen, ist dies ein großer Engpaß im System. Deshalb sollten so wenig Daten wie möglich zwischen der Grafikoberfläche und den Transputern ausgetauscht werden. Idealerweise wird nur das Muster am Anfang verteilt und dann das Endergebnis nach Abschluß eines Lernvorganges bzw. einer Wiedererkennung eingesammelt. Es ist natürlich auch möglich, während des Programmablaufs den aktuellen Zustand anzuzei-

gen, doch dies vergrößert die Laufzeit erheblich, da sehr viele Daten über die einzige Verbindung gesendet werden müssen. Nachstehend soll die Grafikoberfläche vorgestellt werden.

5.5.2 Hauptfenster

Die Grafikoberfläche vermittelt sowohl eine grobe Übersicht als auch detaillierte Information über die präsentierten Muster und Ergebnisse. Mit ihr können Muster betrachtet und manipuliert und Ergebnisse angezeigt werden. Außerdem dient sie zur Ansteuerung des Programmes.

Im Hauptsteuerfenster (Fig. 5.11) wird der Link zu den Transputern ausgewählt ("LINK") und eine Verbindung zu den Transputern hergestellt ("OPEN"). Hierbei werden die Transputer zwar schon initialisiert, dennoch kann das System jederzeit in seinen Anfangszustand gesetzt werden ("INIT"). Um einen Lern- bzw. Erkennungsvorgang zu starten, dient der Button "RUN", während mit "PATTERN" die präsentierten Muster und mit "RESULT" die Ergebnisse betrachtet werden können.

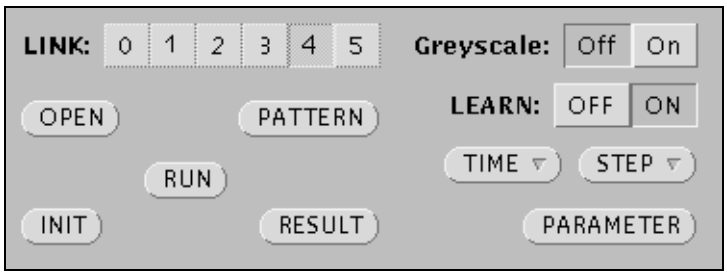


Fig. 5.11: Hauptsteuerfenster der Grafikoberfläche.

Die einzelnen Buttons dienen zur Manipulation der einzelnen CLANs. Das Lernen kann an- und ausgeschaltet werden. Außerdem kann mit der "TIME"-Auswahl die Präsentationszeit in Rechenschritten angegeben werden, die standardmäßig auf 1000 Rechenschritten liegt. Dies reicht aus, um ein Muster zu lernen. Soll nur ein Muster wiedererkannt werden, genügt eine kürzere Simulationszeit.

Die Zeit, die unter "STEP" einstellbar ist, beinhaltet den Zeitraum, nach welchem Ergebnisse von den einzelnen CLANs erwünscht werden. Dieser Wert liegt standardmäßig bei 100 Zeitschritten. Um eine optimale Laufzeit zu erhalten, muß dieser Wert mit dem unter "TIME" eingestellten Wert übereinstimmen. Es werden dann keine Zwischenergebnisse, sondern nur das Endergebnis angezeigt. Über den Button "Parameter" wird ein weiteres Fenster geöffnet, in dem die Parameter der einzelnen Neuronen manipuliert werden können (Fig. 5.12). Es handelt sich im einzelnen um die Größe und Streuung der Anfangsgewichte und die Verringe-

rung der dynamischen Schwellen und des Membranpotentials in jedem Zeitschritt. Die Änderungen werden an allen CLANs auf allen Transputern vorgenommen.

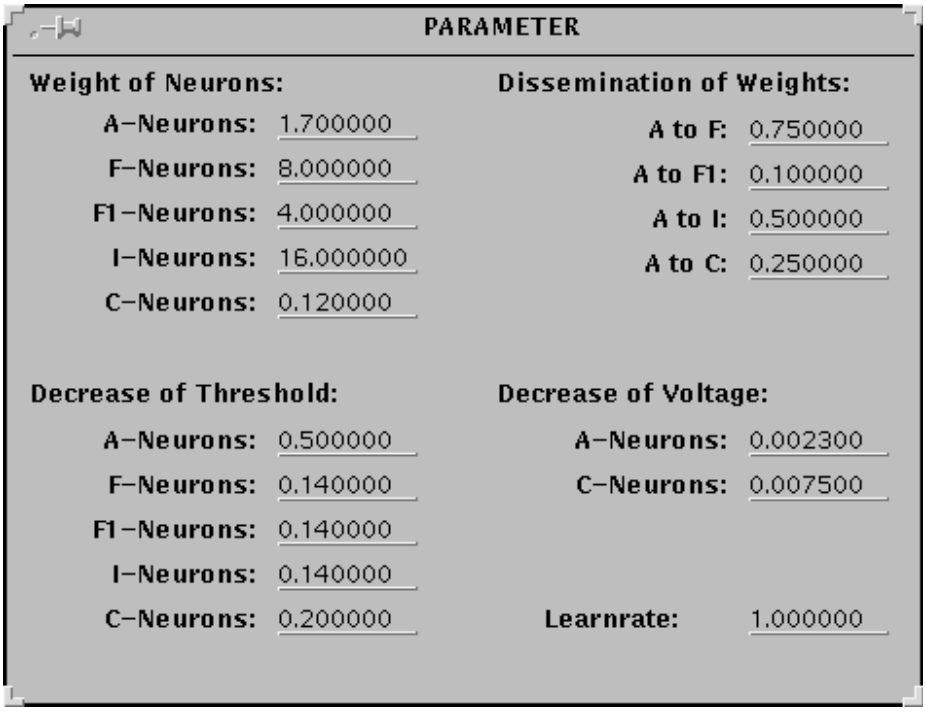


Fig. 5.12: Fenster zum Einstellen der einzelnen Parameter.

5.5.3 Musterfenster

Das Musterfenster "PATTERN" (Fig. 5.13) zeigt die Teilmuster für die einzelnen CLANs an. Unter jedem Teilmuster ist die Nummer des Transputers angegeben, auf dem das Muster bearbeitet wird. Es kann mit künstlich erzeugten Mustern sowie mit Merkmalsvektoren, die von dem vorgeschalteten Erkennungssystem erzeugt wurden, gearbeitet werden.

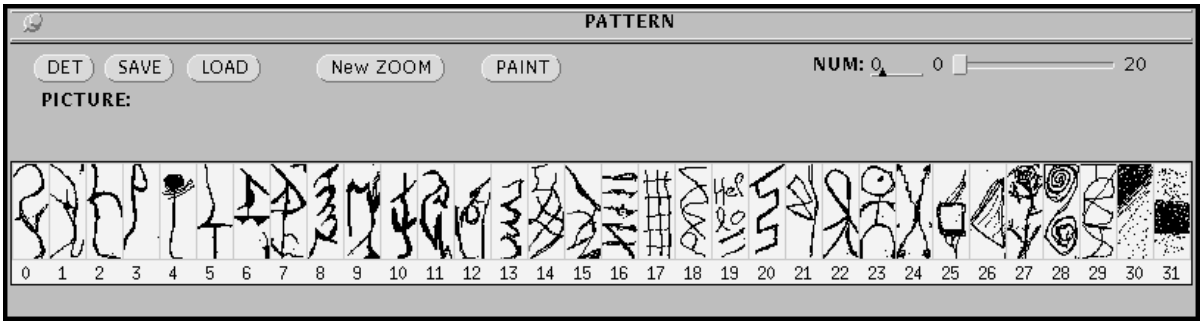


Fig. 5.13: Das Musterfenster stellt die Teilmuster für die einzelnen CLANs 0 bis 31 auf jedem Transputer dar.

Da die dargestellten Teilmuster sehr klein sind, kann für jedes Teilmuster durch Doppelklick auf dem Muster ein weiteres Fenster geöffnet werden, in dem das Teilmuster detailliert gezeigt wird (Fig. 5.14). Jeder Eintrag des Mustervektors wird in einem Quadrat dargestellt, wobei eine aktive Komponente farbig hervorgehoben wird. Mit der Maus kann nun der Merkmalsvektor verändert werden. Es können Komponenten aktiviert bzw. deaktiviert werden.

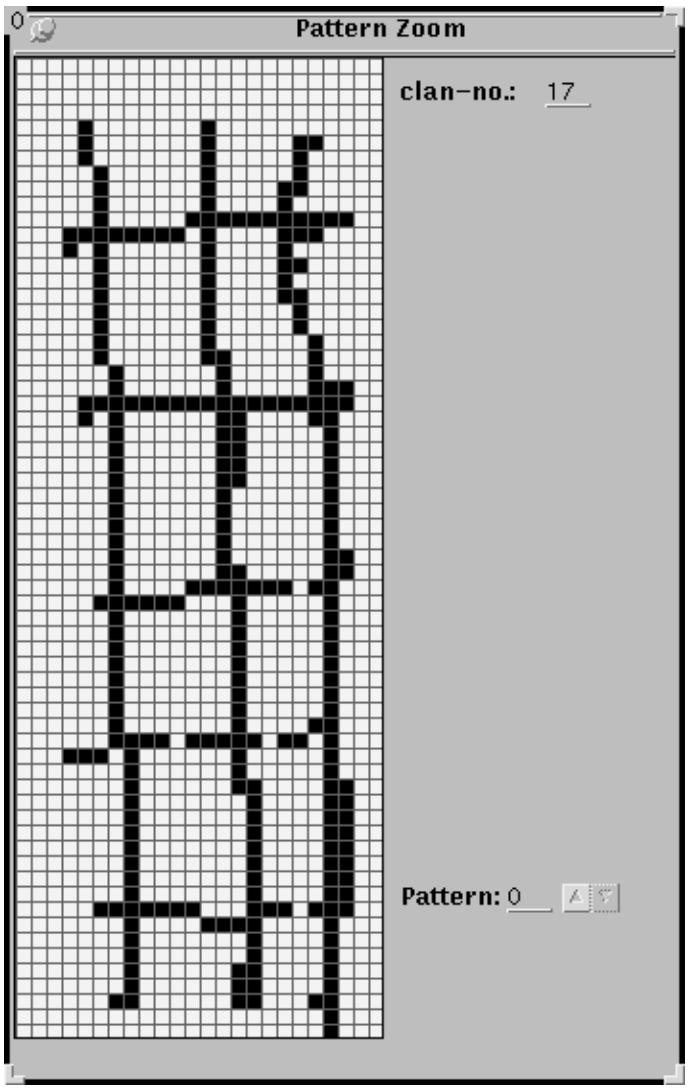


Fig. 5.14: Detaillierte Darstellung eines Teilmusters.

In diesem Fenster kann man außerdem das Teilmuster mit "clan-no" auswählen, um ein Muster zu betrachten, das auf einem anderen CLAN bearbeitet wird. Desweiteren kann das Gesamtmuster gewechselt und so verschiedene Teilmuster angeschaut werden, die auf dem gleichen Transputer und somit vom gleichem CLAN bearbeitet werden. Um einen direkten Vergleich durch führen zu können, kann mit "NEW ZOOM" in Fig. 5.13 ein weiteres Vergrößerungsfenster geöffnet werden; es können dann zwei Teilmuster nebeneinander betrachtet werden.

Die im Mustereditor vorgenommenen Änderungen gelten nur für den aktuellen Programmaufruf. Erst wenn die Daten durch "SAVE" auf die Festplatte geschrieben werden, sind sie für einen weiteren Programmaufruf greifbar. Durch den Button "LOAD" werden die abgespeicherten Mustersätze geladen. Um ein Muster eines Merkmalsvektors aus dem vorgeschalteten Erkennungssystem zu laden, dient der Button "DET". Damit nicht jedesmal ein Muster geladen werden muß, werden immer mehrere Muster gleichzeitig im Speicher gehalten. Mit der Auswahl NUM kann das gewünschte Muster eingestellt werden.

5.5.4 Ergebnisfenster

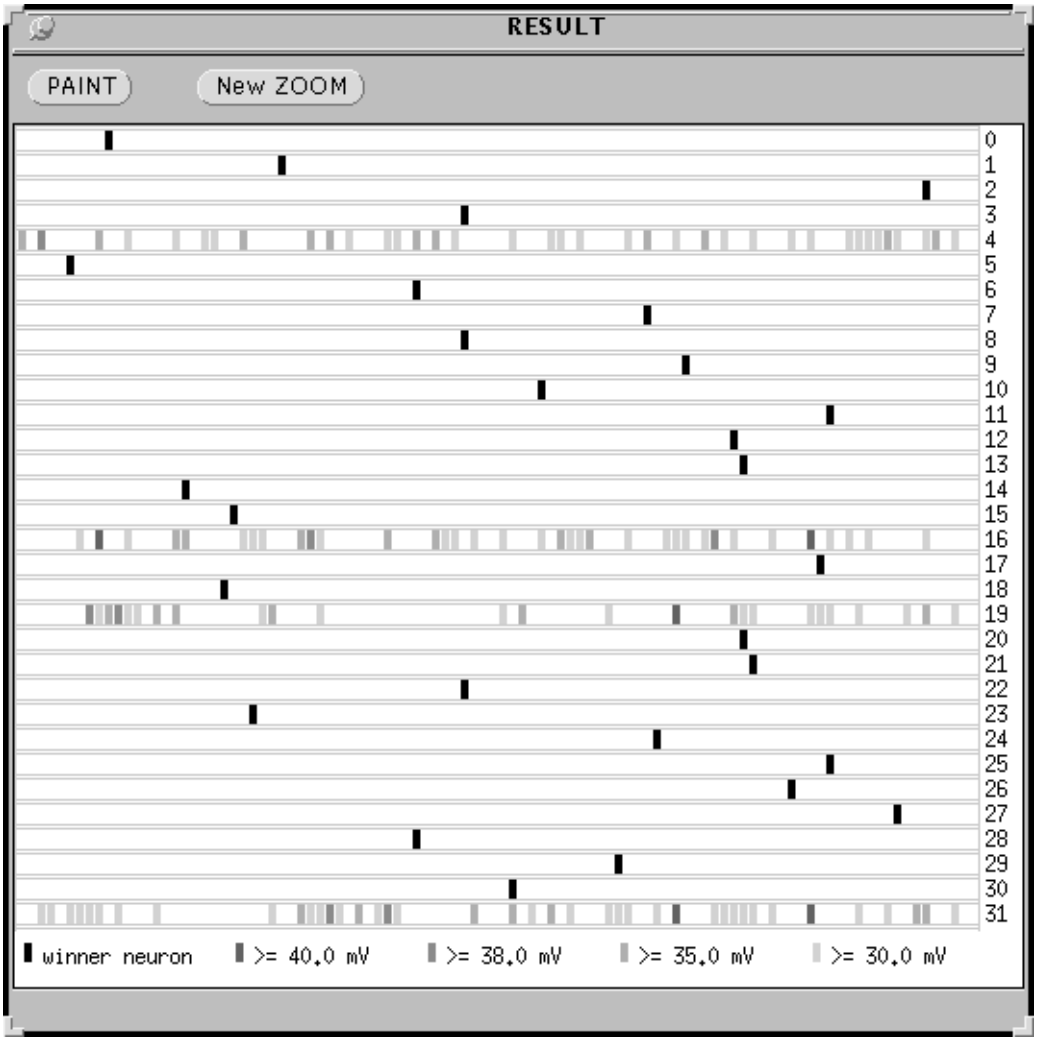


Fig. 5.15: Im Ergebnisfenster werden die Membranpotentiale aller A-Neuronen durch Farben gekennzeichnet dargestellt.

Das Ergebnis wird in zwei verschiedenen Fenstern angezeigt. In einem Übersichtsfenster (Fig. 5.15) wird für alle CLANs der Potentialzustand der A-Neuronen angezeigt. Für jedes CLAN wird eine Zeile benutzt, in der farbig die Größe des Membranpotentials der einzelnen A-Neuronen von rechts nach links durch ein Rechteck aufgetragen wird. Hierbei wird unterschieden zwischen dem Neuron mit dem höchsten Potential, dem Gewinnerneuron, und der Neuronen, deren Membranpotential größer als 38 mV, 35 mV bzw. 30 mV ist. Alle anderen werden nicht mehr dargestellt.

So werden auf einen Blick die Zustände in den einzelnen CLANs sichbar. Durch Anklicken einer Reihe wird für das entsprechende CLAN ein separates Fenster (Fig. 5.16) geöffnet, in dem das Membranpotential in mV für alle A-Neuronen aufgetragen ist. Hierbei wird das Ruhepotential von -80mV mit 0mV bezeichnet. Das Neuron mit dem höchsten Membranpotential wird als "winner neuron" angegeben. Mit "clan-no" kann zwischen den einzelnen CLANs und mit "pattern number" zwischen den einzelnen Mustern für das aktuelle CLAN gewechselt werden. Alle Ergebnisse sind in einem Array gespeichert, so daß sie später miteinander verglichen werden können. Um die Ergebnisse miteinander zu vergleichen, können mit "New ZOOM" zusätzliche Ergebnisfenster mit Detailinformationen geöffnet werden.

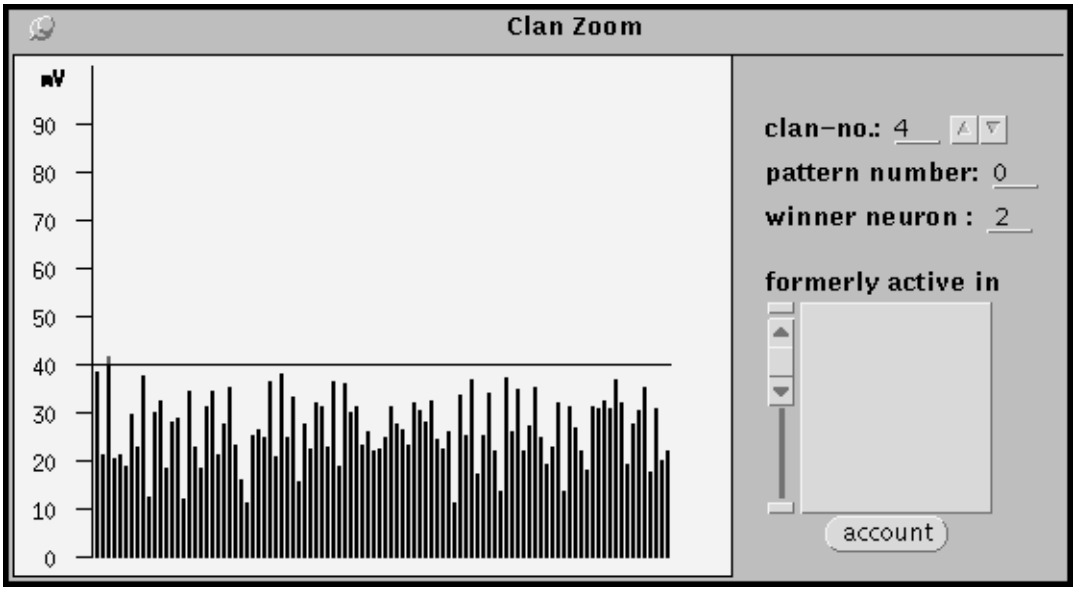


Fig. 5.16: Das Ergebnis wird detailliert dargestellt.

5.6 Bewertung der Ergebnisse

Im Unterschied zur sequentiellen Form des CLANs liefert die parallele Version nicht nur *ein* Neuron als Ergebnis, sondern jedes CLAN verfügt nach einem Lern- bzw. Erkennungsvorgang über ein Gewinnerneuron. Ein Muster wird somit durch eine Anzahl von Neuronen gelernt. Für die Versuche, die mit den parallelen CLANs durchgeführt wurden, reicht eine grobe Abschätzung der Ergebnisse aus. Beim Wiedererkennen ergeben sich folgende Möglichkeiten: Wird ein Muster in all seinen Teilkomponenten erkannt, d. h. alle Gewinnerneuronen in den Teilmustern sind gleich, ist das Muster bekannt. Stimmt ein Großteil der Gewinnerneuronen überein, gilt das Muster weiterhin als erkannt. Ist hingegen der größte Teil aller Gewinnerneuronen unterschiedlich oder gibt es sogar keine Übereinstimmung, ist das Muster unbekannt.

In der vorgestellten Form mit den T800 braucht ein CLAN zum Erlernen eines Musters eine Rechenzeit von ca. 90 s. Um ein Muster wiederzuerkennen, sind ca. 30 s notwendig. Für einen realistischen Einsatz im gesamten Erkennungssystem sind diese Zeiten nicht annehmbar, da eine Erkennungszeit für das ganze Erkennungsystem angestrebt wurde, die im Bereich von einer Sekunde liegt. Außerdem ließ die nächste Transputergeneration, der T9000, noch auf sich warten, so daß die Arbeiten am parallelen CLAN nicht weiter fortgeführt wurden. Mit den heute zur Verfügung stehenden schnelleren Rechnern könnten geringere Simulationszeiten erreicht werden.

5.7 Einsatz des CLANs im PDIS

Wegen der langen Bearbeitungszeit des neuronalen CLANs wird im PDIS eine modifizierte Version eingesetzt, die von der neuronalen Verarbeitung abgeht. Das Entscheidungskriterium des CLANs ist das normierte Qualitätsmaß Q = (3m - l - p) / p. In dieser Version wird für jedes Merkmal eines Musters das Q-Maß zu allen bisher gelernten Mustern berechnet und das Muster anhand der Qualitätsmaße bewertet.

Das PDIS extrahiert, wie im Kapitel 2.1 dargestellt, aus dem aufgenommenen Bild die drei Merkmale

- Orientierung (Kantenverlauf),
- Flächen und
- Ecken.

An das CLAN werden in der aktuellen Version die Merkmale von 8160 Koordinaten übergeben. An jeder Koordinate sind für jedes Merkmal maximal 24 Einträge vorgesehen. Hieraus folgt, daß der binäre Eingabevektor für ein Merkmal mit 195840 Komponenten besetzt ist, wobei die aktiven Komponenten mit 1 codiert werden.

5.7.1 Ermittlung der Q-Maße

Alle drei Eingangsvektoren werden durch die algorithmische Lösung des CLANs unabhängig voneinander klassifiziert. Für jeden Vektor wird das beste Q-Maß zu allen Vektoren der gleichen Merkmalsklasse berechnet. Hierzu werden für das aktuelle Muster zuerst die gesetzten Bits gezählt. Dies ergibt den Wert p. Anschließend wird der aktuelle Vektor nacheinander mit allen vorher gespeicherten Vektoren des gleichen Merkmals verglichen. Die beiden Vektoren werden jeweils durch ein logisches UND verknüpft und die aktiven Bits im resultierenden Vektor gezählt. Als Ergebnis erhält man den Match m. Die Größe des betrachteten gespeicherten Musters l ergibt sich aus der Anzahl der aktiven Bits in dem entsprechenden Vektor. So kann nun das Qualitätsmaß berechnet und abschließend das sein Maximum ermittelt werden.

Aus den drei Qualitätsmaßen muß wieder ein Gesamtergebnis errechnet werden. Hierzu wird für jedes Merkmal zuerst eine Schwelle eingeführt, über der das Qualitätsmaß liegen muß, damit das Muster als wiedererkannt gilt. Die Schwelle wurde so gelegt, daß gleiche Objekte, die aber z.B. in unterschiedlicher Lage aufgenommen wurden, wiedererkannt werden, während ähnliche Objekte nicht erkannt werden. Die Untersuchungen haben gezeigt, daß für jedes Merkmal eine eigene Schwelle notwendig ist (Tabelle 5.1).

Merkmal	Schwelle
Fläche	0,40
Kanten	-0,40
Ecken	-1,00

Tab. 5.1: Die einzelnen Schwellen für die unterschiedlichen Merkmale.

Die negativen Schwellen ergeben sich aus folgender Betrachtung. Es sei Q_{max} der maximale und Q_{min} der minimale Q-Wert. Unter der Bedingung, daß die beiden präsentierten Muster die gleichen aktiven Komponenten (p = l = m) haben, ergibt sich:

$$Q_{max} = (3m - p - l) / p$$

$$= (3p - p - p) / p$$
$$= p / p$$
$$= 1 \qquad (5.1)$$

Falls keine Übereinstimmung vorliegt (m = 0), gilt für p = l

$$Q_{min} = (3m - p - l) / p$$
$$= (0 - p - l) / p$$
$$= -2p / p$$
$$= -2 \qquad (5.2)$$

Somit liegt der Q-Wert im allgemeinen im Intervall [-2; 1]. Ist das gelernte Muster sehr viel größer als das präsentierte Muster ($p \neq l$), kann das Q-Maß kleiner als -2 werden, doch ist dieser Fall aufgrund der unterschiedlichen Mustergrößen eher unbedeutend.

Neben den drei Merkmalsvektoren liefert das PDIS noch die Entfernung d zwischen Kamera und Objekt, in der das Objekt retinafüllend aufgenommen wurde. Hieraus wird für jedes Objekt das *Entfernungsintervall*

$$[d + \sqrt[48]{2}; d - \sqrt[48]{2}]$$

gebildet [Kräuter 1995]. Vor der Berechnung der Qualitätsmaße wird bei der Klassifizierung die Entfernung ausgewertet. Alle bereits gelernten Objekte, deren Aufnahmeentfernung nicht im Entfernungsintervall des präsentierten Musters liegt, können nicht mit diesem übereinstimmen. Sie werden deshalb aussortiert. Von nun an werden nur noch die gelernten Objekte betrachtet, deren Aufnahmeentfernung im Entfernungsintervall liegt.

Für alle dann noch zu betrachtenden Muster wird das Q-Maß der Fläche berechnet. Hierbei werden alle Muster aussortiert, deren Q-Wert nicht über der Schwelle der Fläche liegt. Anschließend wird der Vorgang für die Kanten und Ecken wiederholt, und so werden noch weitere Muster aussortiert, deren Q-Wert nicht über der Schwelle für Kanten bzw. Ecken liegt.

Am Schluß bleiben nur die gelernten Muster übrig, deren drei Q-Maße über den Schwellen liegen. Aus diesen muß das am besten übereinstimmende Q-Maß gefunden werden. Es hat sich gezeigt, daß nicht alle drei Merkmale mit der gleichen Gewichtung zu berücksichtigen sind. Die Fläche eines Objekts ist das robusteste und häufigste Merkmal. Deshalb geht sie mit dem größten Anteil in das Endergebnis ein. Auch der Kantenverlauf enthält markante Informationen. Sie erhalten die nächst niedrigere Gewichtung. Die Ecken werden benötigt, um kreisförmige Ob-

jekte von eckigen zu unterscheiden. Sie werden mit dem niedrigsten Gewichtsfaktor in der Bewertung berücksichtigt (Tabelle 5.2).

Merkmal	Gewichtsfaktor
Fläche	0,6
Kanten	0,3
Ecken	0,1

Tab. 5.2: Die Gewichtsfaktoren für die einzelnen Merkmale.

Die einzelnen Q-Maße werden so normiert, daß der berechnete Faktor zwischen 1 (entspricht: 100% Übereinstimmung) und 0 (keine Übereinstimmung) liegt. Das aktuell präsentierte Muster wird der Musterklasse mit dem höchsten Q-Maß zugeordnet.

Bleibt keines der gelernten Muster übrig, dessen Qualitätsmaße über der Schwelle liegen, gilt das Muster als nicht erkannt und wird gelernt. Hierzu wird es gespeichert und von nun an mit dem jeweiligen neuen Muster verglichen. Im Fall des Nichterkennens wird als Ergebnis der Name des Musters mit dem maximalen Qualitätsmaß der Fläche vorgeschlagen. Das Maß der Fläche wird benutzt, da es bereits vorliegt und somit nicht neu berechnet werden muß. Um den übrigen Grad der Übereinstimmung zu bestimmen, werden die anderen beiden Qualitätsmaße und die Bewertung der drei Q-Maße berechnet. Diese Vorgehensweise ist ausreichend für die Angabe, welches Muster dem aktuellen am ähnlichsten ist.

5.7.2 Implementierung

Das CLAN wird in drei verschiedenen Versionen im Erkennungssystem genutzt. Zum einen gibt es das unabhängige Programm "ccmauto", das in der Kommandozeile aufgerufen wird. Mit diesem Modul können Merkmalsdaten untersucht werden, ohne das vorgeschaltete Erkennungssystem zu benutzen. Es kann immer nur das Qualitätsmaß für einen Merkmalstyp bestimmt werden.

Über Parameter werden die Dateinamen angegeben, in denen die Daten stehen. Außerdem muß das jeweilige Merkmal ausgewählt werden, welches in den Dateien gespeichert ist. Es besteht die Möglichkeit, das erste Muster mit allen anderen Mustern oder alle Muster untereinander zu vergleichen. Als Ausgabe liefert das Programm nicht das maximale Qualitätsmaß, sondern es werden alle berechneten Q-Werte in einer Tabelle aufgelistet. Die endgültige Bewertung ist vom Benutzer selbst durchzuführen.

Die beiden anderen Versionen sind in das vorgeschaltete Erkennungssystem integriert. Es handelt sich um eine Implementation auf Transputerbasis und eine sequentielle Version auf einer Sparc10. Gemeinsam für beide Versionen gibt es eine Grafikoberfläche, in der die Ergebnisse angezeigt werden und der Name und der Winkel eines unbekannten Objekts eingelesen werden können.

5.7.3 Implementierung auf einem Transputernetz

Da das CLAN mit allen anderen Erkennungsmodulen gekoppelt wurde, wird kurz auf die parallele Implementierung des Gesamtsystems eingegangen (Fig. 5.17). Das Objekt wird mit der Kamera und einem Color Frame Grabber (CFG) aufgenommen und über den TIP-Bus auf vier Transputer (Transf) verteilt. Diese führen die Transformation ins logarithmisch-polare Koordinatensystem durch und senden diese Daten dann weiter an die nächsten vier Transputer (E&I), welche die Merkmale extrahieren. Außerdem wird das Objekt in seiner Repräsentation in die Vorzugsorientierung gedreht. Danach werden die Merkmale an das CLAN gesendet. Dort werden in der beschriebenen Weise die Q-Werte berechnet und die Bewertung durchgeführt. Die einzelnen Module arbeiten als Pipeline. Um das erste Objekt zu erkennen, werden einschließlich der Roboterbewegungen 6 bis 7 s benötigt. Ist die Pipeline gefüllt, wird eine Erkennungszeit von 1 bis 2 s erreicht.

Damit das CLAN die Ergebnisse zur unabhängigen Grafikoberfläche senden kann, existiert auf dem Root-Transputer ein Prozeß, der die Ergebnisse vom CLAN-Transputer entgegennimmt und aus dem Transputersystem in eine Datei schreibt. Dieser Umweg ist notwendig, da nur der Root-Transputer mit Modulen außerhalb des Transputersystems kommunizieren kann. Außerdem muß er, wenn das Muster nicht erkannt werden konnte und somit der Name für das Muster angegeben werden muß, den Namen und den Winkel einlesen. Die Kommunikation zwischen den beiden Transputern in beide Richtungen erfolgt über ein Protokoll, das folgende Komponenten enthält:

- Ein Signal, ob das Muster erkannt wurde oder nicht,
- die drei Q-Maße,
- die Bewertung der Übereinstimmung,
- der Name des erkannten bzw. des bestpassenden Musters,
- die x-, y- und z-Koordinaten und
- die Orientierung.

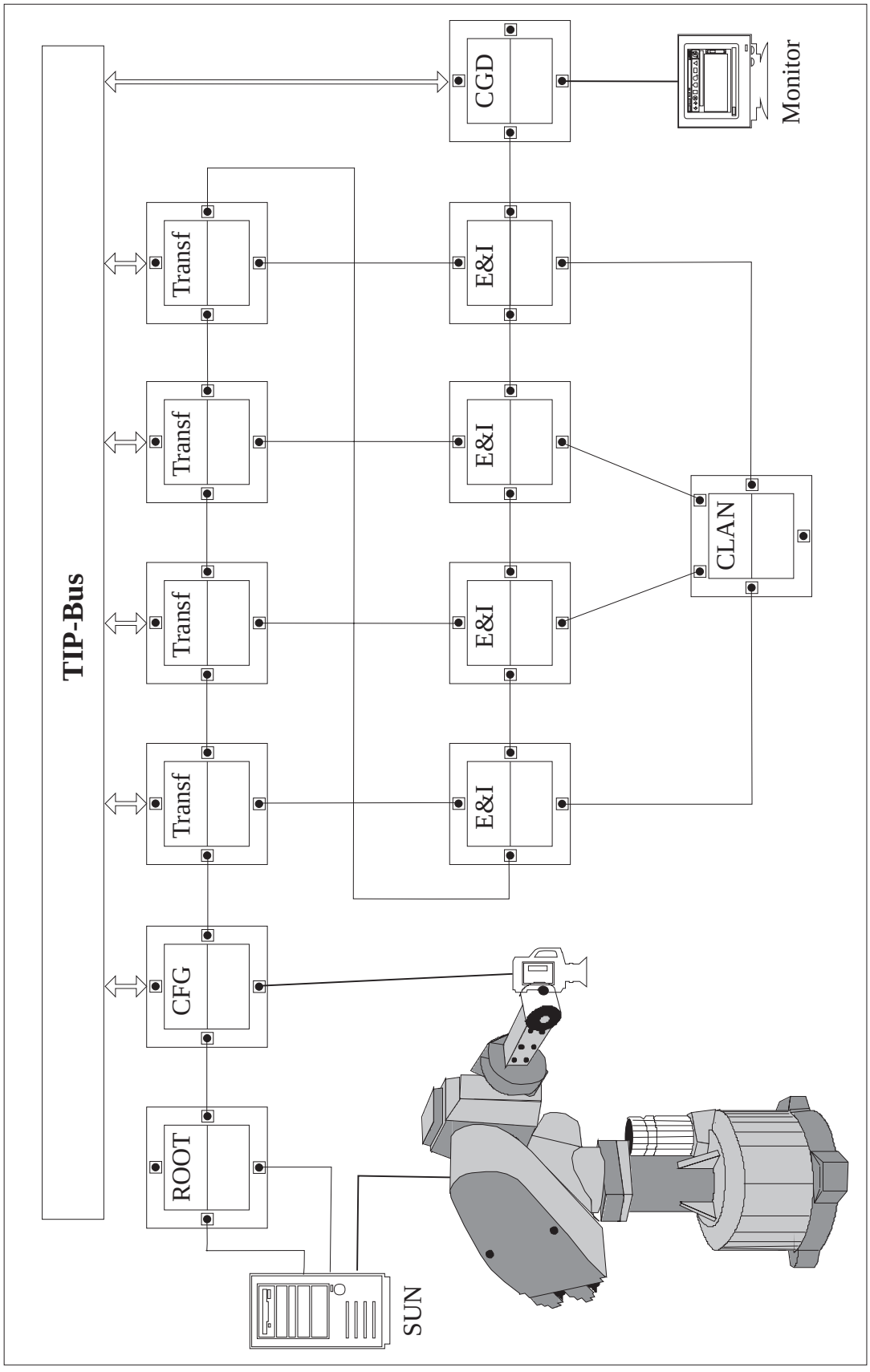


Fig. 5.17: Das gesamte Erkennungssystem auf Transputerbasis.

Wurde das Objekt erkannt, kann der CLAN-Transputer, nachdem er das Ergebnis an den Root-Transputer gesendet hat, sofort das nächste Muster bearbeiten. Konnte das Muster nicht erkannt werden, wartet er so lange, bis er den Namen und den Winkel des Objekts erhalten hat. Das CLAN ist so lange blockiert, damit eine eindeutige Zuordnung zwischen der Benutzereingabe und dem Muster möglich ist.

Die Koordinaten werden von den vorgeschalteten Modulen durchgereicht und ohne weitere Bearbeitung an die Ausgabeeinheit weitergegeben. Anders sieht dies bei der Orientierung des Objekts aus. Das Modul zur Ermittlung der Vorzugsorientierung gibt an, um wieviel Grad das Objekt zurückgedreht wurde, damit es in seiner Vorzugsorientierung liegt. Die gleiche Angabe besteht ebenfalls für das bereits gelernte Muster. Aus der Differenz der beiden Werte ergibt sich die Orientierung des präsentierten Objekts. Kann das präsentierte Muster nicht erkannt werden, ist die Orientierung des Objekts undefiniert. Sie ist vom Benutzer einzugeben.

Um nicht bei jedem Erkennungsversuch die Objekte neu erlernen zu müssen, können die Muster in Dateien gespeichert werden. Die gespeicherten Muster werden als *Prototypen* bezeichnet. In der Initialisierungsphase werden die Prototypen ins CLAN geladen und stehen dann für die weitere Erkennung als bekannte Muster zur Verfügung. Die Speicherung eines Musters als Prototyp geschieht über die Grafikoberfläche. Abgespeichert werden die drei Merkmalsvektoren, der Name des Objektes, die Entfernung und die Orientierung. Jedes Merkmal eines Musters wird in eine eigenständige Datei abgelegt. Zusätzlich wird eine vierte Datei erzeugt, in der die restlichen Informationen, wie Name, Orientierung und Aufnahmeentfernung, eingetragen werden. Die Daten werden vom CLAN-Transputer an den ROOT-Transputer gesendet, die dieser dann direkt auf die Platte schreibt. Beim Laden der Prototypen liest der Root-Transputer die Daten aus den Dateien aus und gibt sie an das CLAN.

Da die Transputer keine direkte Verbindung zur Grafik auf einer Sun-Workstation haben, werden die Ergebnisse mit einem zusätzlichen Programm in einer Grafikoberfläche angezeigt. Die Kommunikation zwischen dem Root-Transputer und der Grafikoberfläche geschieht über eine Datei.

Erhält der Root-Transputer das Ergebnis vom CLAN-Transputer, wird es in eine Datei geschrieben. Die Grafikoberfläche wartet auf diese Angaben und liest sie aus, um dann das Ergebnis anzuzeigen. Um zu verhindern, daß die Grafikoberfläche schon versucht, das Ergebnis aus der Datei zu lesen, während sie noch beschrieben wird, wurde ein spezieller Mechanismus benutzt, über den die beiden Programme miteinander kommunizieren.

Die Grafikoberfläche wartet so lange, bis der Root-Transputer eine zweite Datei, im folgenden als *Lock-Datei* bezeichnet, erzeugt hat. Erst danach liest sie die Ergebnisse ein und zeigt sie an. Die Lock-Datei wird vom Root-Transputer geschaffen, wenn er alle Daten in die erste Datei geschrieben und diese wieder geschlossen hat. Sie enthält keine Daten. Mit ihr wird nur sichergestellt, daß die Grafikoberfläche die vollständige Ergebnisliste erhält.

Wurde vom Benutzer ein Name und der Winkel für ein neues Muster eingegeben, erfolgt die Übermittlung der Daten von der Grafikoberfläche zum CLAN-Transputer in umgekehrter Reihenfolge. Die Grafikoberfläche schreibt zuerst den Namen, die Orientierung und ein Signal, das anzeigt, ob das Muster als Prototyp auf Platte gespeichert werden soll, in eine Datei. Anschließend wird eine Lock-Datei erzeugt, um so ebenfalls die Vollständigkeit der Daten zu garantieren. Sobald die Lock-Datei existiert, liest der Root-Prozeß die Daten aus der ersten Datei und sendet dann die Information an den CLAN-Transputer, der nun den Namen und die Orientierung dem Muster zuordnet.

5.7.4 Sequentielle Version

Aus den beiden Prozessen der parallelen Version auf den Transputern ist die sequentielle Version auf einer Sparc1000 mit mehreren Prozessoren entstanden. Sie wurde erstellt, da die Lieferung der T9000 in der angekündigten Form nicht erfolgte. In dieser Version werden alle Module des Tip-Systems in einem Programm zusammengebunden. Als Ausgabemodul dient weiterhin die schon in der parallelen Version benutzte Grafikoberfläche.

Das CLAN mußte so modifiziert werden, daß die beiden Prozesse, die auf dem CLAN-Transputer bzw. auf dem Root-Transputer laufen, zu einem Prozeß zusammengefaßt werden. Es wird nun von der vorgeschalteten Verknüpfungsfunktion aufgerufen und erhält als Parameter die drei Merkmalsvektoren. Diese werden in der vorgestellten Form bewertet (siehe Kapitel 5.7.1). Die anschließende Kommunikation über Link entfällt, und es erfolgt die direkte Ausgabe der Ergebnisse an die Grafikoberfläche. Die Übertragung zur Grafikoberfläche verläuft über eine Datei, wie dies bereits in der parallelen Version geschieht. Außerdem können die Prototypen direkt gespeichert bzw. geladen werden.

5.7.5 Grafikoberfläche

Wie im letzten Abschnitt erwähnt, selektiert die Grafikoberfläche die Ergebnisse aus der Datei und zeigt sie an (Fig. 5.18). In der Anzeige erscheint der Objektname bzw. die Meldung

"new pattern", wenn das Objekt nicht erkannt wurde. Dahinter werden die x-, y- und z-Koordinaten des Objekts und seine Lage angezeigt. Die Lage gibt an, um wieviel Grad das Objekt aus einer fest definierten Position verdreht ist. Am Ende der Zeile steht die Bewertung der Erkennung. Die Grafikoberfläche zeigt immer die letzten zehn Ergebnisse an.

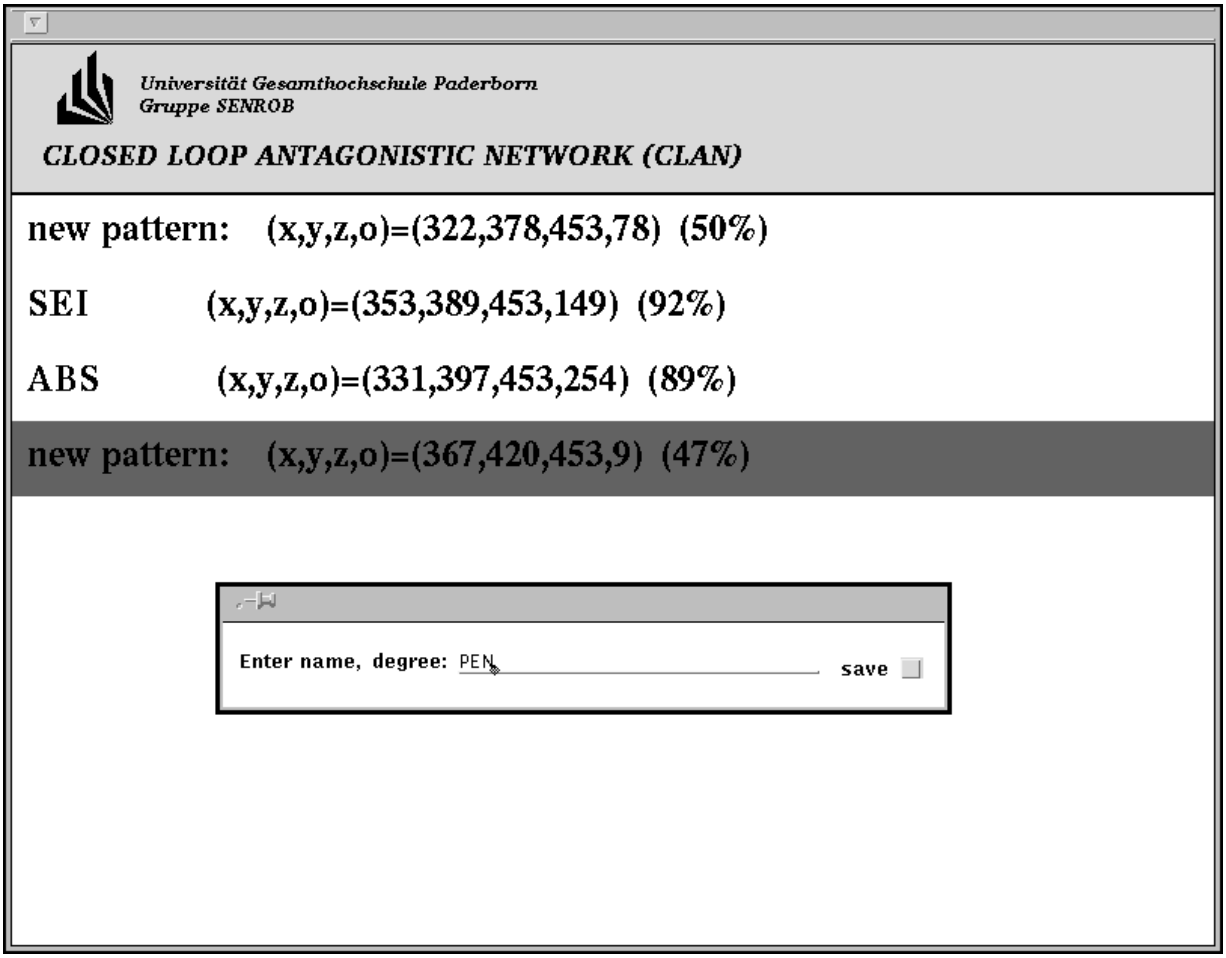


Fig. 5.18: Die Grafikausgabe für das komplette Erkennungssystem.

Um einen Namen für ein neues Objekt einzulesen, wird ein zusätzliches Fenster geöffnet, in das der Name eingegeben werden muß (Fig. 5.18, Mitte). Außerdem kann der Winkel des Objekts eingegeben werden. Wird kein Winkel angegeben, wird automatisch ein Winkel von 0° angenommen. Über "SAVE" kann ausgewählt werden, ob das Muster als Prototyp auf der Festplatte gespeichert werden soll.

5.8 Die Koppelung mit dem Roboter

Wie in Kapitel 1 berichtet wurde, soll das Erkennungsystem mit dem Montageplanungssystem der Dortmunder Gruppe verbunden werden, um Objekte zu greifen und zu montieren. Es wurde eine entsprechende Schnittstelle, die das Ethernet benutzt, definiert und realisiert.

Für das Greifen benötigt das Montageplanungssystem zunächst eine Kennung (Id), die angibt, welches Objekt erkannt wurde. Außerdem werden die x-, y- und z- Koordinaten, bezogen auf den Roboterfuß und die Orientierung des Objekts, übergeben. Das Erkennungssystem schreibt die Daten in dieser Version nicht direkt in die Datenbank des Montagesystems, sondern erst in eine Datei, die dann vom Montagesystem gelesen wird, um sie in die Datenbank einzutragen. In dieser ist bereits die übrige Information, die zum Greifen des Objektes erforderlich ist, enthalten.

Bis zum Abschluß dieser Arbeit konnte die Schnittstelle noch nicht in Betrieb genommen werden. Deshalb wurde eine Testumgebung geschaffen, um zu zeigen, daß mit den gefundenen Ergebnissen die Objekte gegriffen werden können. Hierzu wurde ein Greifersystem für den Roboter angeschafft. Bei der Aktivierung des Greifsystems wird dem CLAN von den vorgeschalteten Erkennungsmodulen über ein Flag mitgeteilt, daß die Objekte am Ende des Erkennungsvorgangs gegriffen werden sollen. Das CLAN schreibt den Namen, die Koordinaten und die Orientierung des erkannten Objekts in eine zusätzliche Datei. Die Syntax dieser Datei entspricht der definierten Schnittstelle zwischen Erkennungssystem und Montageplanungssystem. Wurden alle Objekte, die auf dem Tisch liegen, bearbeitet, wird durch eine Lock-Datei, wie sie schon bei der Grafikoberfläche beschrieben wurde, dem Programm zum Greifen der Objekte mitgeteilt, daß der Erkennungsvorgang abgeschlossen ist. Objekte, die nicht erkannt werden können, werden auch nicht abgespeichert, da das Greifsystem unbekannte Objekte nicht greifen kann.

5.9 Filter für die Aussortierung von Prototypen

Die Zeit, die das CLAN benötigt, um ein Muster zu klassifizieren, ist abhängig von der Anzahl der Muster, bei denen die aktiven Bits im Ergebnis der UND-Verknüpfung gezählt werden müssen (*Bitvergleich*). Deshalb wurde untersucht, wieweit es möglich ist, durch geschickte Auswahlverfahren die Anzahl der Bitvergleiche auf ein Minimum zu reduzieren [Pötzsch 1994]. Es hat sich gezeigt, daß durch einen einfachen Vergleich schon ein Großteil der Muster aussortiert werden kann, die überhaupt nicht mit dem präsentierten Vektor übereinstimmen können.

In einer ersten Stufe wird für den zu vergleichenden Mustersatz ein Referenzmuster bestimmt. Im allgemeinen Fall wird das erste präsentierte Muster als solches betrachtet. Es könnte zwar für jeden Mustersatz ein spezielles Referenzmuster erzeugt werden, jedoch liefert die Benutzung des ersten präsentierten Musters bereits eine hinreichende Genauigkeit. Wichtig ist nur, daß das Referenzmuster nicht ein Muster ist, das mit den später präsentierten Mustern keine Ähnlichkeit hat. Durch die Wahl des ersten präsentierten Musters dürfte dies auch in der Regel gegeben sein.

Wird ein neues Muster mit p gesetzten Bits präsentiert, dann wird mit dem Muster zuerst ein Bitvergleich mit dem Referenzmuster durchgeführt und so der Match m zwischen diesen beiden Mustern berechnet. Hierzu ist immer ein Bitvergleich notwendig. Hierbei wird außerdem gleichzeitig das gesamte Muster in vier gleich große disjunkte Segmente aufgeteilt, die zusammen das gesamte Muster ergeben. Für diese Segmente wird der Teilmatch m_i und die Anzahl der gesetzten Bits p_i berechnet. Hierzu sind keine weiteren aufwendigen Rechenoperationen erforderlich, da die Werte sowieso als Zwischenergebnisse vorliegen und nur umgespeichert werden müssen. Dies wurde vorher schon für alle anderen bereits gelernten Muster zum Referenzmuster gemacht und die entsprechenden Werte abgespeichert.

Nun wird der Match m und die Mustergröße p mit allen anderen Matches der Prototypen verglichen. Hierzu müssen nur zwei Integerzahlen überprüft werden. Weichen diese Werte zu sehr voneinander ab, wird der entsprechende Prototyp aussortiert, da das präsentierte Muster nicht mit dem entsprechenden Prototypen übereinstimmen kann. Somit braucht mit diesem Element kein aufwendiger Bitvergleich durchgeführt werden. So werden schon ca. 20% der Prototypen aussortiert.

Anschließend wird mit allen resultierenden Mustern das genannte Verfahren für die einzelnen Segmente und deren Matches m_i und Mustergröße p_i durchgeführt. Durch diese aufeinander folgenden Filter brauchen sehr viele Prototypen nicht mehr betrachtet werden. Auf der anderen Seite werden aber keine Muster, zu denen das präsentierte Muster paßt, aussortiert.

Genaue Zahlen, wieviele Muster aussortiert werden, lassen sich nicht angeben, da die Zahl der aussortierten Prototypen sehr stark von den betrachteten Mustern abhängt.

Diese Version liefert eine gute Beschleunigung des CLANs. Dennoch wurde sie nicht in das Erkennungssystem integriert, da die betrachteten Werkstücke eine unterschiedliche Größe aufweisen. So ist es möglich, den größten Teil der gelernten Muster mit Hilfe des Erkennungsintervalls auszusortieren. Die vorgestellte Filtermethode bringt aber große Vorteile, wenn viele Objekte in der gleichen Entfernung aufgenommen werden.

5.10 Ergebnisse bei der Arbeit mit Crandfieldteilen

In unserer Arbeitsgruppe wird hauptsächlich mit dem Cranfieldmontagesatz (Fig. 5.19) gearbeitet. Neben den ursprüglichen Cranfieldteilen wurden fehlerhafte Teile angefertigt, um zu zeigen, inwieweit das System in der Lage ist, ähnliche Teile voneinander zu unterscheiden. Auf der anderen Seite muß das System so fehlertolerant sein, gleiche, aber unterschiedlich aufgenommene Objekte als gleiche Objekte wiederzuerkennen. Die Erkennungsversuche blieben aber nicht nur auf die Cranfieldteile beschränkt, ebenso wurden Stifte, Klebebandroller, Markierer usw. untersucht (Fig. 5.20).

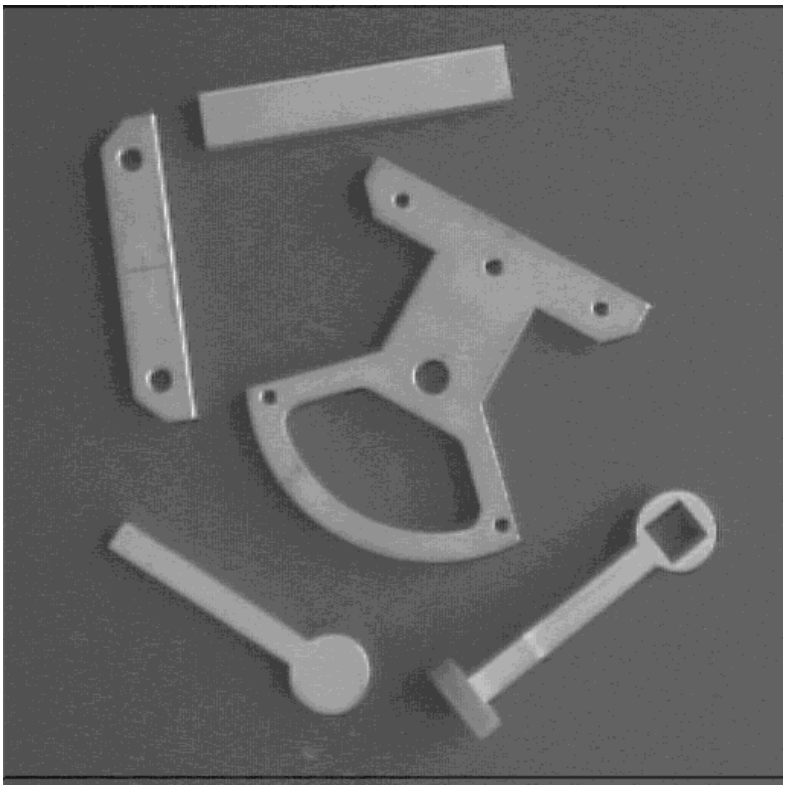


Fig. 5.19: Verschiedene Teile des Cranfieldmontagesatzes und speziell gefertigte fehlerhafte Cranfieldteile.

Zu Beginn eines Erkennungsversuches müssen alle Teile in ihren Vorzugsorientierungen gelernt werden. Theoretisch könnten sie vor jedem Erkennungsversuch neu erzeugt werden, doch ist dies sehr umständlich. Deshalb werden sie in einem vorgeschalteten Schritt erzeugt und bei der Initialisierung geladen.

Das beschriebene Erkennungssystem erkennt Objekte zu einem großen Prozentsatz wieder. Nur in wenigen Ausnahmefällen erfolgt eine Nicht- oder Fehlklassifizierung. Mit der ermittelten Orientierung und den Koordinaten können die Objekte gegriffen werden.

Fig. 5.20: Teile, die klassifiziert werden.

5.11 Ergebnisse bei der Ziffernerkennung

Neben den Objekten aus Fig. 5.19 und 5.20 wurde Untersuchungen durchgeführt handgeschriebene Ziffern zu erkennen. Es wurde speziell betrachtet, ob die auftretenden Abweichungen vom gelernten Konturverlauf toleriert werden. Zur Verfügung stand uns ein vom Daimler-Benz-Forschungsinstitut in Ulm bereitgestellter Ziffernsatz. Um Mißverständnisse zu vermeiden, sei betont, daß diese Versuche nicht dazu dienen sollten, eine neuartige oder gar überlegene Methode für die Erkennung handgeschriebener Ziffern zu schaffen. Das Ziel war es vielmehr, zu zeigen, daß ein auf grundsätzlichen Überlegungen basierendes Erkennungssystem gute Leistungen auch in solchen Anwendungsgebieten erbringen kann.

Grundsätzlich hätten für die Erkennung handgeschriebener Ziffern die drei Merkmalsklassen benutzt werden können. Da aber die meisten Verfahren zur Ziffernerkennung derzeit auf flächenbasierten Merkmalen beruhen, interessierte bei den ersten Versuchen vor allem die Erkennung auf der Basis von Konturmerkmalen, was ein späteres Einbeziehen von Flächenmerkmalen und Eckenmerkmalen nicht ausschließt.

Der Ziffersatz des Daimler-Benz-Forschungsinstituts bestand aus zehnmal 2000 handgeschriebenen Ziffern. Leider handelte es sich dabei um Grauwertmatrizen mit 16 x 16 Feldern, aus denen die Konturauflösung nicht in der von uns gewünschten Auflösung extrahiert werden konnte. Eine Alternative hätte darin bestanden, einen Ziffernsatz selbst zu schreiben und ihn mit der Kamera des SENROB-Systems aufzunehmen. Wegen des damit verbundenen Aufwands, vor allem aber wegen der dann nicht mehr gegebenen Vergleichbarkeit mit Ergebnissen anderer Verfahren, wurde auf diese Alternative verzichtet.

Deshalb wurden die Ziffern auf 96 x 96 Matrizen umgerechnet, die dabei entstehenden "schach-brettartigen" Konturverläufe (Fig. 5.21, oben) durch einen 15 x 15 Gaußfilter geglättet und schließlich noch einmal um den Faktor zwei auf 192 x 192 vergrößert. Diese Bilder wurden mit einer durch das Grauwerthistogramm gesteuerten dynamischen Schwelle binärisiert und in dieser Form in das SENROB-System eingelesen (Fig. 5.21, unten). Diese Auflösung entspricht einer normierten Repräsentation, d.h. höher aufgelöste Bilder von nähergelegenen Objekten werden über die parametrischen Abbildungen ohnehin auf diese Auflösung reduziert.

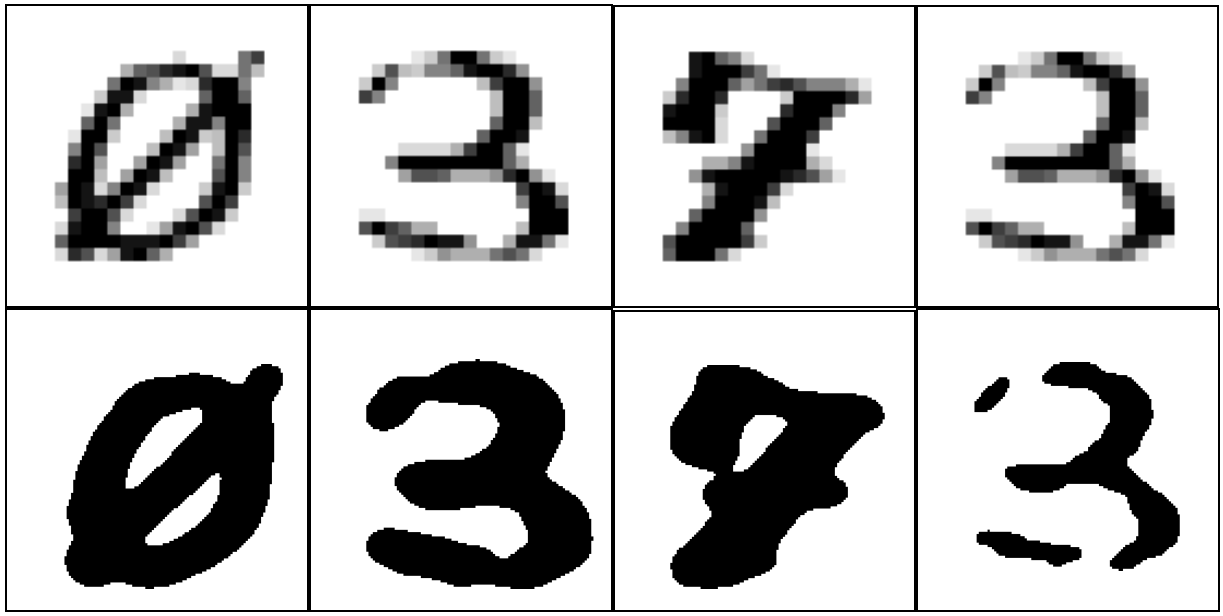


Fig. 5.21: Verschiedene Ziffern dargestellt in der 16 x 16 Grauwertmatrix (oben) und nach durchgeführter Glättung und Vergrößerung (unten).

Im Robotik-Betrieb erfolgt die grobe Blickpunktsteuerung über die Bewegung der Roboterhand, die Feinsteuerung über die Auswahl des Blickpunktes in der wesentlich größeren 512 x 512

Matrix der Kamera. Für die Fovealisierung der Ziffern und ihre Umrechnung in das Retina-Koordinatensystem wurden zwei Alternativen untersucht. Im ersten Fall wurde der Mittelpunkt des 192 x 192 Bildes der Ziffer als Zentrum der Retina gewählt, im zweiten Fall der Flächenschwerpunkt der jeweiligen binarisierten Ziffer. Da sich bei den ersten Vergleichen keine deutlichen Unterschiede zwischen beiden Verfahren erkennen ließen, wurde für die weiteren Untersuchungen der Mittelpunkt der 192 x 192 Matrix gewählt.

Nach der Transformation des Bildes in die künstliche Retina wurden die Kantenelemente detektiert und die beschriebenen ortstoleranten Konturrepräsentationen erzeugt. Fig. 5.22 zeigt typische Konturrepräsentationen unterschiedlicher Ziffern.

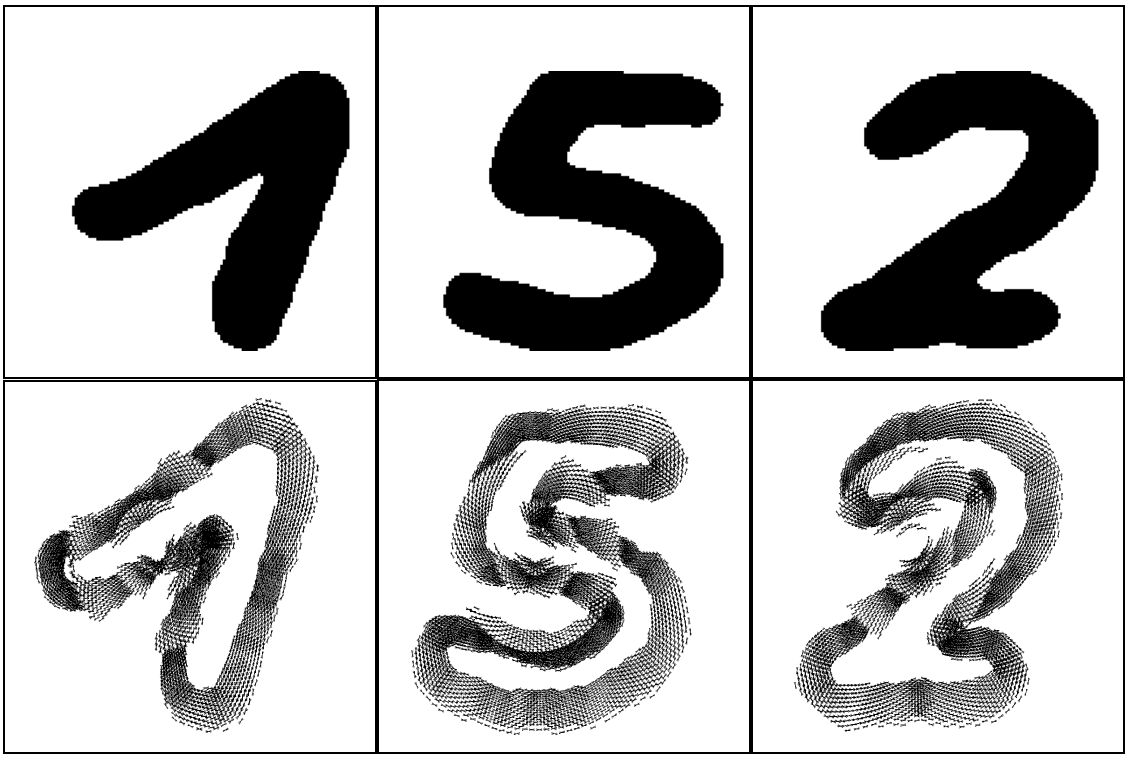


Fig. 5.22: Konturrepräsentation dreier Ziffern (oben) durch (unten) vgl. [Kräuter 1995].

Selbstverständlich ist die Umsetzung von 16 x 16 auf 192 x 192 Matrizen mit Qualitätsverlusten behaftet, die beim direkten Aufnehmen bzw. Einscannen der Ziffern vermieden würde. Besonders störend ist die beim Glätten auftretende Verbreiterung der Linien, die oft zu Verbindungen zwischen vorher getrennten Strukturen führt (Fig. 5.21c). Andererseits führt der Versuch, solche Verbreiterungen zu vermeiden, häufig zu einem Zerfall vorher zusammenhängender Strukturen (Fig. 5.21d). Da aber die hier beschriebene Umsetzung von 16 x 16 auf 192 x 192 Matrizen später ohnehin durch ein direktes Einscannen ersetzt werden soll, wurde

der Aufwand hier nicht weiter erhöht, etwa durch den Einsatz morphologischer Operatoren. Dies bedeutet aber, daß sich die Erkennungsraten bei besserer Vorverarbeitung durchaus noch erhöhen lassen.

Wegen der aufwendigen Vorverarbeitung wurden in die erste Versuchsreihe nur jeweils die ersten 1000 Exemplare jeder Ziffer einbezogen. Von diesen jeweils 1000 Exemplaren pro Ziffer wurden wieder die ersten 400 als Lernsatz abgespaltet und die Erkennungsversuche anhand der als Testsatz verbleibenden 600 Exemplare je Ziffer durchgeführt.

Bei der Arbeitsweise des CLAN werden im Prinzip alle 400 Exemplare der Lernstichprobe einer Ziffer bereits nach einmaliger Präsentation einer Menge von Subklassen zugewiesen. Allerdings würde dann immer das erste Element der Lernstichprobe Prototyp der ersten Subklasse und das jeweils erste nicht dazu passende Element der Lernstichprobe Prototyp einer weiteren Subklasse. Dies kann im Extremfall dazu führen, daß das Prototyp-Element als einziges im Zentrum des von ihm aufgespannten Klassenbereichs liegt und alle anderen zu dieser Klasse passenden Elemente sich als Cluster am seinem Rand ansammeln. Um dies zu vermeiden, wurde der Lernprozeß in drei Schritten wie folgt durchgeführt: Zunächst wurde mit einer sehr hohen Schwelle für das Ähnlichkeitsmaß von $q_1 = - 0.3$ gelernt, wodurch nur im Merkmalsraum eng benachbarte Elemente einer Subklasse zugewiesen wurden. Bei einem zweiten Durchgang wurden nur die sechs bis acht Subklassen mit den meisten Elementen beibehalten und die Klassenbildung mit einer niedrigeren Schwelle von $q_2 = - 0.4$ für die restlichen Elemente der Lernstichprobe wiederholt. Erst beim dritten Durchlauf mit $q_3 = - 0.5$ wurden alle entstandenen Subklassen beibehalten. Diese sukzessive Verringerung des Ähnlichkeitsmaßes sorgte dafür, daß zumindest eine gewisse Häufung von Elementen im Zentrum der größeren Subklassen auftrat und sich die Zahl der Subklassen je Ziffer verringerte. Die Ziffer **6** benötigte mit 36 die geringste Zahl an Subklassen, während die Ziffer **4** mit 131 die höchste Zahl an Subklassen erreichte.

Bei der Präsentation der Teststichprobe von je 600 Elementen pro Ziffer wurden diese mit den gelernten Prototypen verglichen. Für jede zu erkennende Ziffer wurden alle Subklassen bestimmt, bei denen für das Ähnlichkeitsmaß $q > - 0.5$ galt. Da im allgemeinen eine Ziffer mehreren Ziffernklassen zugeordnet werden konnte, wurde eine Mehrheitsentscheidung herbeigeführt (Mehrheitsmethode). Die Ziffer wurde also der Ziffernklasse zugeordnet, in der sich die meisten ähnlichen Prototypen befanden. Nur wenn sich damit keine Entscheidung herbeiführen ließ, wurde die Ziffer der Klasse zugewiesen, mit der sie aufgrund des Ähnlichkeitsmaßes am besten übereinstimmte. Diese Vorgehensweise zeigte deutlich weniger Fehlklassifikationen als eine Zuordnung allein auf der Basis des Ähnlichkeitsmaßes (Maximumsmethode, vgl. Tabelle 5.3).

Klassifikations-methode	richtig erkannte Ziffern	Fehl-klassifikationen	Zurück-weisungen
größte Ähnlichkeit	90,50%	4,47%	5,03%
Mehrheit der Überschwelligen	89,72%	1,98%	8,30%
Mehrheit der Beiden Besten	85,00%	1,10%	13,9%

Tab. 5.3: Gegenüberstellung der Ergebnisse für die verwendeten Auswertungsmethoden

Die '**3**' ist die Ziffer, die am häufigsten fehlklassifiziert wurde. An dieser Stelle sollen einige Beispiele gezeigt und diskutiert werden.

Folgende '**3**en' wurden z.B. als '**5**' fehlklassifiziert:

Folgende '**3**en' wurden z.B. als '**9**' fehlklassifiziert:

Folgende '**3**en' wurden z.B. als '**8**' fehlklassifiziert:

Man erkennt deutlich, daß diese Fehler auf die starke Verbreiterung der Konturen und das damit verbundene Zusammenfließen ursprünglich getrennter Konturelemente aufgrund der Filterung zurückzuführen sind. Diese Gruppe der fehlklassifizierten Ziffern macht etwa die Hälfte der gesamten Fehlklassifikationen aus, so daß eine deutliche Verbesserung des Ergebnisses bei einer Verwendung von höher aufgelösten Vorlagen zu erwarten ist.

6. Theorie des Raumrepräsentationsnetzwerkes

Im Gehirn wird Wissen über Positionen und Formen von Objekten und Hindernissen in unserer Umgebung offensichtlich repräsentiert. Außerdem werden visuelle Informationen von verschiedenen Ansichten und Informationen von weiteren Sensoren zu einer räumlichen Vorstellung verarbeitet. Diese Vorstellung scheint in unserem köperfesten Koordinatensystem stabil zu sei. Dies gilt sogar, wenn wir uns ohne sensorischen Input bewegen, z. B. mit geschlossenen Augen. Wir haben jederzeit eine Vorstellung von dem Raum, in dem wir uns bewegen (Fig. 6.1). Von einem Objekt, das wir anschauen, uns dann langsam um 90 Grad drehen, können wir sagen, daß es neben uns liegt, ohne es von neuem zu betrachten.

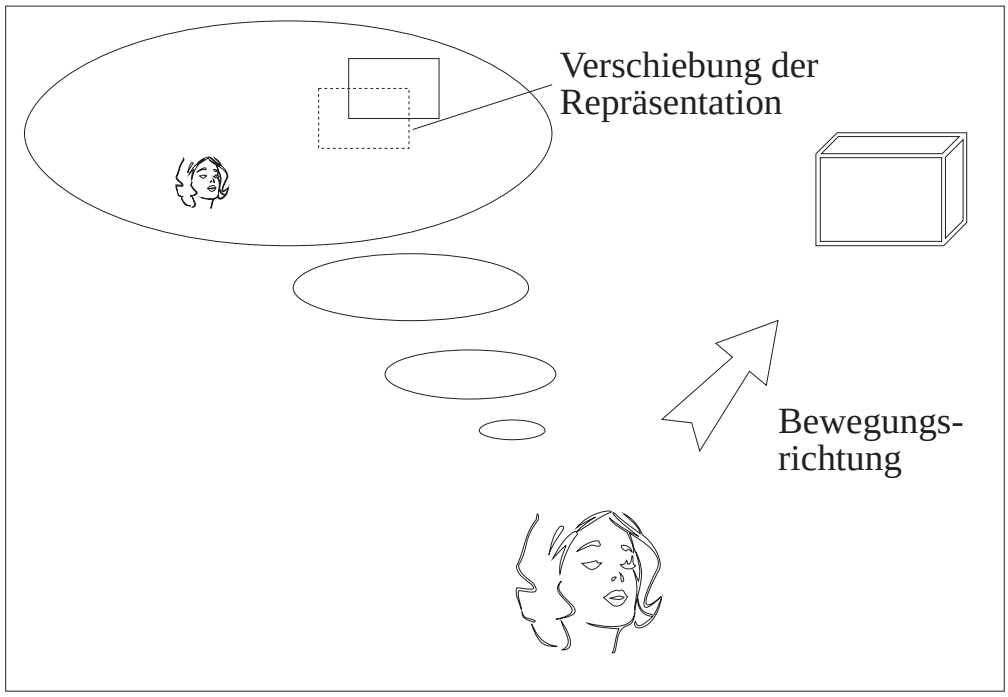


Fig. 6.1: Bei Bewegungen verschiebt sich die interne Repräsentation von Objekten im Gehirn entgegengesetzt.

In dem *Raumrepräsentationsnetzwerk* (*RRN*) werden Neuronen (*Gitterneuronen*) lokalen Volumenelementen außerhalb des Körpers zugeordnet. In dem entstehenden körperbezogenen *neuronalen Gitter* wird ein Objekt durch eine *Aktivitätswolke* von aktiven Neuronen dargestellt (Fig. 6.2). Bei Bewegungen wird das neuronale Gitter gegen die Objekte verschoben. Hieraus folgt, daß ein Objekt bzw. dessen Aktivitätswolke sich in dem neuronalen Gitter in die entgegengesetzte Richtung bewegen muß. Eine Aktivitätswolke kann sich nur verschieben, indem zusätzliche Neuronen am Anfang der Wolke aktiviert und andere Neuronen am

Ende der Wolke deaktiviert werden. Vereinfacht gesagt: Aktivität muß bei einer Bewegung erzeugt und gelöscht werden.

Die einzelnen Bewegungskomponenten lassen sich näherungsweise linear kombinieren, wenn der Raum nach bestimmten Regeln in kleine Domänen aufgeteilt wird. Dies erlaubt es, die Betrachtung mit einer eindimensionalen Wolke und der Bewegung in eine Richtung zu beginnen. Anschließend wird der Mechanismus auf zweidimensionale und später auf dreidimensionale Translationen und Rotationen erweitert.

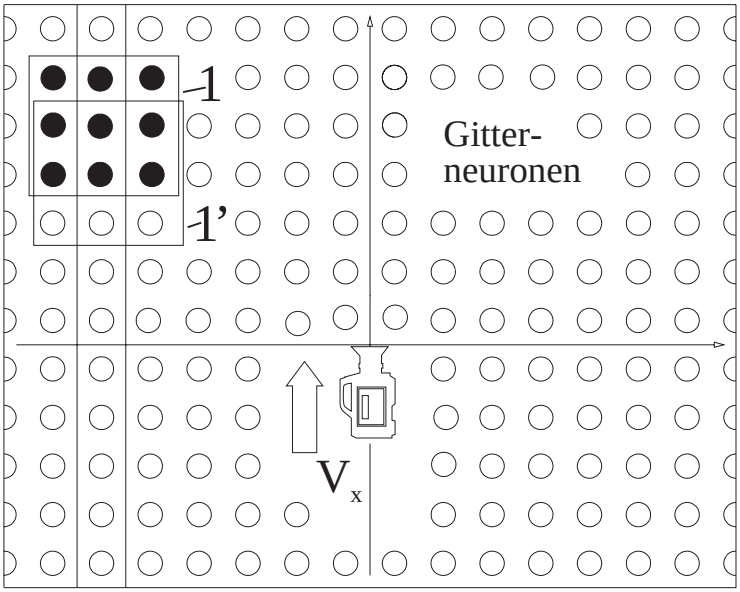


Fig. 6.2: Repräsentation in einem körperfesten Koordinatensystem.

Es läßt sich zeigen, daß der dreidimensionale Mechanismus sich in eine zweidimensionale biologische Gehirnstruktur einbetten läßt. Das entstehende Modell hat Ähnlichkeit mit dem *Cerebellaren Cortex* [Hartmann 1992a und Hartmann 1992c].

Weitere neuronale raumrepräsentierende Modelle wurden in der Literatur nur wenige gefunden. S. Grossberg et. Al. arbeiten z.B. ebenfalls in einem körperfesten Koordinatensystem, wobei die Lage eines Objekts durch den Abstand zum Kopf und den horizontalen und vertikalen Winkeln bezogen auf die Kopfposition beschrieben wird [Grossberg 1993]. R. Hosaka und T. Nagano kodieren die Objektbeziehungen und bildet so eine Vorstellung vom Raum [Hosaka 1993]. Durch das Anschauen eines Objektes werden alle Objekte im Raum sichtbar.

6.1 Translatorische Bewegungen

Zuerst wird ein eindimensionales Gitter betrachtet (Fig. 6.3). Die Modellneuronen im Netzwerk werden durch eine Software-Simulation der pulscodierten Neuronen simuliert [vgl.

Kapitel 3.4]. Alle Neuronen in Fig. 6.3, die ein Volumenelement beschreiben, an dem sich ein Teil eines Objekts befindet, werden durch einen Impuls aktiviert, der durch einen sensorischen Input erzeugt wird. Sie bleiben ohne weiteren Impuls vom sensorischen Input über eine positive Rückkopplung aktiv (Fig. 6.3) und repräsentieren zusammen die Objekte (Neuron 2 und 3).

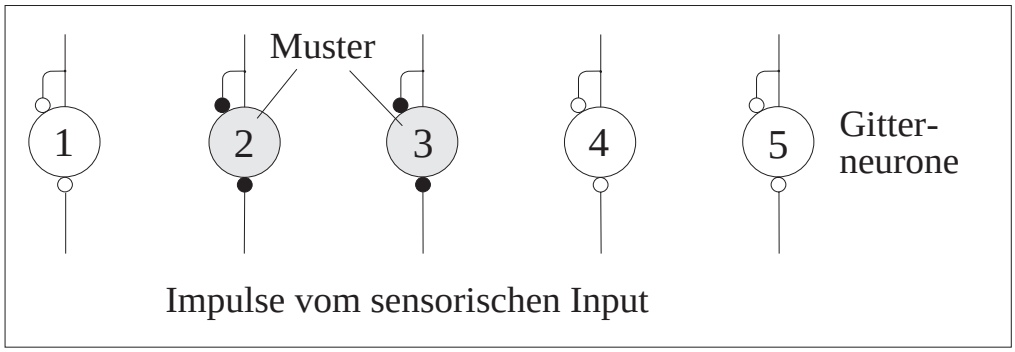


Fig. 6.3: Die Neuronen des eindimensionalen Gitters.

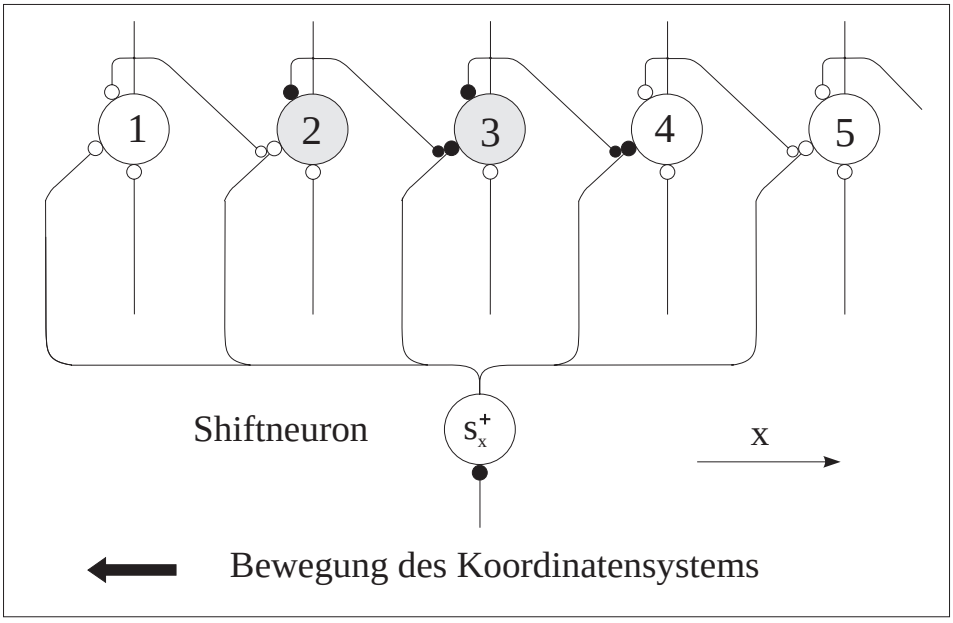


Fig. 6.4: Aktivierung von Neuronen im Gitter.

Wenn sich das Koordinatensystem nach links bewegt, muß sich das Aktivitätsmuster nach rechts verschieben. In Abhängigkeit von der Geschwindigkeit müssen Neuronen auf der rechten Seite des Aktivitätsmusters aktiviert bzw. auf der linken Seite deaktiviert werden. Die Verschiebung ist vergleichbar mit dem Schieben in einem Schieberegister.

Es wird ein *Shiftneuron* s_x^+ eingeführt, das eine Spikerate proportional zur Geschwindigkeit erzeugt und zu jedem Gitterneuron eine Synapse hat, die als *Shiftsynapse* bezeichnet wird. Ein Spike vom Shiftneuron wird somit zu allen Gitterneuronen geleitet (Fig. 6.4). Dort ist das synaptische Gewicht so hoch eingestellt, daß durch einen einzelnen Spike des Shiftneurons alle Gitterneuronen aktiviert würden. Ein Shiftneuron erregt ein Gitterneuron aber nur dann, wenn die Shiftsynapse durch eine präsynaptische Erregung vom linken Nachbarn freigeschaltet wurde. Dies gilt nur für die Shiftsynapsen von Neuron 3 und 4 in unserem Beispiel (schwarz in Fig. 6.4), und somit kann der Shiftimpuls nur zu diesen Neuronen gelangen. Da Neuron 3 bereits aktiv ist, wird nur noch Neuron 4 aktiviert. Zwischen dem Erregen von Neuron 4 und dessen erstem Spike vergeht nur kurze Zeit. Aus diesem Grund bewirkt der erste Spike von Neuron 4 eine sofortige Durchschaltung der Shiftsynapse von Neuron 5. Durch eine große Zeitkonstante bleibt die präsynaptische Erregung so lange aktiv, wie Neuron 4 aktiv ist. Somit kann mit dem nächsten Shiftimpuls Neuron 5 aktiviert und die Shiftsynapse von Neuron 6 durchgeschaltet werden.

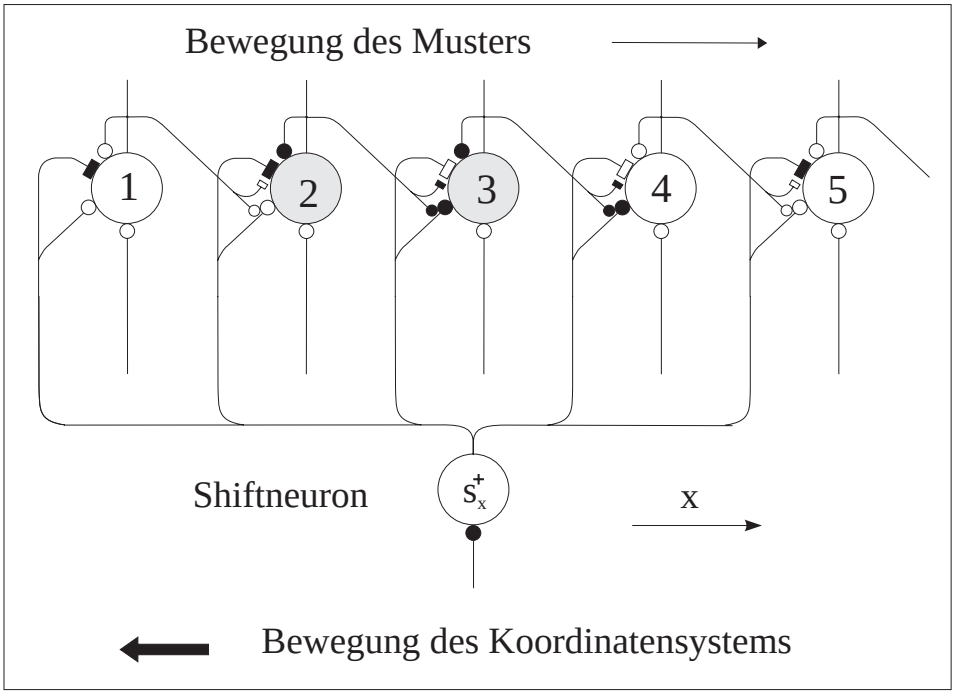


Fig. 6.5: Deaktivierung von Neuronen im Gitter.

Mit einem Shiftimpuls muß aber nicht nur Aktivität auf der rechten Seite erzeugt, sondern auch Aktivität auf der linken Seite gelöscht werden. Dies geschieht durch zusätzliche hemmende Shiftsynapsen von den Shiftneuronen in Fig. 6.5. Diese Synapsen werden durch präsynaptische hemmende Verbindungen vom linken Nachbarn beeinflußt. Diese hemmende

Verbindung blockiert einen Shiftimpuls, wenn der linke Nachbar aktiv ist. Daraus folgt, daß ein Shiftimpuls nur ein Neuron deaktivieren kann, wenn dessen linker Nachbar passiv ist. Dies gilt nur für die Neuronen 1, 2 und 5 in Fig. 6.5. Da die Neuronen 1 und 5 nicht aktiv sind, kann der Shiftimpuls nur Neuron 2 ausschalten.

Durch die Überlagerung der beiden Mechanismen zum Erzeugen und Löschen von Aktivität wird mit einem Shiftimpuls das aktive Muster jeweils um einen Platz nach rechts bewegt. Bis auf das gemeinsame Shiftneuron sind beide Mechanismen unabhängig voneinander, da sie keine gemeinsamen Verbindungen benutzen. Es können, also beliebig große Muster verschoben werden.

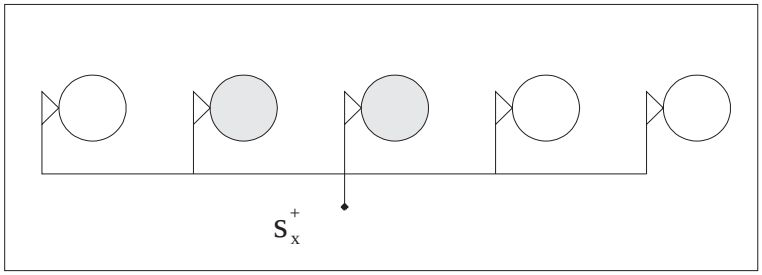


Fig. 6.6: Die Verbindungsstruktur aus Fig. 6.5 in symbolischer Darstellung.

Bevor wir zur Bewegung nach links kommen und danach den Shiftmechanismus auf ein zwei- und dreidimensionale Gitter ausweiten, vereinfachen wir die Darstellung zu einer symbolischen Beschreibung. In Fig. 6.6 werden die Verbindung für die Impulse vom sensorischen Input, die Axone und die Rückkopplungsverbindungen weggelassen. Vom Shiftneuron wird eine Verbindung zu jedem Gitterneuron gezeichnet, die mit einem dreieckigen Symbol links neben jedem Gitterneuron endet. Dieses Symbol ersetzt alle Verbindungen, die für den Shift nach rechts benötigt werden. Somit ist Fig. 6.6 eine Vereinfachung der Beschreibung in Fig. 6.5.

Durch ein zweites Shiftneuron s_x^- und die spiegelbildlichen synaptischen Verbindungen ergibt sich ein Mechanismus, mit dem das Muster mit jedem Shiftimpuls nach links verschoben wird. Fig. 6.7 mit den Dreiecken auf der rechten Seite ist eine vereinfachte Beschreibung der Shiftbewegung nach links. Beide Shiftmechanismen können überlagert werden (Fig. 6.8), und man erhält einen Mechanismus für Bewegungen nach rechts und links, je nachdem welches Shiftneuron aktiv ist.

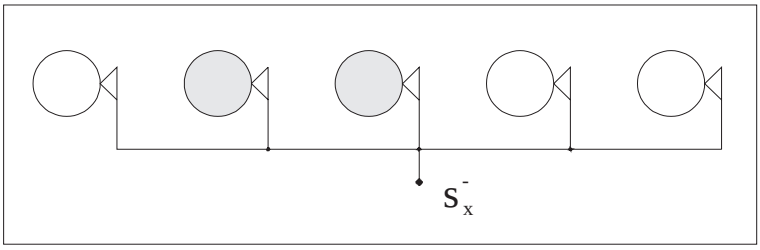


Fig. 6.7: Die Verbindungsstruktur für die Verschiebung nach links.

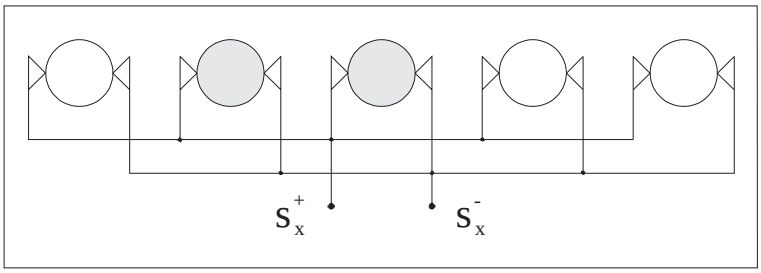


Fig. 6.8: Beide Bewegungsrichtungen werden überlagert.

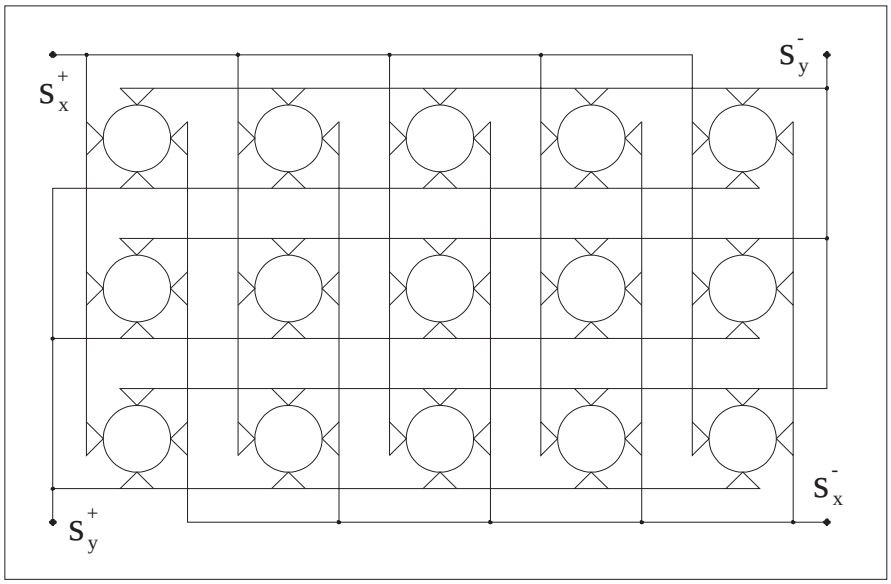


Fig. 6.9: In der zweidimensionalen Version sind Bewegungen nach "oben", "unten", "links" und "rechts" möglich.

Nun kann der eindimensionale Mechanismus auf den zweidimensionalen Fall ausdehnen werden (Fig. 6.9). Die gemeinsame Verbindungsstruktur für Nachbarn in einer Zeile sowie in einer Spalte ist in Fig. 6.9 dargestellt. Es gibt Shiftneuronen für die Richtungen "oben" und

"unten" bzw. "rechts" und "links". Ein aktives Shiftneuron bewirkt eine Bewegung entlang der entsprechenden Achse. Inaktive Shiftneuronen beeinflussen die Bewegung nicht. Die Bewegungen "oben" und "unten" oder "rechts" und "links" werden niemals gleichzeitig aktiviert, da ein System sich nicht gleichzeitig in entgegengesetzte Richtungen bewegen kann. Jedoch werden Bewegungen nach "unten" oder "oben" gleichzeitig mit Bewegungen nach "rechts" oder "links" durchgeführt, da diese möglich, sogar wünschenswert sind. Weil die Shiftneuronen unabhängig arbeiten, ist das Zusammentreffen der Shiftimpulse zufällig. Simulationen zeigen, daß hierdurch das sich bewegende Muster nicht nennenswert beeinflußt wird.

Durch die Hinzunahme von zwei weiteren Shiftneuronen s_z^+ und s_z^- und deren entsprechenden Verbindungen wird das System auf den dreidimensionalen Fall erweitert. Mit dem entstandenen neuronalen Netz ist es möglich, alle translatorischen Bewegungen im dreidimensionalen Raum nachzubilden.

6.2 Rotation

Bisher kann sich das zweidimensionale Koordinatensystem mit dem Geschwindigkeitvektor $\mathbf{V} = (V_x, V_y)$ bewegen. Nun wird ein Winkelgeschwindigkeitsvektor $\vec{\Omega}$ hinzugefügt. Zum Verständnis des folgenden Berechnungen gilt es, sich einzuprägen, daß $\mathbf{V}$ und $\vec{\Omega}$ sich auf das bewegte körperfeste Koordinatensystem beziehen, z.B. auf die sich bewegende Kamera (Fig. 6.2), und nicht auf das feststehende Koordinatensystem eines externen Beobachters. Die Kamera kann sich vorwärts, rückwärts und im Kreis bewegen. Der Mittelpunkt für die Drehung ist jeweils der Ursprung des sich bewegenden Systems (Fig. 6.10). Der Vektor $\vec{\Omega}$ steht senkrecht auf der x, y - Ebene.

Wenn sich das Koordinatensystem dreht, muß sich die Aktivitätswolke innerhalb des neuronalen Gitters entgegengesetzt drehen. Der einzige Mechanismus, um die Aktivitätswolke zu bewegen, ist das Erzeugen und Löschen von Aktivitäten durch benachbarte Neuronen in der gleichen Zeile bzw. Spalte. Im folgenden wird erläutert, wie es mit dem Mechanismus näherungsweise möglich ist, die Rotation zu simulieren.

Die Verschiebung der Aktivitätswolke in einer Zeile erfolgt mit der Geschwindigkeit v_x. In einer Spalte bewegt sich die Wolke mit der Geschwindigkeit v_y. Die relative Geschwindigkeit $\mathbf{v} = (v_x, v_y)$ der Aktivitätswolke ist abhängig von dem sich bewegenden Koordinatensystem. Aber $\mathbf{v} = (v_x, v_y)$ darf nicht mit der Geschwindigkeit $\mathbf{V} = (V_x, V_y)$ des Koordinatensystems verwechselt werden, denn ein Muster bewegt sich mit der Geschwindigkeit $\mathbf{v} \neq 0$ bei $\mathbf{V} = 0$,

wenn sich das Koordinatensystem mit der Winkelgeschwindigkeit $\vec{\Omega}$ wie in Fig. 6.10 bewegt. Außerdem variiert v (k, l) bei unterschiedlichen Positionen (x = k, y = l) innerhalb des neuronalen Gitters.

Die Geschwindigkeit **V** und die Winkelgeschwindigkeit $\vec{\Omega}$ des sich bewegenden Koordinatensystems sind global. Auf der anderen Seite sind v_x und v_y lokal und hängen von der Position (k, l) innerhalb des neuronalen Gitters ab. Um die Beziehung zwischen den lokalen Werten v_x und v_y und den globalen Werten V_x, V_y und $\vec{\Omega}$ zu erklären, diskutieren wir zuerst den einfachen Fall der translatorischen Bewegungen ($\vec{\Omega}$ = 0). In diesem Fall sind die Beziehungen

$$v_{xt}(\mathbf{V}) = -V_x \text{ und } v_{yt}(\mathbf{V}) = -V_y \qquad (6.1)$$

unabhängig von der Position innerhalb des neuronalen Gitters.

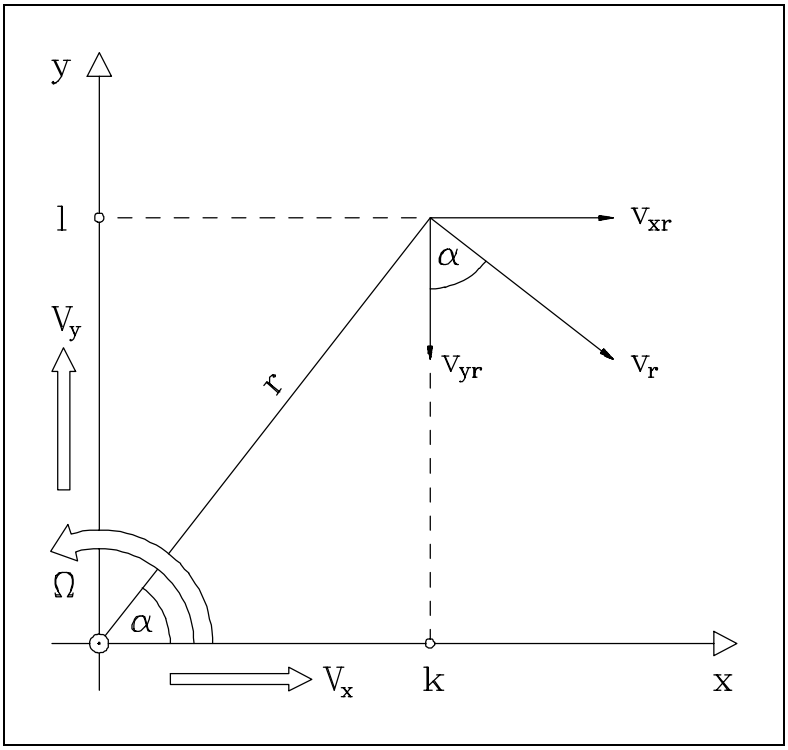


Fig. 6.10: Die Winkelgeschwindigkeit $\vec{\Omega}$ mit Hilfe der Geschwindigkeiten v_{xr} und v_{yr}.

Der Index t zeigt an, daß v_{xt} und v_{yt} sich auf translatorische Bewegungen beziehen, während die Geschwindigkeiten v_{xr} und v_{yr} für Drehungen mit einem r gekennzeichnet werden. Der Geschwindigkeitsvektor v_r an den betrachteten Koordinaten (k, l) steht immer senkrecht auf dem Radius r (Fig. 6.10). Sein Betrag $v_r = |v_r|$ ist durch $v_r = r \cdot \vec{\Omega}$ gegeben, und die Aufteilung von v_r in seine x- und y-Komponenten ist

$$v_{xr}(\vec{\Omega}) = r \cdot \sin\alpha\ \vec{\Omega} \text{ und } v_{yr}(\vec{\Omega}) = -r \cdot \cos\alpha \cdot \vec{\Omega}. \quad (6.2)$$

Da $r = (k^2 + l^2)^{1/2}$ und $\alpha = \arctan(l / k)$ nur von k und l abhängen, können die Faktoren $w_x(k, l) = r \cdot \sin\alpha$ und $w_y(k, l) = r \cdot \cos\alpha$ für jeden Knoten innerhalb des Gitters definiert werden. Nun können wir (6.2) durch

$$v_{xr}(\vec{\Omega}, k, l) = w_x(k, l) \cdot \vec{\Omega} \text{ und } v_{yr}(\vec{\Omega}, k, l) = -w_y(k, l) \cdot \vec{\Omega} \quad (6.3)$$

ersetzen, um die Abhängigkeit von v_{xr} und v_{yr} von (k, l) zu zeigen. Die Werte v_t der translatorischen Bewegung und v_r der rotatorischen Bewegung können zu $v = v_t + v_r$ mit den Komponenten

$$v_x(\mathbf{V}, \vec{\Omega}, k, l) = -V_x + w_x(k, l) \cdot \vec{\Omega} \text{ und } v_y(\mathbf{V}, \vec{\Omega}, k, l) = -V_y - w_y(k, l) \cdot \vec{\Omega} \quad (6.4)$$

kombiniert werden. Die Komponenten v_x und v_y der relativen Bewegung der Aktivitätswolke sind gemäß (6.4) linear abhängig von V_x, V_y und $\vec{\Omega}$ mit den Faktoren w_x und w_y.

Die Faktoren w_x und w_y sind abhängig von der Lage der Wolke. Aus diesem Grund wird das neuronale Gitter derart modifiziert, daß Bewegungen von Aktivitäten mit lokal verschiedenen Geschwindigkeiten zugelassen werden. Es werden Shiftneuronen eingeführt, die ortsabhängige Spikeraten erzeugen, die proportional zu $v_x(k, l)$ und $v_y(k, l)$ sind. Die ursprüngliche Idee war, für alle Gitterneuronen ein Shiftneuron zu benutzen, aber in diesem Fall hätte jedes Gitterneuron vier Shiftneuronen. Deshalb wurde das neuronale Gitter in kleine Domänen von 8 x 8 = 64 Neuronen aufgeteilt. Die Neuronen einer Domäne werden gemeinsam durch Shiftimpulse von den vier Shiftneuronen angesteuert. Verglichen mit der Anzahl der Gitterneuronen ist die Zahl der Shiftneuronen nun unbedeutend klein.

Der Fehler, der durch den gemeinsamen Shiftimpuls für alle Neuronen einer Domäne entsteht, beeinflußt eine rotierende Wolke nur in geringem Maße. Dies gilt besonders, wenn die Größe der Domänen im Vergleich zu r klein ist. Sehr gut geeignet ist ein Polarkoordinatensystem, in dem die Domänengröße proportional zum Radius gewählt wird. Dort ist der Fehler überall im Gitter gleich [Bredenbals 1993].

Gemäß der Beschreibung versorgen Shiftneuronen jeweils eine Domäne. Deshalb werden von nun an die einzelnen Domänen mit (k, l) bezeichnet. Es gibt vier Shiftneuronen s_x^+ (k, l), s_x^- (k, l), s_y^+ (k, l) und s_y^- (k, l) für eine Domäne (k, l), die die Wolke in x- und y-Richtung vor (+) und

zurück (-) bewegen. Die ankommenden Impulse (Fig. 6.11) von den Neuronen V_x^+, V_x^-, V_y^+, V_y^-, Ω^+ und Ω^- repräsentieren die globalen Variablen V_x, V_y und $\vec{\Omega}$.

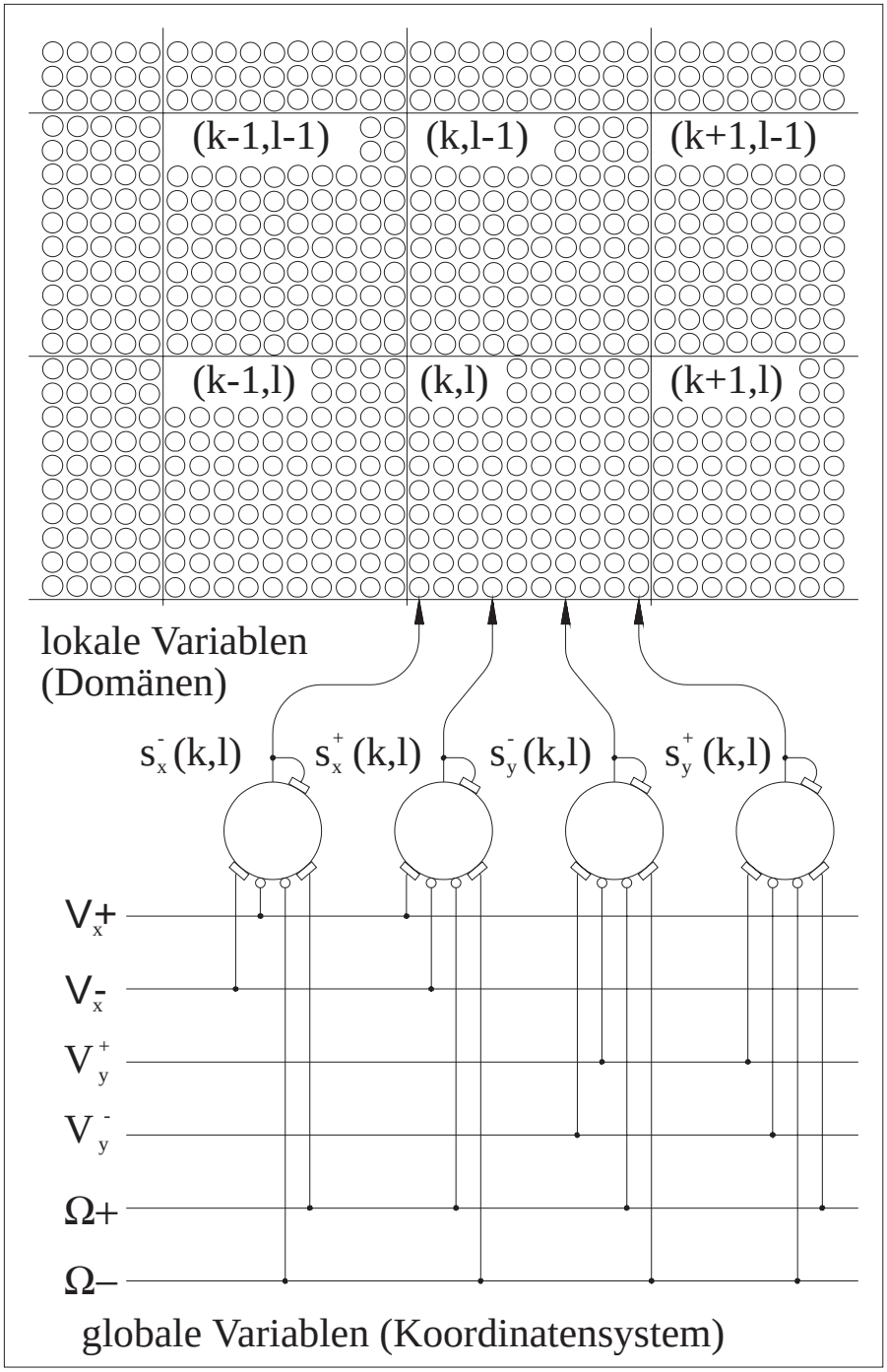


Fig. 6.11: Die Shiftneuronen berechnen aus der globalen Bewegung (**V**, $\vec{\Omega}$) des Koordinatensystems die lokalen Shiftimpulse für jede Domäne (k,l).

Nun kann das komplette neuronale Gitter, das in einzelne unabhängige Domänen (k, l) aufgeteilt ist, beschrieben werden. Innerhalb einer Domäne kann die Aktivität nur durch den gemeinsamen Shiftimpuls verschoben werden. Dieser Mechanismus war bereits Gegenstand einer genauen Betrachtung. Somit muß nur der Effekt zwischen zwei benachbarten Domänen (k, l) und (k+1, l) beschrieben werden. Falls das rechte Neuron in der Domäne (k, l) nicht aktiv ist, wird das linke Neuron in der Domäne (k+1, l) gehemmt, und es kann daher nicht aktiv werden. Falls das rechte Neuron in (k, l) gerade durch den Shiftimpuls s(k, l) aktiv wird, kann das benachbarte Neuron in (k+1, l) durch den folgenden Shiftimpuls s (k+1,l) aktiviert werden. Das einzige neue Problem ist, das Zeitintervall zwischen den Shiftimpulsen s (k, l) und s (k+1, l) zu schätzen. Da beide Shiftimpulse unabhängig sind, beträgt das maximale Zeitintervall eine volle Periode von s (k+1, l). Im Mittel wird das Intervall eine halbe Periode s (k+1, l) lang sein. Aus diesem Grund verdoppelt sich die Geschwindigkeit an der Grenze zwischen zwei Domänen. Da dies nur für die Neuronen am Domänenrand gilt, kann der Fehler vernachlässigt werden.

Mit dem beschriebenen Modell können alle Bewegungen in einem zweidimensionalen System simuliert werden. In gleicher Form läßt sich der dreidimensionale Fall diskutieren. Die translatorische Bewegung entlang der z-Achse wurde bereits erwähnt, und die Rotation um die z-Achse entspricht der Rotation im zweidimensionalen Raum. Zu betrachten sind nur noch die zwei zusätzlichen Rotationen. Sie lassen sich in ihre x- und z-Komponenten für die Drehung um die y-Achse und ihre y- und z-Komponenten für die Drehung um die x-Achse zerlegen. Die entsprechenden Spikeraten sind ebenfalls ortsabhängig. Somit wird das dreidimensionale neuronale Gitter in kleine Domänen (k, l, m) mit 8 x 8 x 8 Neuronen aufgeteilt. Zusätzlich zu den vier Shiftneuronen für die x- und y-Richtung erhält jede Domäne zwei weitere Shiftneuronen s (k, l, m) und s (k, l, m), die die Wolke nach oben (+) oder nach unten (-) bewegen. Die globalen Bewegungsvariablen sind die drei translatorischen und drei rotatorischen Bewegungskomponenten.

Für das komplette neuronale Gitter mit den unabhängigen Domänen (k, l, m) ist einerseits die Verschiebung der Aktivität innerhalb einer Domäne und anderseits die Verschiebung der Aktivität zwischen direkt benachbarten Domänen zu untersuchen. Beide Mechanismen wurden bereits diskutiert. Es kommen somit keine zusätzlichen Untersuchungen hinzu.

7. Eindimensionale Raumrepräsentationsnetzwerk

Im ersten Schritt wurde das eindimensionale RRN implementiert. Hierdurch wurde gezeigt, daß mit der vorgestellten Theorie im eindimensionalen Raum Bewegungen einer Aktivitätswolke proportional zur Geschwindigkeit des Koordinatensystems möglich sind. In dieser Version wurden die in Kapitel 3.4 vorgestellten pulscodierten Neuronen benutzt.

7.1 Struktur des eindimensionalen RRNs

Es wurde ein neuronales Netz mit 280 Gitterneuronen mit der beschriebenen Verbindungsstruktur für eine Verschiebung in (x^+)-Richtung aufgebaut. Außerdem wurde ein Shiftneuron für die (x^+)-Richtung implementiert, das mit allen Gitterneuronen verbunden ist und somit für die gesamte Verschiebung zuständig ist. Alle Gitterneuronen erhalten zum gleichen Zeitpunkt einen Shiftimpuls. Es wurde die Festlegung getroffen, daß sich das System vorerst mit einer konstanten Geschwindigkeit bewegt.

Zwecks übersichtlicher Gestaltung des Systems wurde eine Beschränkung auf die Verschiebung in (x^+)-Richtung vorgenommen. Dieses Netzwerk simuliert Bewegung von einem Startpunkt zu einem Endpunkt. Damit aber auch kontinuierliche Bewegungen durchgeführt und getestet werden konnten, wurde das letzte Gitterneuron wieder mit dem ersten Gitterneuron verbunden. Es ergibt sich ein Ring, auf dem die Aktivität fortlaufend verschoben werden kann.

In Fig. 7.1 ist die dazugehörige Grafikoberfläche zu sehen. Jedes Gitterneuron wird durch ein Quadrat dargestellt. Die Quadrate sind in einem Rechteck angeordnet, das den Ring der Gitterneuronen beschreibt. Aktive Neuronen (linke obere Ecke) werden durch graue und schwarze Rechtecke hervorgehoben. Da in dieser Version noch keine Schnittstelle zu einem sensorischen Input besteht, werden die Neuronen in der Initialisierungsphase vom Benutzer mit der Maus aktiviert bzw. deaktiviert. Auf die explizite Darstellung des Shiftneurons wird verzichtet.

Das Shiftneuron erhält über die drei Buttons „STEP“, „-->“ und „RUN“ Bewegungsinput unterschiedlicher Dauer. Nach der Betätigung von "STEP" erzeugt das Shiftneuron einen Spike und verschiebt somit das Muster um eine Position. Durch das Drücken von "-->" werden mehrere Spikes und von "RUN" eine längere Spikefolge von Shiftneuron versendet.

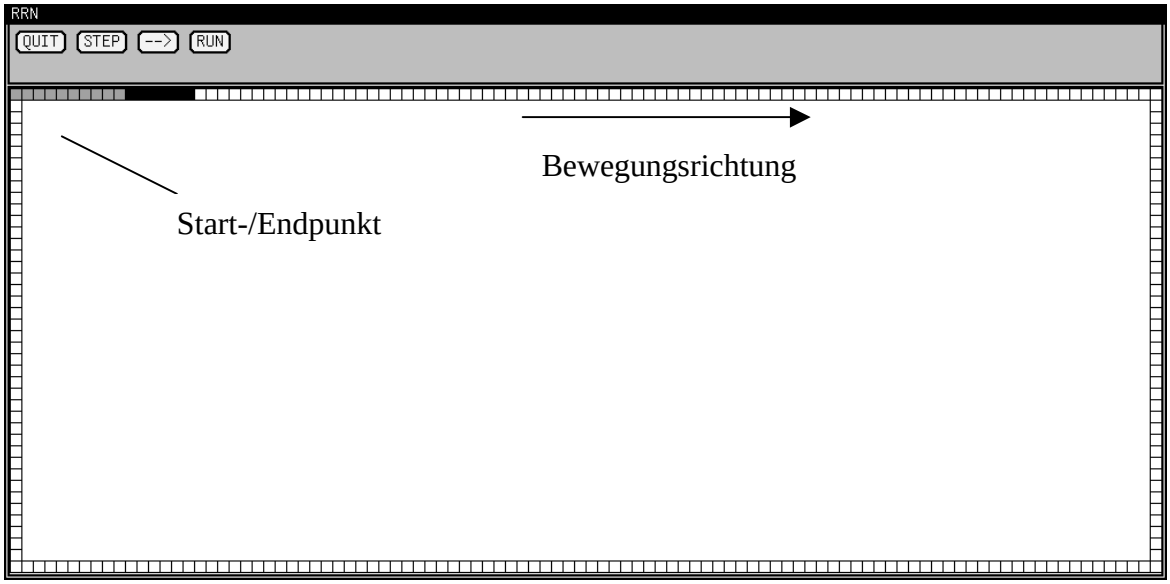


Fig. 7.1: Grafikoberfläche des eindimensionalen RRNs.

7.2 Parameter der Gitterneuronen

Bei der Simulation des Netzwerkes wurde angenommen, daß die Neuronen ein Ruhepotential von 0 mV anstatt von –80 mV haben. Alle anderen Parameter der Neuronen verschieben sich entsprechend [vgl. Kapitel 3]. In einem Simulationsschritt (Zeitschritt) wird jeweils eine Millisekunde (ms) betrachtet.

Die aktiven Gitterneuronen haben ein Membranpotential, das zwischen 45 mV und 60 mV liegt und durch einen Maximalwert von 60 mV begrenzt ist. In jedem Zeitschritt verringert sich das Membranpotential um 1,4 %. Damit ein aktives Neuron auch im ruhenden System aktiv bleibt, wirkt es mit einem Gewichtsfaktor von 4 mV/Spike auf sich selbst zurück. Das Membranpotential der passiven Neuronen liegt zwischen 30 mV und 37 mV. Hierdurch können sie mit einem Impuls erregt werden.

Die statische Schwelle T_S eines Neurons liegt bei 40 mV, und die dynamische Schwelle T_D hat im Anfangszustand 0 mV. Feuert das Neuron, wird T_D auf 80 mV erhöht und sinkt in jedem Zeitschritt um 40 %. Somit fällt der dynamische Schwellwert wieder gegen 0 mV. Ein aktives Gitterneuron feuert ca. alle 10 ms. Dies reicht aus, um ein aktives Neuron ohne zusätzlichen Input im ruhenden System aktiv zu halten.

Bewegt sich nun das System und erzeugt das Shiftneuron einen Impuls, wirkt die erregende durchgeschaltete Synapse des Shiftneurons mit 8 mV/Spike auf alle verbundenen Gitterneuronen. Mit diesem Impuls kann ein deaktives Gitterneuron aktiviert werden. Gitterneuronen, die bereits aktiv waren, erhöhen nur kurzzeitig ihr Membranpotential.

Um das Membranpotential eines aktiven Gitterneurons, das deaktiviert werden soll, unter die Schwelle von 40 mV zu senken, wird ein hemmender Impuls von 25 mV/Spike verwendet. Dieser Impuls senkt das Membranpotential mit einem Schritt unter die Schwelle.

Es hat sich gezeigt, daß mit den angegebenen Parametern eine Aktivitätswolke im eindimensionalen Fall in ihrer Länge erhalten bleibt. Dies gilt für den ruhenden Zustand genauso wie für eine fortlaufende Bewegung in die (x^+)-Richtung.

Da die (x^-)-Richtung durch ein weiteres Shiftneuron mit den spiegelbildlichen Verbindungen unabhängig von der (x^+)-Bewegung gesteuert wird, beeinflussen sich die Bewegungen nicht, und es ist nicht notwendig, die entsprechende Richtung für den eindimensionalen Fall zu implementieren.

Somit ist gezeigt, daß im Eindimensionalen die Position und Länge eines Objekts in Bezug auf die Kameraposition durch eine Aktivitätswolke dargestellt werden kann. Bei einer Bewegung des Objekts verschiebt sich die Wolke geschwindigkeitsabhängig in die entgegengesetzte Richtung, wobei die Struktur der Wolke stabil bleibt.

7.3 Größeninvarianz der Aktivitätswolke

Bei der Betrachtung der Repräsentation von Objekten in Gehirnen stellt man fest, daß gleiche Objekte in Abhängigkeit von der Entfernung unterschiedlich groß repräsentiert sind. Ein nahes Objekt wird viel größer dargestellt als das gleiche Objekt in einiger Entfernung. Es wurde ein Mechanismus gefunden, mit dem dieses Verhalten in das beschriebene neuronale Netz zu integrieren ist.

Die Größe der Repräsentation wird durch die Anzahl der aktiven Neuronen angegeben. Ein nahes Objekt muß durch mehr aktive Neuronen dargestellt werden als das gleiche Objekt in weiterer Entfernung von der Kameraposition, dem Ursprung des Koordinatensystems. Dies bedeutet, daß sich die Aktivitätswolke vergrößert, wenn sich die Kamera auf das Objekt zu bewegt, während sie sich beim Entfernen vom Objekt verkleinert.

Die Gitterneuronen werden in kleine Gruppen zusammenhängender Neuronen zusammengefaßt, die im folgenden mit *Domänen* bezeichnet werden. Jede Domäne wird durch ein eigenständiges Shiftneuron angesteuert. Alle Shiftneuronen erhalten ihren Input von den globalen Geschwindigkeitsvariablen. Um die Vergrößerung bzw. Verkleinerung der Aktivitätswolke zu beschreiben, wird davon ausgegangen, daß sich das System mit einer konstanten Geschwindigkeit bewegt.

Alle Shiftneuronen erhalten somit den gleichen konstanten Input und erzeugen hieraus ihre Spikes. Die Frequenz, mit der ein Shiftneuron feuert, ist abhängig vom Abstand r der entsprechenden Domäne vom Ursprung des Koordinatensystems. Mit zunehmendem r wird die Frequenz kleiner. Allein durch diese unterschiedlichen Shiftfrequenzen läßt sich der gewünschte Effekt realisieren.

In Fig. 7.2 ist das Prinzip der Verkleinerung der Aktivitätswolke zu sehen. In diesem Beispiel besteht jede Domäne aus 4 Neuronen (Kreisen), die in r-Richtung hintereinander angeordnet sind. Das Beispiel beschreibt die eindimensionale Bewegung der Aktivitätswolke in (r^+)-Richtung. Die Neuronen der einzelnen Domänen sind zu den Zeitpunkten aufgetragen, zu denen sie durch ihr Shiftneuron einen Impuls erhalten. Die Neuronen der Domäne 1 empfangen z.B. im Zeitpunkt t_0 und dann wieder im Zeitpunkt t_8 einen Impuls. Die Shiftneuronen selbst sind nicht dargestellt. Es ist zu sehen, daß die Neuronen der Domäne 5 doppelt so oft auftreten, wie die Neuronen der Domäne 1. Dies bedeutet, daß das Shiftneuron der Domäne 5 mit der doppelten Frequenz des Shiftneurons der Domäne 1 feuert. Die Verbindung zwischen zwei Neuronen gibt an, von welchem Nachbarneuron das Neuron, das im aktuellen Zeitschritt betrachtet wird, die Aktivität übernimmt.

Zum Zeitpunkt t_0 sind jeweils drei Gitterneuronen der beiden Domänen 4 und 5 aktiv. Nun bewegt sich das System in (r^-)-Richtung. Dies bedeutet, daß sich die Aktivität in (r^+)-Richtung verschieben muß. Als erstes ist das Shiftneuron der Domäne 5 zum Zeitpunkt t_4 aktiv. Die Aktivität in dieser Domäne wird nun jeweils auf das obere Nachbarneuron verschoben. Die Aktivität von Neuron 1 (schwarz) wird auf kein Nachbarneuron verschoben, da das Shiftneuron des Nachbarneurons 4 in Domäne 4 zur Zeit nicht aktiv ist. Die Aktivität geht "verloren". So sind anschließend nur noch das erste und zweite Neuron aktiv. Erst im nächsten Zeitschritt t_5 verschiebt sich die Aktivität in Domäne 4, so daß nun alle vier Neuronen in dieser Domäne aktiv sind. Entsprechend verschiebt sich die Aktivitätswolke in den nächsten Zeitschritten.

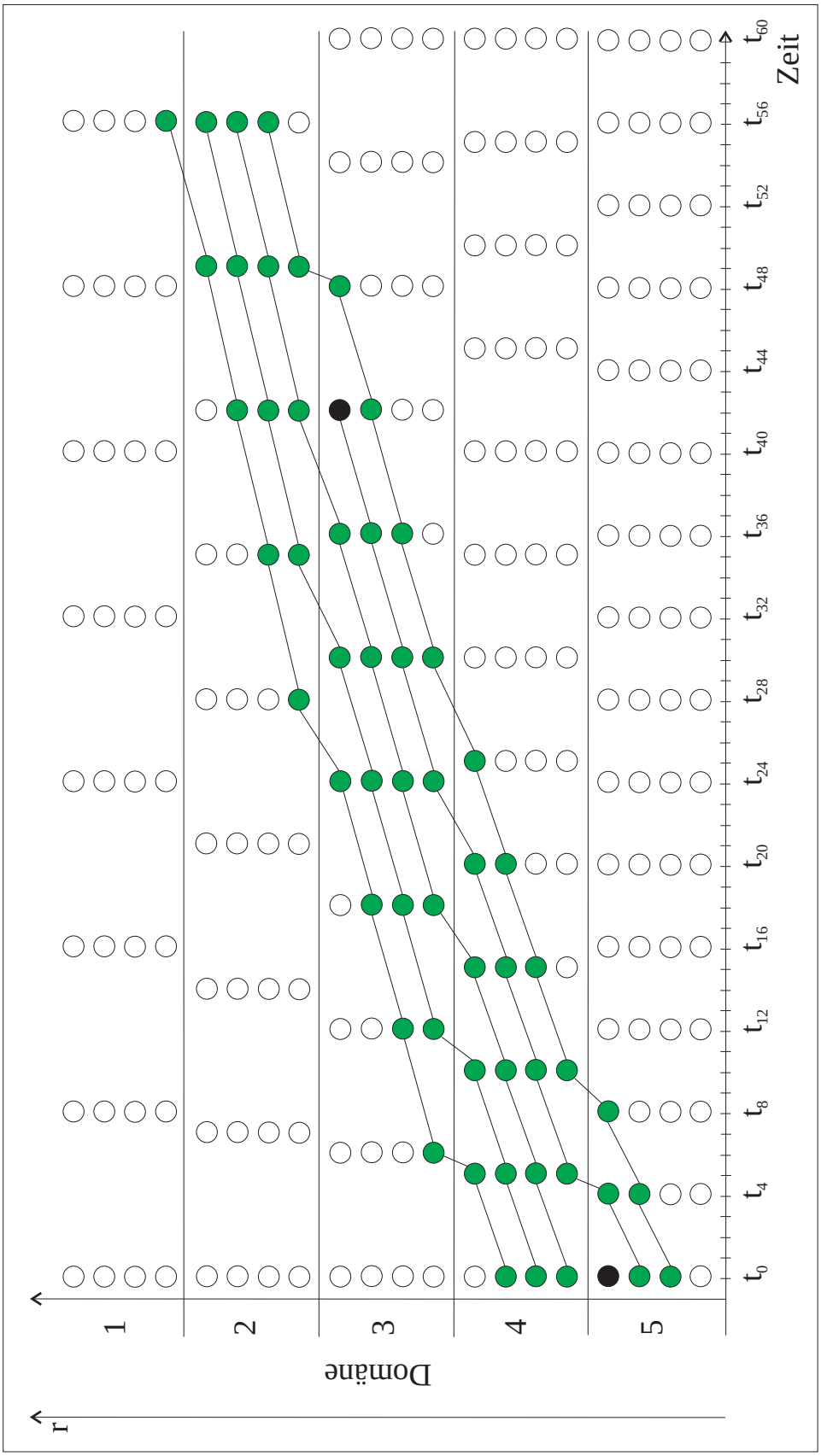


Fig. 7.2: Die Verschiebung der Aktivitätswolke in (r^+)-Richtung.

Um die Länge der Aktivitätswolke zu diskutieren, müssen diejenigen Zeitpunkte betrachtet werden, zu denen die Shiftneuronen der Domänen, in denen Neuronen aktiv sind, gleichzeitig feuern. Dies ist bei t_0 und t_{56} der Fall. Die Aktivitätswolke besteht in t_0 aus sechs aktiven Neuronen und hat sich bis zum Zeitpunkt t_{56} auf vier aktive Neuronen reduziert. Die Reduzierung resultiert aus der Situation in den Zeitschritten t_4 und t_{48}, wo jeweils die Aktivität von Neuron 1 noch nicht auf das Nachbarneuron 4 der Nachbardomäne weitergegeben wurde, aber das entsprechende Shiftneuron von Neuron 1 schon wieder feuert. Dadurch verringert sich die Länge jeweils um ein aktives Neuron.

Allgemein kann festgestellt werden, daß die Aktivitätswolke kürzer wird, wenn das Shiftneuron mit der höheren Frequenz in der Zeit zwischen zwei Spikes des anderen Neurons zwei mal feuert. Hierbei können die Zeitpunkte der ersten Feuerns übereinstimmen.

Die entsprechende Betrachtung zur Verlängerung der Aktivitätswolke ist in Fig. 7.3 dargestellt. Zu Beginn ist die Wolke vier aktive Neuronen lang. Bis zum Zeitschritt t_{60} hat sie sich auf sieben Neuronen vergrößert. Zum ersten Mal verlängert sich die Wolke im Zeitschritt t_{42} um ein Neuron. Die Aktivität von Neuron 4 in Domäne 2 überträgt sich zweimal auf das Neuron 1 der Domäne 3 (Zeitschritte t_{35} und t_{42}). Die beiden anderen zusätzlichen Aktivierungen resultieren aus der doppelten Übertragung der Aktivitäten von Neuron 4 aus Domäne 3 zum Zeitpunkt t_{55} und t_{60} auf Neuron 1 der Domäne 4 und von Neuron 4 aus Domäne 4 im Zeitschritt t_{56} und t_{60} auf Neuron 1 der Domäne 5.

Auch hier kann allgemein gesagt werden, daß sich die Aktivitätswolke verlängert, wenn das Shiftneuron mit der höheren Frequenz in der Zeit zwischen zwei Spikes des anderen Neurons zwei mal feuert. Hierbei können die Zeitpunkte des zweiten Feuerns der beiden Neurone übereinstimmen.

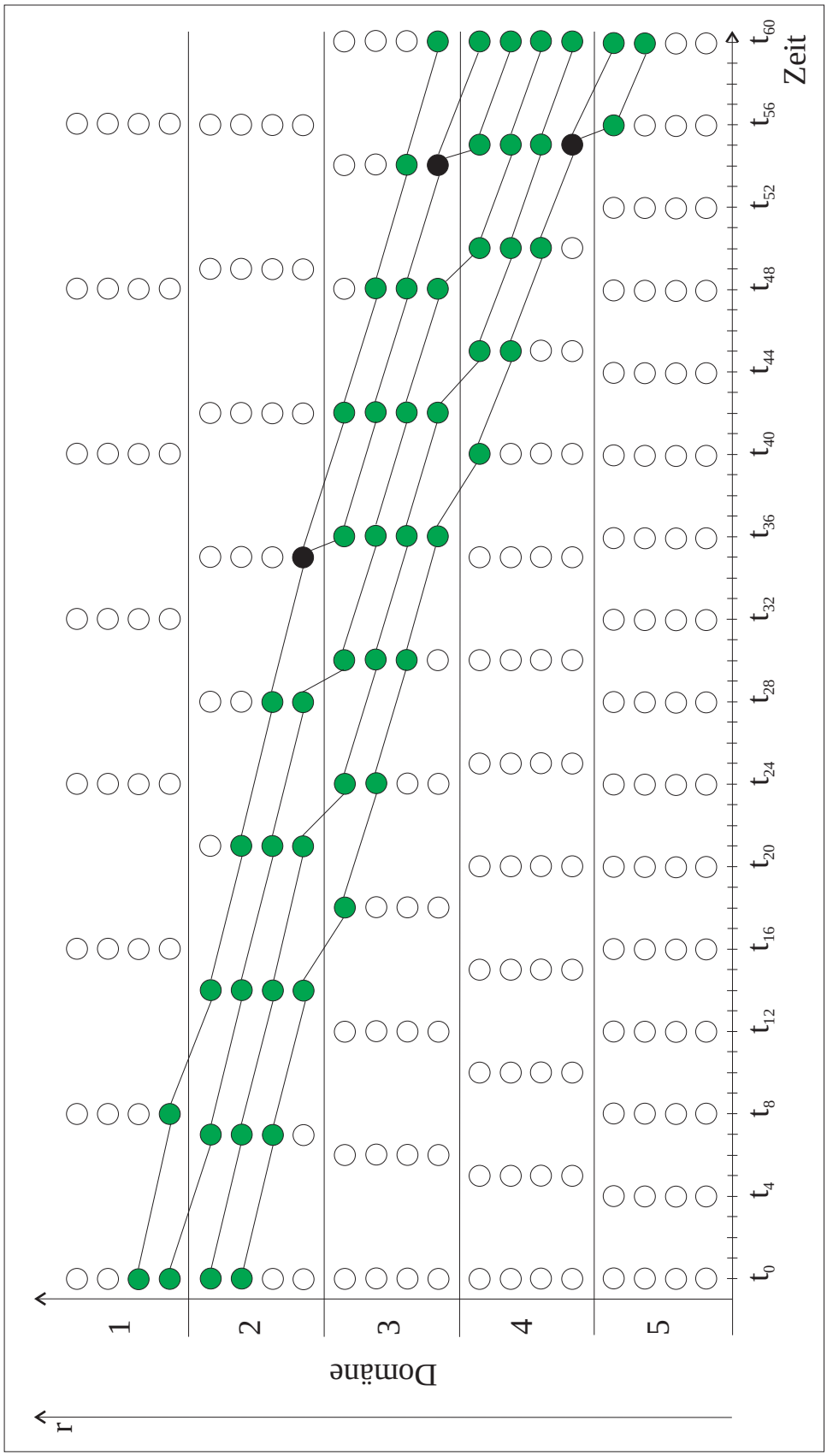


Fig. 7.3: Die Verschiebung der Aktivitätswolke in ($\bar{r}$)-Richtung.

7.4 Ergebnisse

Das eindimensionale System wurde entsprechend modifiziert, um die Größeninvarianz zu testen. Hierzu wurde das Netzwerk in 28 Domänen mit je 10 Neuronen aufgeteilt. Die Geschwindigkeit des sich bewegenden Koordinatensystems bleibt weiterhin konstant.

Das Shiftneuron, das mit der ersten Domäne verbunden ist, feuert mit einer konstanten Frequenz f_1. Die Frequenz der anderen Shiftneuronen erniedrigt sich mit zunehmendem r jeweils um p%. Sie berechnet sich folgendermaßen:

$$f_i = f_{i-1} - f_{i-1} \cdot p\%, \; 1 \leq i \leq n \qquad (7.1)$$

Durch den Wert p wird die Geschwindigkeit der Wolkenveränderung bestimmt. Je größer dieser Wert ist, desto kleiner ist die Frequenz der einzelnen Domänen und desto schneller verkürzt bzw. verlängert sich die Wolke. Für unsere Simulationen wurde eine Steigerung der Frequenz von 5% gewählt. Dies bedeutet, daß die Domäne 14 ca. mit der halben Frequenz der Domäne 1 feuert. Die Aktivitätswolke verkleinert sich zwischen der ersten und 14 ten Domäne um die Hälfte. In der Grafikoberfläche befinden sich die Neuronen der 14 en Domäne unten rechts.

Um wieder eine kontinuierliche Bewegung zu erreichen, wird die Frequenz nur bis zur 14 ten Domäne erniedrigt. Von da an erhöht sie sich folgendermaßen:

$$\text{Frequenz (Domäne[14+i])} = \text{Frequenz (Domäne[14-i])}, \; 0 \leq i \leq 13$$

Somit wird die Aktivitätswolke bis zur 15 ten Domäne verkleinert und dann bei der Verschiebung durch die Domänen 16-28 wieder auf ihre Ausgangsgröße verlängert. Hiermit kann nun die Aktivitätswolke kontinuierlich bewegt werden.

Die Simulation zeigt, daß die Aktivitätswolke mit diesem Mechanismus verkleinert und wieder vergrößert werden kann. Die Länge der Aktivitätswolke aus Fig. 7.3 reduziert sich bei einer Verschiebung bis zur 15 ten Domäne von 16 auf 8 Neuronen (Fig. 7.4). Vergleicht man die Größe der Wolke während einer kontinuierlichen Bewegung an einer festen Position k, so gilt, daß durch die Unabhängigkeit der Frequenzen der einzelnen Shiftneuronen zwar nicht immer eine gleichmäßige Veränderung garantiert werden kann, da die Anzahl der aktiven Neuronen schwankt, daß im. Durchschnitt jedoch an der Position k die gleiche Anzahl von Neuronen aktiv ist.

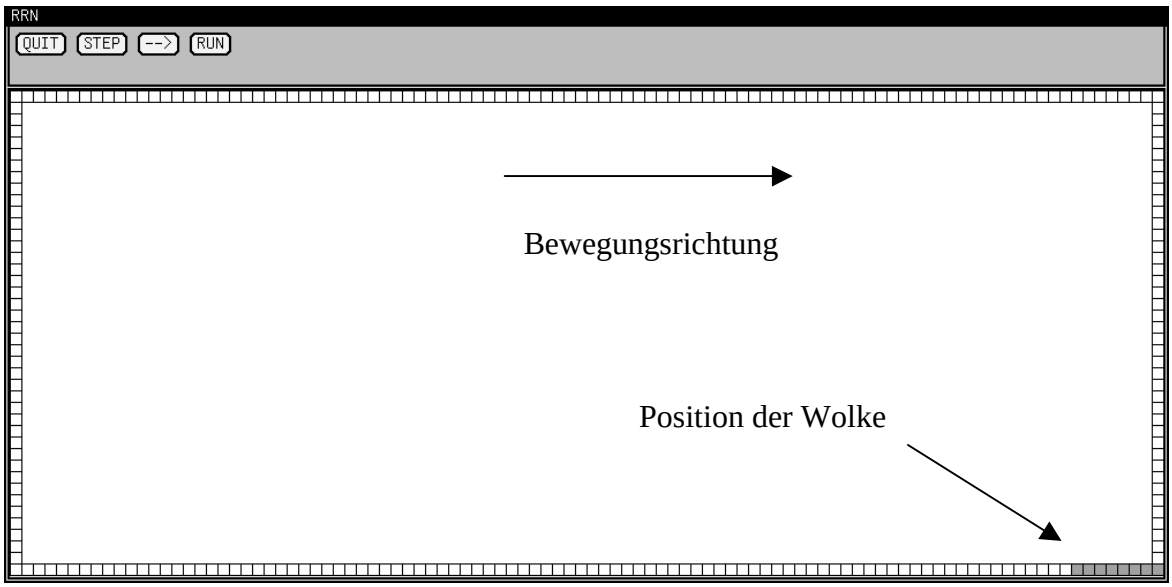


Fig. 7.4: Die Länge der Aktivitätswolke (grau) aus Fig. 7.1 (16 Neuronen) hat sich auf die Hälfte reduziert.

Der vorgestellte Mechanismus wurde in ein zweidimensionales Polarkoordinatensystem übertragen. Die entsprechende Vorgehensweise und Ergebnisse sind im Kapitel 8.9 zu finden.

8. Zweidimensionale Raumrepräsentationsnetzwerk

Im Anschluß hieran wurde das RRN in einem zweidimensionalen kartesischen Koordinatensystem implementiert. Es wurde eine Fläche mit 100 x 100 Domänen aufgebaut. Jede Domäne besteht aus 8^2 Neuronen. Da die Simulation von pulscodierten Neuronen sehr zeitaufwendig ist und sich für eine so große Anzahl von Neuronen nicht in einer annehmbaren Zeit durchführen läßt, wurde das Neuronenmodell vereinfacht.

8.1 Simulation der Gitterneuronen

Für den Verschiebemechanismus selbst ist es nur von Bedeutung, ob ein Gitterneuron aktiv oder passiv ist und deshalb werden die Gitterneuronen als binäre Neuronen betrachtet. Somit können ein Neuron durch ein Bit und eine Domäne durch zwei Integerzahlen kodiert werden, und die aufwendige Berechnung der Aktivitätszustände für die Neuronen fällt weg. Die Bewegung der Aktivitätswolke läßt sich somit auf logische Verschiebungen und Verknüpfungen der einzelnen Bits reduzieren. Es sind also nur einfache elementare Operationen notwendig. Außerdem wurde diese Kodierung gewählt, da sie den geringsten Speicherplatz benötigt.

Um die 8^2 Neuronen einer Domäne durch zwei Integerzahlen zu kodieren, werden die Zahlen nach jedem Byte gefaltet (Fig. 8.1). Jedes Byte beschreibt somit eine Zeile der Domäne. Die beiden Zahlen werden im folgenden mit *oberer* und *unterer Domänenvariablen* bezeichnet.

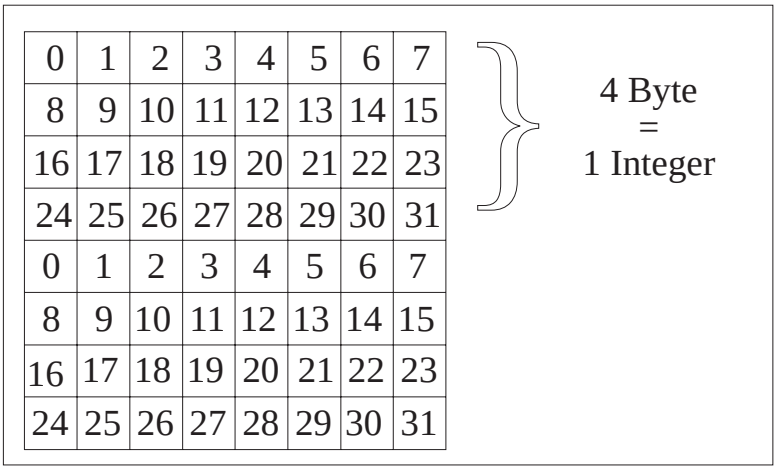


Fig. 8.1: Die Kodierung einer Domäne mit 8^2 Neuronen durch zwei Integerzahlen.

Nun soll zuerst beschrieben werden, wie die Aktivität einer Domäne um eine Position nach links verschoben wird. Da für beide Domänenvariablen die gleichen Anweisungen durch-

zuführen sind, beschränkt sich diese Arbeit auf die Beschreibung der Aktivitätsverschiebung, die durch eine Zahl kodiert ist (Fig. 8.2).

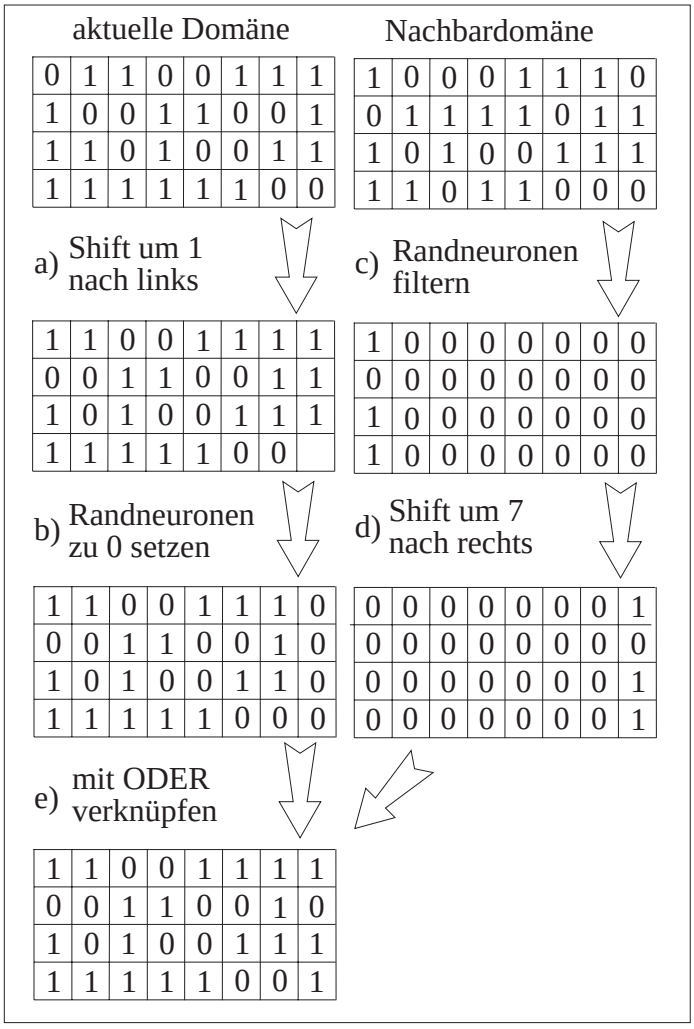


Fig. 8.2: Beispiel für eine Bewegung der Aktivitätswolke nach links.

Zuerst werden die Bits um eine Position nach links verschoben (Fig. 8.2a). Hiermit ist die gesamte Aktivitätsverschiebung innerhalb der Domäne bereits abgeschlossen. Nun muß nur noch die Aktivität der Randneuronen der Nachbardomäne (Bits 0, 8, 16 und 24) auf die angrenzenden Randneuronen der aktuellen Domäne (Bits 7, 15, 23 und 31) übertragen werden. Zuerst werden die Randneuronen in der betrachteten Domäne auf Null gesetzt (Fig. 8.2b). Anschließend werden die Zustände der Nachbarrandneuronen in eine zusätzliche Variable kopiert (Fig. 8.2c). Diese wird nun um sieben Bits nach rechts verschoben, um die Position zu erreichen, in der die Information in der aktuellen Domäne stehen muß (Fig. 8.2d). Zum Schluß werden die beiden Zahlen bitweise mit Oder verknüpft und in die aktuelle Domänenvariable geschrieben (Fig. 8.2e).

Mit diesen fünf Anweisungen wird die Aktivität, die durch eine Domänenvariable kodiert ist, um eine Position verrückt. Um die gesamte Aktivität in einer Domäne zu verschieben, müssen die gleichen Anweisungen für die zweite Variable ebenfalls durchgeführt werden. Somit wird die Aktivität durch insgesamt zehn Anweisungen verschoben.

Auch für die anderen Richtungen lassen sich die Verschiebungen durch entsprechende Anweisungen realisieren. Es ändert sich jeweils nur die Richtung bzw. die Länge der Shifts der Variablen. Außerdem ändert sich jeweils die Lage und somit auch die Verschiebelänge und Verschieberichtung der Randneuronen. In der Tabelle 8.1 sind die entsprechenden Werte für alle Bewegungsrichtungen der Aktivitätswolke aufgelistet.

	Bewegung nach			
	links	rechts	oben	unten
Länge und Richtung des internen Shifts	+1	-1	+8	-8
Bits der Randneuronen	7 und 15 23 und 31	0 und 8 16 und 24	0 bis 7	24 bis 31
Bits der Nachbarrandneuronen	0 und 8 16 und 24	7 und 15 23 und 31	24 bis 31	0 bis 7
Länge und Richtung des Shifts der Randneuronen der Nachbardomäne	-7	+7	-24	+24

Tab. 8.1: Zuordnung der Shiftlänge, Shiftrichtung und Bitnummer zur Bewegungsrichtung der Aktivitätswolke.

Bei der Bewegung nach unten und oben ist noch eine Besonderheit zu beachten. Bewegt sich die Aktivitätswolke z. B. nach oben, grenzt der obere Teil der Domäne, der durch die obere Domänenvariable repräsentiert wird, nicht an die Nachbardomäne, aus der die Aktivität übernommen wird, sondern sie grenzt an die untere Domänenvariable der gleichen Domäne. Die Aktivitätswolke übernimmt deren Aktivität. Die untere Domänenvariable hingegen übernimmt die Aktivität von den acht Randneuronen der Nachbardomäne. Trotz dieses Unterschiedes sind die gleichen Verschiebungen für beide Zahlen durchzuführen. Bei der entgegengesetzten Bewegung nach unten ist es genau umgekehrt.

Da die Verschiebung der Aktivität einer Domäne sequentiell abgearbeitet wird, muß bei allen Bewegungen beachtet werden, in welche Richtung eine Verschiebung erfolgt. Feuern zwei

Shiftneuronen benachbarter Domänen zur gleichen Zeit, dann müssen die Domänen immer in der inversen Richtung bearbeitet werden: bei einer Bewegung der Wolke nach links müssen die Domänen also von links nach rechts betrachtet werden. Hierdurch wird verhindert, daß die Aktivität der Randneuronen überschrieben wird, bevor sie auf die Nachbarneuronen der angrenzenden Domäne verschoben wurde.

8.2 Simulation der Shiftneuronen

Die Vereinfachung der Simulation der Shiftneuronen soll nun analysiert werden. Die Shiftneuronen erzeugen Spikeraten, die proportional zur Geschwindigkeit des Koordinatensystems sind. Sie berechnen aus der globalen Bewegung ($\mathbf{V}$, $\vec{\Omega}$) des Koordinatensystems die lokalen Shiftimpulse für jede Domäne (k, l) (vgl. Kapitel 6.2). In Abhängigkeit von der Geschwindigkeit läßt sich für jedes Shiftneuron die Feuerfrequenz f($\mathbf{V}$, $\vec{\Omega}$, k, l) für jede Richtung bestimmen. Für die Frequenz eines Shiftneurons s gilt:

$$f_s(\mathbf{V}, \vec{\Omega}, k, l) \approx v(\mathbf{V}, \vec{\Omega}, k, l)$$

Da sich das zugrundeliegende Koordinatensystem mit unterschiedlichen Geschwindigkeit bewegt, müssen bei jeder Geschwindigkeitsveränderung aufwendige Berechnungen durchgeführt werden, um die aktuelle Feuerfrequenz aller Shiftneuronen zu berechnen. Dies ist sehr zeitaufwendig, und eine Bewegung im RRN kann erst simuliert werden, wenn die Berechnungen jeweils abgeschlossen sind. Hieraus folgt, daß die Bewegung des Koordinatensystems durch die Berechnungszeit im RRN beeinflußt und womöglich verlangsamt wird.

Eine Speicherung aller möglichen Feuerfrequenzen für jedes Shiftneuron in Tabellen ist aufgrund der großen Anzahl möglichen Geschwindigkeiten nicht realisierbar. Durch die Begrenzung auf bestimmte Geschwindigkeiten könnten alle dann möglichen Feuerfrequenzen in Tabellen abgelegt werden. Doch würde hierdurch die Funktion der Experimentierumgebung sehr stark eingeschränkt.

Um dies zu umgehen, werden die Bewegungen des körperbezogenen Koordinatensystemes und der Aktivitätswolke entkoppelt. Die Synchronisation geschieht über die zurückgelegte Strecke. Nach jeder *Streckeneinheit* Δs_k, die sich das Koordinatensystem bewegt, wird ein Impuls an die Shiftneuronen im RRN gesendet, der dann die Aktivitätswolke um die gleiche Strecke Δs_{RRN} in die entgegengesetzte Richtung im RRN verschiebt.

Bei dieser Betrachtungsweise kommt es nur noch darauf an, daß beide Systeme die gleiche Strecke zurücklegen. Die Geschwindigkeit mit denen dies geschieht, ist irrelevant. Hieraus ergibt sich, daß sich die Aktivitätswolke im RRN mit einer konstanten Geschwindigkeit bewegt, und daß folglich jedes Shiftneuron ebenfalls eine konstante Feuerfrequenz besitzt. Diese kann vor dem Start der Simulation berechnet und dann in einer Tabelle abgelegt werden. Während der Umsetzung hat sich gezeigt, daß es günstiger ist, ihren Kehrwert in der Tabelle zu speichern. Dieser wird im folgenden mit *Feuerabstand* bezeichnet. Daneben wird für jedes Shiftneuron der letzte Zeitpunkt festgehalten, zu dem es das letzte Mal aktiv war (*Feuerzeitpunkt*). Es sind für jede Richtung unterschiedliche Tabellen notwendig, da ein Shiftneuron je nach Bewegungsrichtung unterschiedliche Frequenzen besitzt.

Zur Simulation der Shiftneuronen wird nun eine rechnerinterne *Zeitskala* benutzt. Anhand dieser Zeitskala wird die Verschiebung vorgenommen. Für jede Bewegungsrichtung wird nun diese Zeitskala aufgebaut. Gestartet wird bei der Zeit $t_0 = 0$. Bei jedem Impuls in die entsprechende Richtung muß die Simulationszeit Δt_{RRN} abgearbeitet werden. Hierzu wird das *Zeitintervall* $[t_0, t_0+t_{RRN}]$ gebildet. In jedem Zeitschritt t_j des Intervalls wird überprüft, ob ein oder mehrere Shiftneuronen zum betrachteten Zeitpunkt aktiv sind. Ist dies der Fall, dann wird die Aktivität in den entsprechenden Domänen verschoben. Außerdem wird für das aktive Neuron der Feuerabstand zum aktuellen Feuerzeitpunkt addiert und so ermittelt, zu welchem Zeitpunkt das Neuron das nächste Mal aktiv ist. Sind alle aktiven Neuronen abgearbeitet oder ist kein Neuron aktiv, wird der nächste Zeitschritt betrachtet. Ist das Ende des Zeitintervalls erreicht, wird

$$t_0 = t_0 + t_{RRN}$$

gesetzt und so beim nächsten Impuls in diese Richtung das angrenzende Zeitintervall abgearbeitet. Hierdurch wird sichergestellt, daß nicht immer die selben Neuronen feuern, sondern wirklich die Frequenz der einzelnen Neuronen simuliert wird.

8.3 Aktivitätsfenster

Betrachtet man die gesamte simulierte Fläche im Verhältnis zur Größe der Objekte, muß man feststellen, daß in den meisten Flächenelementen kein Objekt vorhanden ist. Deshalb ist die Anzahl der passiven Gitterneuronen viel größer als die Anzahl der aktiven Gitterneuronen. Doch bisher wird die Verschiebung im gesamten Netzwerk vorgenommen. Es wird nicht berücksichtigt, ob die entsprechenden Gitterneuronen überhaupt aktiv sind.

Dies führt zu dem Schluß, nur noch dort eine Verschiebung vorzunehmen, wo Neuronen aktive sind. Deshalb wurde um alle Domänen, in denen sich aktive Neuronen befinden, ein rechteckiges Fenster gelegt, welches im folgenden als *Aktivitätsfenster* bezeichnet wird. Neben den Domänen mit aktiven Neuronen wird in jede Ausdehnung in der Initialisierungsphase eine zusätzliche Domänenreihe mit in das Aktivitätsfenster aufgenommen. Dies ist notwendig, da die Randneuronen einer Domäne ihre Aktivität an die angrenzenden Neuronen ihrer Nachbardomäne weitergeben müssen.

Bei einer Verschiebung muß nun nach jedem Zeitschritt zusätzlich kontrolliert werden, ob sich das Aktivitätsfenster verändern muß. Beispielhaft soll dies an einer Bewegung der Aktivitätswolke nach rechts verdeutlicht werden. Nach der Verschiebung werden zuerst die Gitterneuronen am vorderen Rand des Aktivitätsfensters überprüft (Bereich A in Fig. 8.3). Sind ein oder mehrere aktiv, dann ist das Fenster um eine Domänenbreite zu vergrößern.

Die Kontrolle, ob ein Randneuron aktiv ist, geschieht durch eine logische UND-Verknüpfung mit einer Maske, die die Aktivität der Randneuronen herausfiltert. Ist das Ergebnis größer als Null, ist mindestens ein Randneuron aktiv. Ansonsten ist kein Neuron am Rand aktiv.

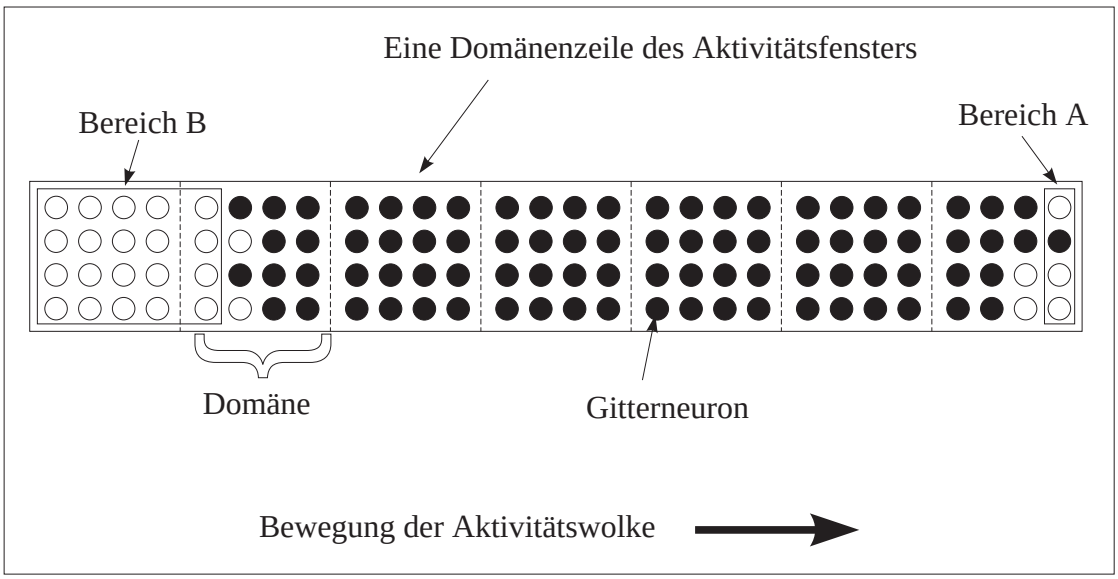


Fig. 8.3: Veränderung der Aktivitätswolke bei einer Bewegung nach rechts.

Auf der anderen Seite gilt, daß das Aktivitätsfenster verkleinert werden muß, wenn in der letzten Domänenreihe keine Neuronen und in der vorletzten Domänenreihe keine Randneuronen aktiv sind (Bereich B in Fig. 8.3). Ob in einer Domäne kein aktives Neuron ist, wird dadurch festgestellt, daß beide Integerzahlen gleich Null sind. Die Kontrolle der Randneuronen der

nächsten Domäne geschieht wiederum durch eine logische UND-Verknüpfung mit einer Maske und mit dem Test, ob das Ergebnis gleich Null ist. Dies sind alles einfache Operationen, die sehr schnell durchgeführt werden können.

Wurde festgestellt, daß das Aktivitätsfenster vergrößert werden muß, dann muß eine zusätzliche Domänenreihe in das Aktivitätsfenster aufgenommen werden. Die Shiftneuronen dieser Domänenreihen, wurden bisher nicht betrachtet. Das heißt, daß ihr Feuerzeitpunkt auf der Zeitskala nicht aktuell ist. Der Feuerzeitpunkt muß aktualisiert werden. Hierzu wird zuerst berechnet, wann die einzelnen Shiftneuronen das letzte Mal aktiv gewesen wären, wenn sie kontinuierlich mitsimuliert worden wären. Anschließend wird der nächste Feuerzeitpunkt für die einzelnen Neuronen entsprechend gesetzt.

Die Berechnung der letzten Aktivität eines Shiftneurons ist ebenfalls nicht besonders aufwendig. Für jedes Neuron ist der konstante Abstand zwischen den einzelnen Spikes und außerdem der aktuelle Zeitpunkt bekannt. Hieraus läßt sich die Anzahl der Spikes des Shiftneurons bis zu diesem Zeitpunkt berechnen und somit dann der Zeitpunkt, zu dem das Neuron zum letzten Mal aktiv gewesen wäre.

Wenn das Aktivitätsfenster verkleinert wird, ist keine weitere Anweisung durchzuführen. Die entsprechenden Domänen werden durch das Setzen der Zeiger für das Aktivitätsfenster aus dem aktuellen Bereich genommen und dann nicht mehr betrachtet.

8.4 Zeitsprungverfahren

Wird die Verschiebung durch die Zeitskala in der beschriebenen Form realisiert, ist nicht in jedem Zeitpunkt ein Shiftneuron aktiv. Es gilt sogar, daß in den meisten Zeitschritten kein Shiftneuron feuert. Zu diesen Zeitpunkten geschieht nichts. Würden nun in jedem Zeitschritt die Shiftneuronen überprüft, kostete dies sehr viel Rechenzeit. Deshalb wurde ein Verfahren gewählt, mit dem nur Zeitschritte betrachtet werden, in denen mindestens ein Shiftneuron aktiv ist.

Bei der ersten Bearbeitung der Zeitskala für eine Richtung wird zuerst der Zeitpunkt t_i bestimmt, zu dem erstmals ein Shiftneuron innerhalb des Aktivitätsfensters aktiv ist. Hierzu wird aus den Feuerzeitpunkten aller Shiftneuronen innerhalb der Aktivitätswolke der Feuerzeitpunkt bestimmt, zu dem auf der Zeitskala das erste Shiftneuron einen Impuls sendet. Es wird also das Minimum der Feuerzeitpunkte berechnet.

Wird nun das Zeitintervall für die Bewegung der Aktivitätswolke abgearbeitet, so wird sofort der Zeitpunkt t_i betrachtet. In t_i wird dann nicht nur die Aktivitätswolke verschoben, sondern das Minimum der Feuerzeitpunkte aller Shiftneuronen innerhalb der Aktivitätswolke ermittelt. So wird der nächste Zeitpunkt t_j mit dem minimalem Abstand zum aktuellen Zeitschritt t_i gefunden, in dem wieder ein oder mehrere Shiftneuronen einen Spike aussenden. Im nächsten Durchlauf der Schleife wird t_i gleich t_j gesetzt und somit ein Sprung zum nächsten Zeitpunkt durchgeführt, zu dem mindestens die Aktivität einer Domäne verschoben wird. Hier wird nach dem gleichen Verfahren wieder der Zeitschritt gesucht, in dem das nächste Shiftneuron aktiv ist. Das gesamte Zeitintervall wird in dieser Form abgearbeitet. Es ist damit sichergestellt, daß kein Zeitschritt ausgelassen wird, in dem ein Neuron feuert, aber auch keiner betrachtet wird, in dem kein Shiftneuron aktiv ist.

Ist die Zeitskala für eine Richtung abgearbeitet, wird der zuletzt betrachtende Zeitschritt t_i festgehalten. So kann bei der nächsten Bearbeitung dieser Richtung auf den gespeicherten Wert t_i aufgesetzt werden; er braucht also nicht neu berechnet zu werden.

8.5 Implementierung

In der Simulation wurde die Größe einer Domäne auf 2 cm^2 festgelegt. Somit deckt ein Neuron eine Fläche von 2,5 mm^2 ab. Dies bedeutet, daß mit dem RRN eine Fläche von 2 m x 2 m um den Startmittelpunkt der Kamera simuliert wird. In der Versuchsanordnung hängt die Kamera in fester Höhe über der Fläche und bewegt sich in dieser Ebene.

Mit den vorgestellten Techniken wurde das zweidimensionale RRN nun implementiert. Nun soll eine grobe Übersicht über den gesamten Programmablauf gegeben werden:

Während der Initialisierungsphase werden die Tabellen für die Shiftneuronen berechnet. Das RRN läuft dann in einer Endlosschleife und wartet so lange, bis ein Bewegungsimpuls ankommt (Fig. 8.4). Anhand des Bewegungsimpulses wird das abzuarbeitende Zeitintervall berechnet und dann der erste Zeitpunkt t_i bestimmt, zu dem ein Shiftneuron im Zeitintervall aktiv ist. Dann wird für den betrachteten Zeitpunkt t_i jeweils kontrolliert, welche Shiftneuronen innerhalb des Aktivitätsfensters aktiv sind, und die Aktivität der entsprechenden Gitterneuronen wird verschoben. Außerdem wird der nächste Feuerzeitpunkt der Shiftneuronen berechnet und für die Shiftneuronen das Minimum der Feuerzeitpunkte ermittelt. Nachdem das gesamte Aktivitätsfenster abgearbeitet ist, wird der nächste zu bearbeitende Zeitpunkt t_i gefunden. Zunächst wird jedoch die Größe des Aktivitätsfensters überprüft und entsprechend

modifiziert. Dies erfolgt für das gesamte Zeitintervall. Anschließend wird auf den nächsten Bewegungimpuls gewartet.

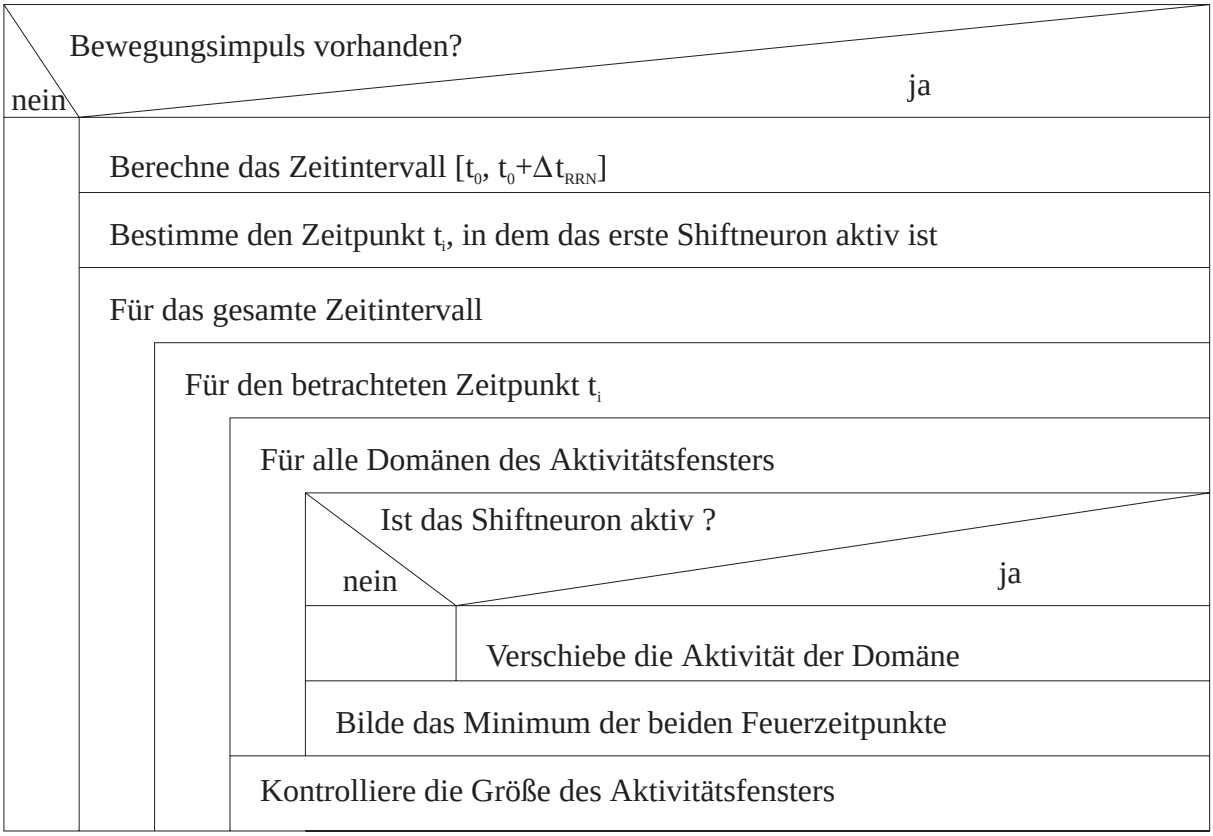


Fig. 8.4: Struktogramm des Programmablaufes

8.6 Grafikoberfläche

Zur Visualisierung der Raumzustände wurde eine Grafikoberfläche geschaffen (Fig. 8.5), in der die Fläche unter der Kamera dargestellt wird. Eine symbolische Kamera zeigt die Position an, über der die Kamera hängt. Diese Position ist immer gleich. Aktivitätswolken von neu aufgenommenen Objekten werden immer unter der Kamera in das RRN integriert. Bei einer Bewegung verschiebt sich die Aktivitätswolke innerhalb der Grafik. Da nicht die gesamte simulierte Fläche in einem Fenster gleichzeitig darzustellen ist, kann mit einem Scrollbar die sichtbare Fläche verschoben werden. Jeder Punkt der Grafik steht für ein Neuron im RRN. Um die einzelnen Domänen abzugrenzen, werden sie durch eine ein Punkt breite Linie voneinander abgegrenzt. Dies ist an der Aktivitätswolke (schwarze Seitenplatte) in der Grafik zu

sehen. Sie wird durch waagerechte und senkrechte weiße Linien unterbrochen, während es in Wirklichkeit eine zusammenhängende Wolke ist.

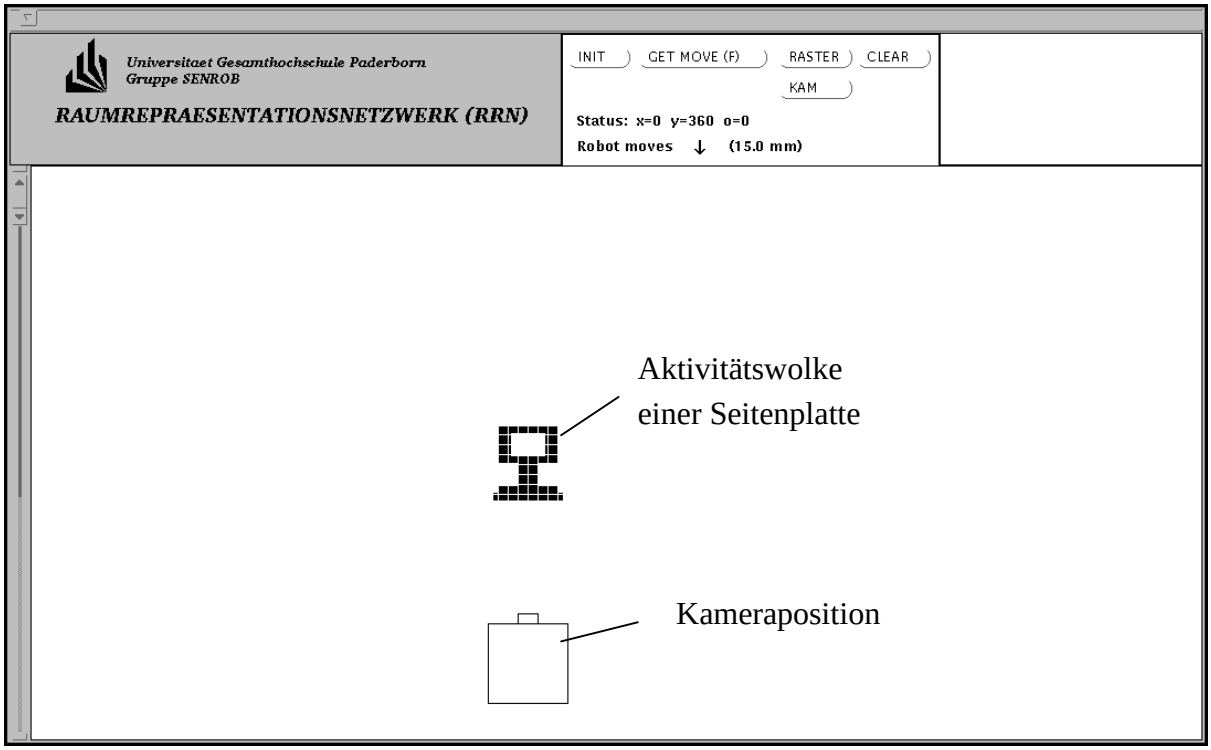


Fig. 8.5: Grafikoberfläche des zweidimensionalen RRNs.

Die Grafikoberfläche bietet außerdem die Möglichkeit, Bewegungsimpulse zu erzeugen. Mit dem Button "GET MOVE (F)" werden entsprechende Bewegungsfolgen aus einer Datei eingelesen und die Aktivitätswolke verschoben. Es ist jeweils angegeben, welche Strecke in welche Richtung zurückzulegen ist.

Als weiteres kann über die Datei eine Aktivitätswolke erzeugt werden. Hierzu muß das Zeichen "B" in der Datei stehen. Wird dieses gelesen, wird die Wolke im Rahmen der Kamera erzeugt. Form und Größe sind im Programm einstellbar. So können beliebige Bewegungsfolgen im RRN simuliert werden. In einer Statuszeile wird während des Programmablaufs angezeigt, um wieviel mm in x- und y-Richtung die bisherige Bewegung erfolgte und um wieviel Grad sich die Kamera gedreht hat. In der Zeile darunter wird die letzte Bewegung des Roboters angezeigt. Im Beispiel haben sich die Aktivitätswolke um 360 mm in y-Richtung und zuletzt der Roboter um 15 mm nach unten in y^--Richtung bewegt.

Neben der Ansteuerung des RRN aus einer Datei werden Aktivitätswolken und Bewegungsimpulse noch von einem Robotersimulationsprogramm erzeugt. Eine genaue Beschreibung dieses Programms und der Koppelung zwischen den beiden Modulen ist in Kapitel 10 zu finden.

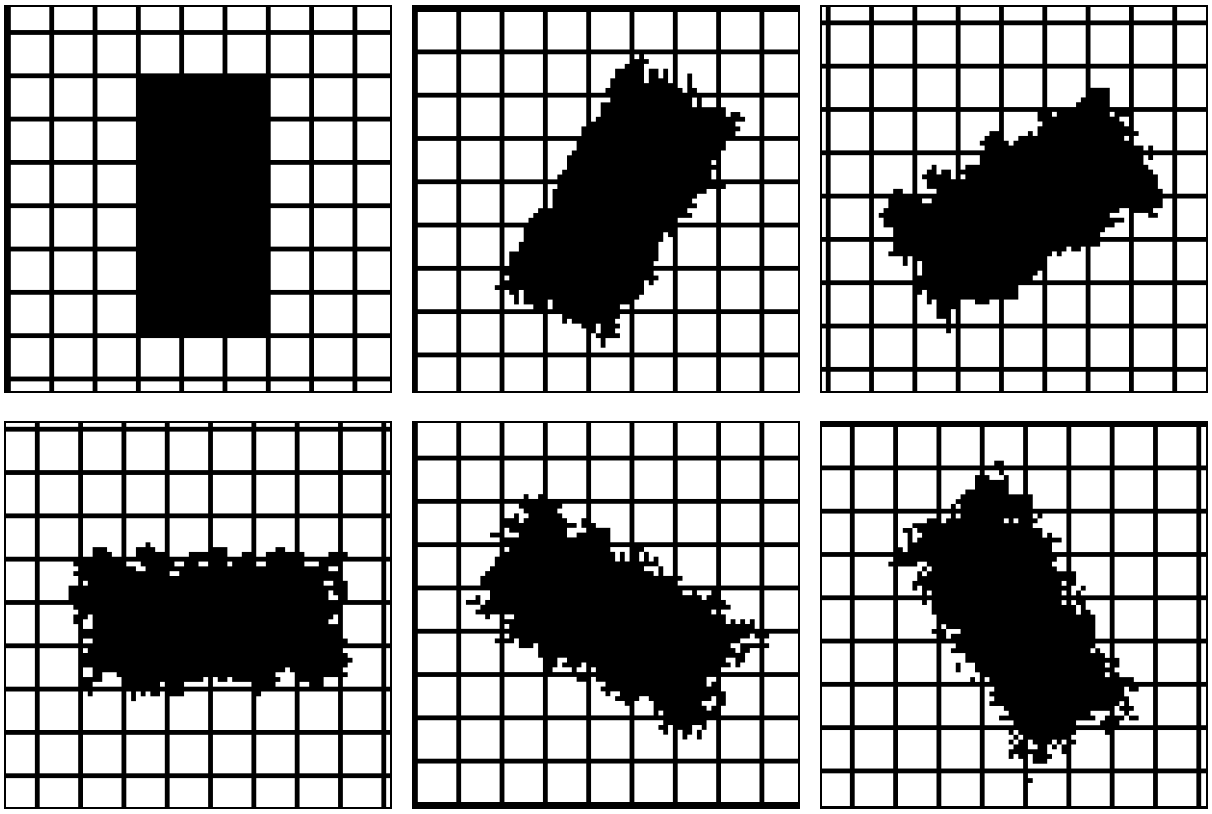


Fig. 8.6: Form der Aktivitätswolke, ausgehend von der Startwolke (oben links) nach Drehung um jeweiliges 30°.

In Fig. 8.6 ist die Form der Aktivitätswolke nach einer Drehung um jeweils 30° zu sehen. Die Wolke dreht sich um die Kamera und bleibt dabei in ihrer Form stabil. Lediglich am Rand sind die Konturverläufe etwas unscharf, was auf die zusammengesetzte Bewegung bei der Drehung zurückzuführen ist. Diese leichten Veränderungen sind aber tolerierbar. Im folgenden werden Mechanismen vorgestellt, die zur Verbesserung des Ergebnisses beitragen.

8.7 Auflösung des Feuerabstandes

Bei der Bearbeitung der Zeitskala ist die kleinste betrachtete Einheit ein Zeitschritt t_i. Die aktuellen Feuerzeitpunkte der einzelnen Shiftneuronen liegen jeweils auf einem solchen Zeitschritt. Um den Spikeabstand für die Shiftneuronen bei einer Drehung zu bestimmen, muß die Anzahl der Zeitschritte für eine Rotation um 360° festgelegt werden. Hieraus wird dann der Spikeabstand der jeweils beteiligten Shiftneuronen, der auch als *Auflösung des Feuerabstandes* bezeichnet wird, berechnet. In den Simulationen hat sich gezeigt, daß die Anzahl der für eine vollständige Drehung notwendigen Zeitschritte die Qualität des Ergebnisses beeinflußt.

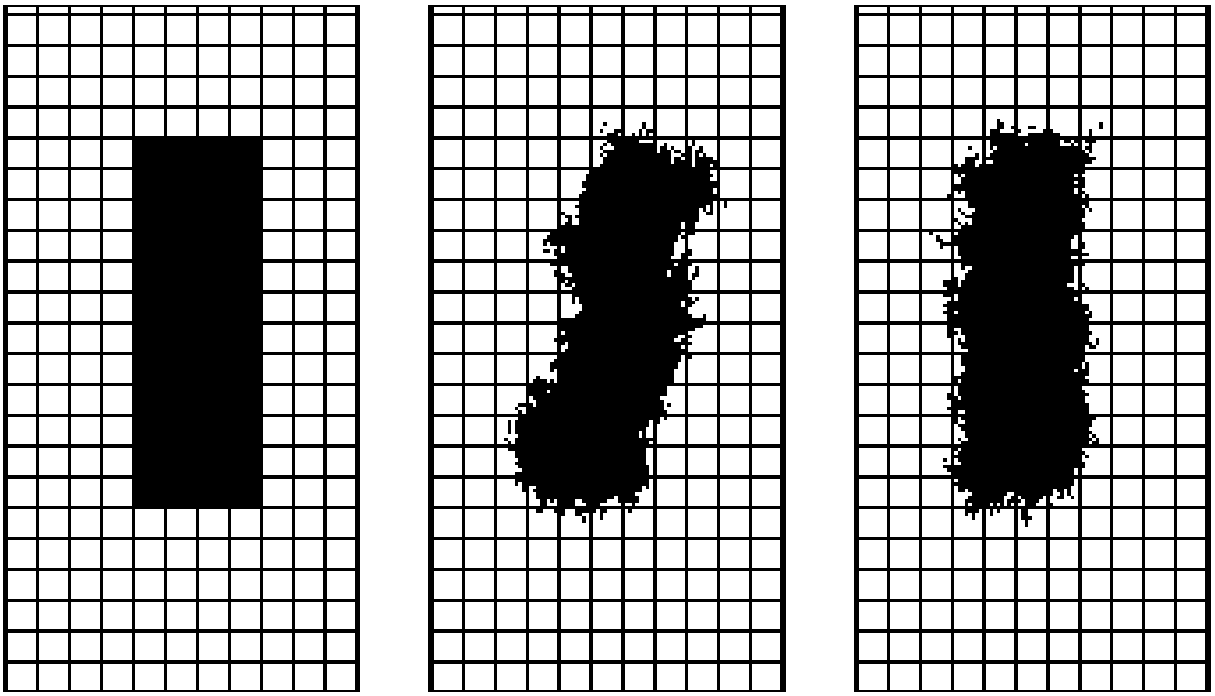


Fig. 8.7: Je nach Auflösung der Spikeabstände der Shiftneuronen verändert sich die Qualität des Ergebnisses.

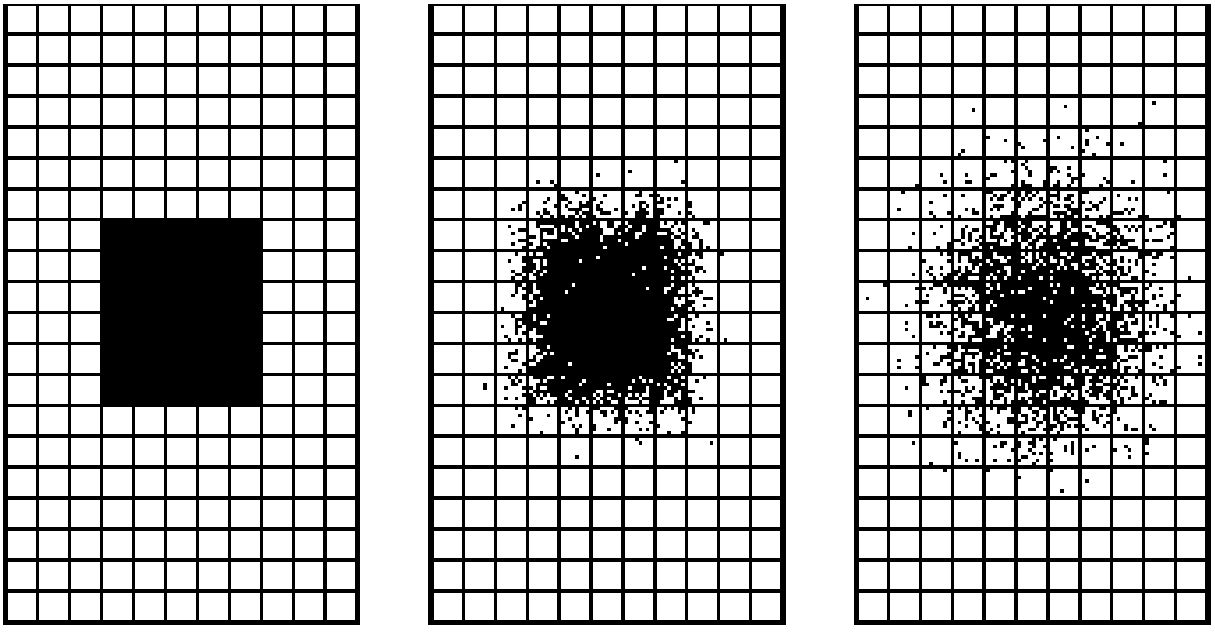
Wählt man eine geringe Auflösung, das heißt eine geringe Anzahl von Zeitschritte, feuern die Neuronen, die sich weiter entfernt vom Drehzentrum befinden, zu oft. Dies hat zur Folge, daß die Aktivitätswolke nach einer Umdrehung um 360° nicht wieder an der Anfangsposition liegt, sondern leicht verdeht ist (Fig. 8.7b). Wird die Auflösung erhöht, befindet sich die Aktivitätswolke nach einer Umdrehung an der richtigen Position und in der richtigen Orientierung (Fig. 8.7c).

8.8 Laterale Koppelung

Bewegt sich die Kamera auf einer Kreisbahn um das Objekt, bleibt die Aktivitätswolke nicht kompakt. Am Rand der Wolke bilden sich isolierte aktive Neuronen. Es sieht so aus, als franse die Aktivitätswolke langsam aus. Besonders deutlich ist dies zu sehen, wenn sich die Aktivitätswolke mehrmals um die Kamera dreht.

In Fig. 8.8 ist links die Startaktivitätswolke (schwarzes Rechteck) zu sehen. Nach einer Umdrehung bilden sich am Rand einzelne isolierte aktive Neuronen (8.8b). Besonders deutlich wird dieser Effekt in (8.8c), wo die Form der Aktivitätswolke nach fünf Umdrehungen dargestellt ist. Es haben sich sehr viele isolierte aktive Neuronen gebildet, so daß das Objekt nicht mehr durch eine kompakte Aktivitätswolke dargestellt wird.

Daneben ist zu beobachten, daß sich bei der Drehung der Aktivitätswolke isolierte passive Gitterneuronen in der Aktivitätswolke bilden (Fig. 8.8b). Auch dieser Effekt verstärkt sich mit der Anzahl der Drehungen (Fig. 8.8c).



a b c

Fig. 8.8: Aktivitätswolke ohne lateraler Koppelung.

Wünschenswert ist aber, eine kompakte Aktivitätswolke zu behalten. Toleriert wird nur eine leichte Verformung der Gesamtstruktur. Dies wird durch eine laterale Koppelung von Nachbarneuronen erreicht. Jedes Gitterneuron ist mit seinen vier direkten Nachbarn verbunden. Die

laterale Koppelung ist so eingestellt, daß ein Neuron, das mindestens drei aktive Nachbarn hat, durch diese Nachbarn aktiv gehalten wird.

Hingegen kann ein aktives Neuron, das nur von passiven Neuronen umgeben ist, seine Aktivität durch seine Rückkopplung nicht selber über die Schwelle halten und wird nach einiger Zeit passiv. So können sich nur in geringem Maß isolierte aktive und passive Neuronen bilden. Die Aktivitätswolke bleibt stabil.

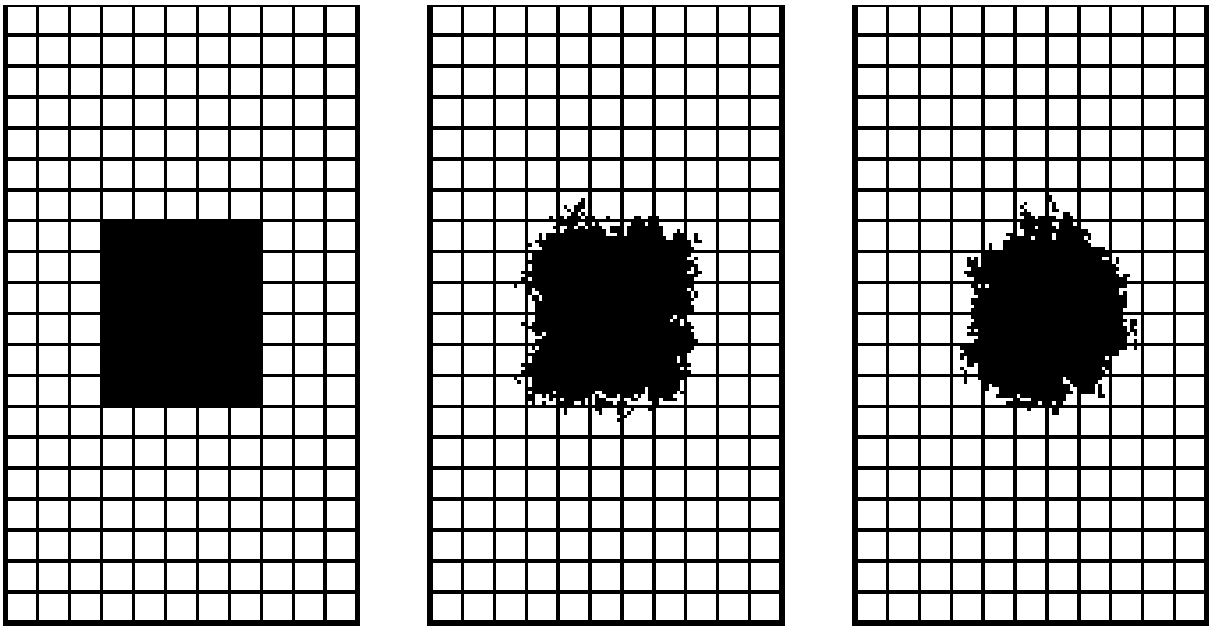


Fig. 8.9: Aktivitätswolke mit lateraler Koppelung.

Bei der lateralen Kopplung kommt es nicht darauf an, die isolierten Gitterneuronen sofort zu beeinflussen. Weitaus bedeutender ist es, das richtige Maß der Kopplung zu finden. Beeinflussen sich die Nachbarneuronen zu stark, d. h. werden isolierte Gitterneuronen sofort von ihren Nachbarneuronen deaktiviert, löst sich die Aktivitätswolke sehr schnell auf. Schon nach kurzer Zeit sind keine aktiven Neuronen mehr vorhanden. Anderseits kann sich die Aktivitätswolke vergrößern, so daß am Ende nicht mehr die Form der Startwolke zu erkennen ist. Besonders wenn das Objekt Aussparungen oder Bohrungen besitzt, verschwinden diese schnell. Es hat sich gezeigt, daß es sinnvoll ist, eine laterale Koppelung nach jeweils 15° durchzuführen.

8.9 Polarkoordinatensystem

Bisher wurden alle Betrachtungen für das zweidimensionale kartesische Koordinatensystem vorgenommen. Biologisch plausibler erscheint jedoch eine Betrachtung im Polarkoordinatensystem. Deshalb wurde die Repräsentation des Raumes im Polarkoordinatensystem untersucht und implementiert [Bredenbals 1993].

Im betrachteten Polarkoordinatensytem befindet sich die Kamera in gleicher Höhe wie die Objekte. Sie kann sich um ihre eigene Achse drehen (φ^+ und φ^-) und sich vor (r^+) sowie zurück (r^-) bewegen. Bei der Implementierung wurde davon ausgegangen, daß nur ein Objekt vorhanden ist, dessen Aktivitätswolke bereits erzeugt wurde. Das Erzeugen von Aktivitätswolken durch das Anschauen von Objekten wird im Kapitel 12 behandelt.

8.9.1 Bewegungen im Polarkoordinatensystem

Die Gitterneuronen sind in Ringen zu je 120 Domänen mit je 8^2 Neuronen um den Ursprung verteilt (Fig. 8.10). Der Abstand zwischen zwei Ringen vergrößert sich mit zunehmenden Abstand. Insgesamt gibt es 60 *Domänenringe*. Diese Anordnung hat zur Folge, daß entfernte Domänen eine größere Fläche abdecken als Domänen, die sich näher am Ursprung befinden. Da jede Domäne aus gleich vielen Neuronen besteht, gilt dies auch für die Neuronen.

Im Polarkoordinatensystem besitzt ein Gitterneuron direkte Nachbarn in r- und φ-Richtung. Somit kann die Aktivität in r- und φ-Richtung innerhalb des Netzes verschoben werden. Es gibt Shiftneuronen für die Drehung φ^+ und φ^- und die Vor- und Zurückbewegung der Aktivität in r^+- und r^--Richtung. Die Verbindungsstruktur der Gitterneuronen untereinander und der Shiftneuronen zu den Gitterneuronen wird aus dem kartesischen Koordinatensystem übernommen. Deshalb verändert sich der Mechanismus zum Verschieben der Wolke nicht und wird nicht nochmals betrachtet (vgl. Kapitel 6.1).

Nun sind die Bewegungen der Kamera und die sich ergebene Verschiebung der Aktivitätswolke zu diskutieren. Bei einer Drehung der Kamera um die Hochachse verschiebt sich die Aktivität entlang der Gitterneuronen in φ-Richtung in die entgegengesetzte Richtung. Sie entspricht der translatorischen Bewegung im kartesischen System (vgl. Kapitel 6.1).

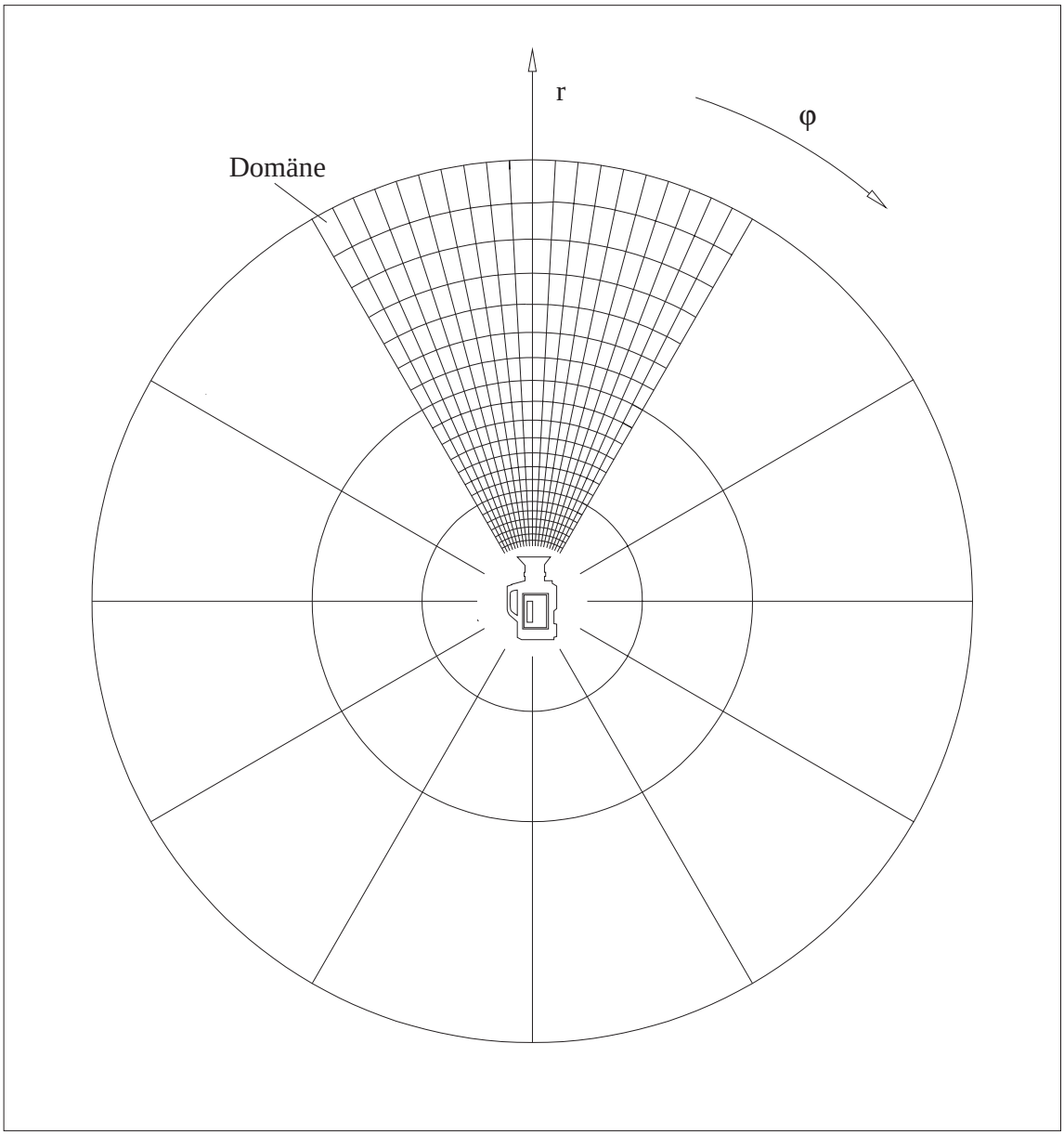


Fig. 8.10: Grundstruktur des RRN in Polarkoordinaten.

Schwieriger ist die Vor- und Zurückbewegung der Kamera zu beschreiben. Bewegt sie sich z.B. mit der Geschwindigkeit V_K in r^+-Richtung, verschiebt sich die Aktivitätswolke nicht nur in r^--Richtung auf den Ursprung zu. Stattdessen bewegt sie sich um x_{RRN} parallel zur Kamerabewegung, d. h. sie macht eine Bewegung, die sich aus einer r^-- und φ-Komponente zusammensetzt (vgl. Fig. 8.11). Dies ist vergleichbar mit der Drehung im kartesischen Koordinatensystem (vgl. Kapitel 6.2).

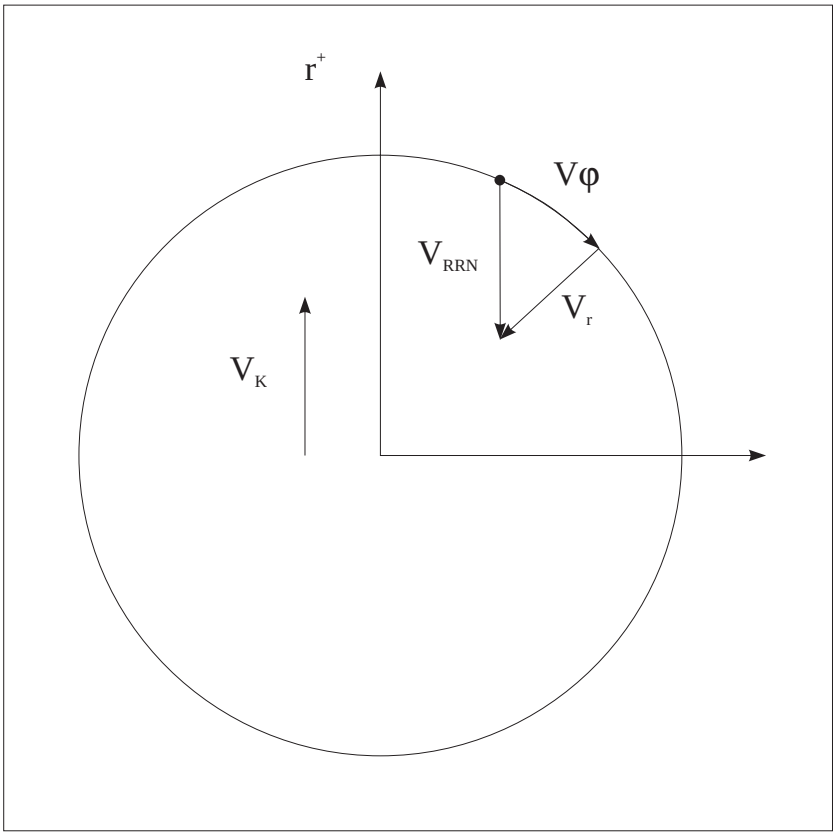


Fig. 8.11: Eine Bewegung der Kamera nach r^+ wird durch eine Verschiebung der Wolke nach φ und r^- realisiert.

8.9.2 Größeninvarinz im Polarkoordinatensystem

Wie schon in Kapitel 7.3 erwähnt, gilt auch für dieses System, daß Objekte, die sich nah am Ursprung befinden, eine größere Aktivitätswolke besitzen als Objekte gleicher Größe, die weiter entfernt sind. Die Größe der Aktivitätswolke wird durch die Zahl der aktiven Neuronen angegeben. Da sich aber die Repräsentation der Form und Größe der Objekte selbst nicht ändern darf, muß die Fläche, die durch die Neuronen abgedeckt wird, immer gleich groß sein.

Die Vergrößerung bzw. Verkleinerung von Aktivitätswolken durch unterschiedliche Feuerfrequenzen der Shiftneuronen wurde bereits in Kapitel 7.3 für eine eindimensionale Wolke diskutiert. Bei der Erweiterung auf das zweidimensionale System steuert ein Shiftneuron die Verschiebung der Aktivität in einer Domäne anstatt in nur einer Neuronenreihe. Ansonsten bleibt der Mechanismus der gleiche.

In Fig. 8.12 sind zwei Aktivitätswolken für ein Objekt in unterschiedlicher Entfernung zu sehen. Im oberen Bild befindet sich die Wolke in der Mitte des Koordinatensystems. Jedes aktive Gitterneuron wird durch einen Punkt dargestellt. Die Linien symbolisieren nur die

Anordnung und Größe der einzelnen Domänen. Verschiebt sich nun die Wolke an den äußeren Rand des Koordinatensystems, verringert sich die Zahl der aktiven Gitterneuronen, während die abgedeckte Fläche gleich bleibt. In Fig. 8.12 unten ist die Repräsentation des gleichen Objekts aus dem oberen Bild zu sehen, in dem die Wolke bis an den äußeren Rand des Koordinatensystems verschoben wurde.

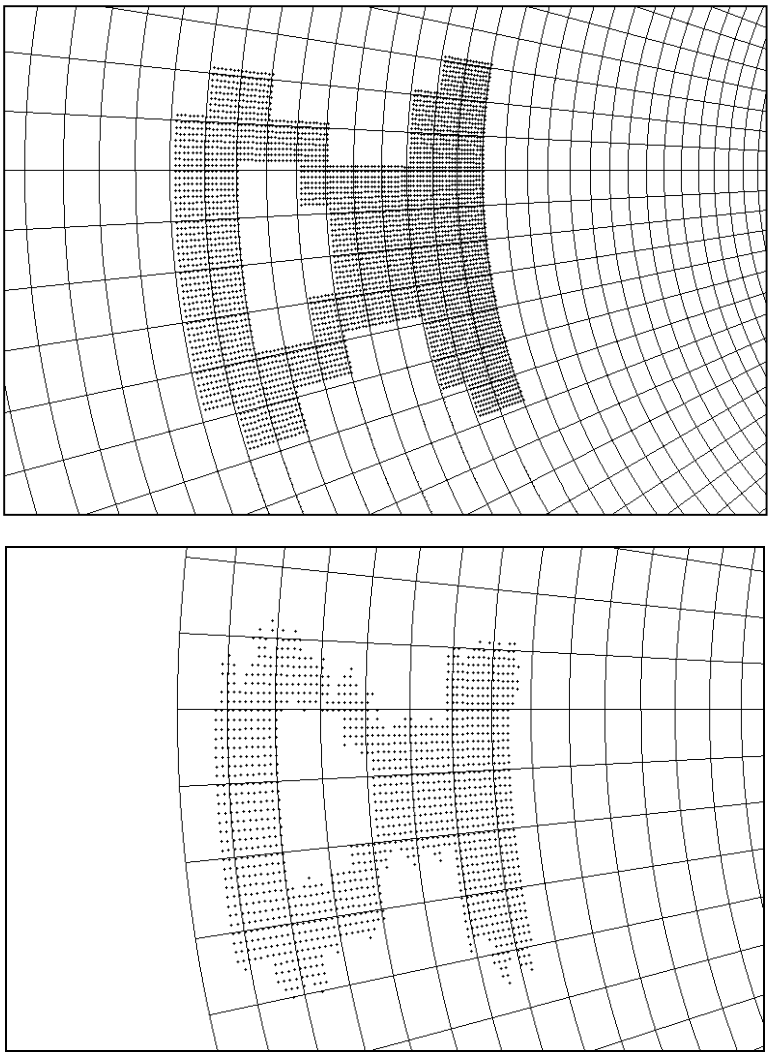


Fig. 8.12: Aktivitätswolken des gleichen Objektes in unterschiedlichen Entfernungen vom Ursprung.

9. Dreidimensionale Raumrepräsentationsnetzwerk

Bisher beschränkten sich die Betrachtungen auf das zweidimensionale System. Doch um den Raum in der Umgebung der Kamera zu repräsentieren, ist es notwendig, das RRN auf den dreidimensionalen Fall auszudehnen. In Kapitel 6 wurde die entsprechende Theorie bereits kurz erläutert. Nun soll die Erweiterung der Programme vorgestellt werden. Um die nötige Rechenleistung zu erreichen, wurde die Parallelisierung des RRN untersucht. Anschließend wurde eine sequentielle Version erstellt.

9.1 Erweiterung der Gitter- und Shiftneuronen

Die im zweidimensionalen System verwendeten Datenstruktur (vgl. Kapitel 8.1ff), die im folgenden als *Domänenscheibe* bezeichnet wird, wurde entsprechend erweitert. Für eine Domäne der Größe 8^3 Neuronen werden für die dritte Ausdehnung acht Domänenscheiben übereinandergelegt. Somit wird eine Domäne durch 16 Integerzahlen codiert. Das Verschieben von Aktivität innerhalb einer Domäne wird dann durch Shift-, Verknüpfungs- und zusätzliche Zuweisungsoperationen realisiert. Um die nachfolgende Beschreibung zu vereinfachen, werden die Domänenscheiben in z^+ -Richtung von 1 bis 8 fortlaufend numeriert.

Um die Aktivität in die x- und y-Richtung zu verschieben, werden die Anweisungen aus Kapitel 8.1 achtmal für jede Domänenscheibe durchgeführt. Die Verschiebung in z-Richtung ist noch einfacher. Muß z. B. die Aktivität in z^+-Richtung verschoben werden, werden die einzelnen Domänenscheiben in z^--Richtung abgearbeitet. Zuerst wird die Domänenscheibe 8 auf die Nachbarscheibe 1 in der Nachbardomäne übertragen. Dann werden jeweils die anderen Domänenscheiben i auf die Scheibe i+1 kopiert (i = 2, ..., 8). Hierdurch ist schon die gesamte Aktivität verschoben worden. Für die Verschiebung der Wolke in z^--Richtung wird in umgekehrter Reihenfolge vorgegangen.

Die Simulation der Shiftneuronen muß ebenfalls auf den dreidimensionalen Fall erweitert werden. Hierzu sind die Tabellen für die zusätzlichen Shiftneuronen z^+ und z^- zu erzeugen. Zusätzlich zu den bisherigen Bewegungsrichtungen sind nun translatorische Bewegungen in z-Richtung und Drehungen um die x- und y-Achse erlaubt. Für die translatorische Bewegung sind nur Feuerabstands- und Feuerzeitpunkttabellen für die z^+- und z^--Richtung zu erstellen. Die Drehung um die x-Achse läßt sich durch Verschieben der Wolke in z- und y-Richtung und die Drehung um die y-Achse durch eine Bewegung in x- und z-Richtung erreichen. Hierzu sind

entsprechende Tabellen wie bei der Drehung um die z-Achse zu erzeugen. Bei jeder Drehung sind immer nur zwei Shiftneuronen aktiv.

Für jede zusätzliche Bewegungsrichtung wird eine Zeitskala aufgebaut. Außerdem wird nun ein dreidimensionales Aktivitätsfenster um die Domänen mit aktiven Neuronen gelegt, und nur diese Domänen werden simuliert. Damit auf der Zeitskala keine Zeitschritte betrachtet werden, in denen kein Shiftneuron aktiv ist, wird ebenfalls das Zeitsprungverfahren angewandt (vgl. Kapitel 8.4).

9.2 Parallelisierung

Nun wurde das RRN parallelisiert. Da eine Domäne sehr kompakt kodiert ist, ist es nicht sinnvoll, auf einem parallelen System Teile von Domänen auf verschiedene Rechner zu verteilen. Als kleinste elementare Einheit wurde deshalb eine Domäne definiert.

Das dreidimensionale neuronale Netz läßt sich durch zwei Prozesse simulieren. Der erste Prozeß wird als "Master" bezeichnet. Er verwaltet die gesamte Verschiebung der Aktivitätswolke und läuft nur auf dem Root-Transputer, während der Prozeß "Shift" die Daten aller Domänen der Aktivitätswolke hält und die Aktivität der Neuronen verschiebt. Er läuft auf allen anderen Transputern.

9.2.1 Master-Prozeß

Aktiviert wird der Master durch die Bewegungsimpulse des externen Bewegungssystems, das diese bei jeder Bewegung sendet (Fig. 9.1). Der Master ermittelt jeweils zuerst den zu simulierenden Zeitabschnitt und die ersten Domänen, deren Neuronen einen Shiftimpuls erhalten. Diese Domänen werden im folgenden als aktive Domänen bezeichnet. In einem Zeitschritt wird immer Aktivität in mehreren Domänen verschoben. Eine Begründung hierfür folgt in Abschnitt 9.2.2.

Als nächstes arbeitet der Master in einer Schleife zwei parallele Prozeduren ab. Die erste Prozedur versendet die Shiftimpulse an die aktiven Domänen auf den einzelnen Transputerns. Anschließend wird auf ein Statussignal gewartet, das die Shiftprozesse zurücksenden, wenn alle Verschiebungen vorgenommen wurden. Anhand des Statussignals ist zu erkennen, ob und wie das Aktivitätsfenster verändert werden muß. Wenn eine Veränderung notwendig ist, ist das Fenster entweder zu vergrößern oder zu verkleinern. Wird das Fenster verkleinert,

werden an die entsprechenden Shiftprozesse Signale zum Löschen der Domänen gesendet. Im anderen Fall werden zusätzliche Domänen erzeugt und auf die Transputer verteilt.

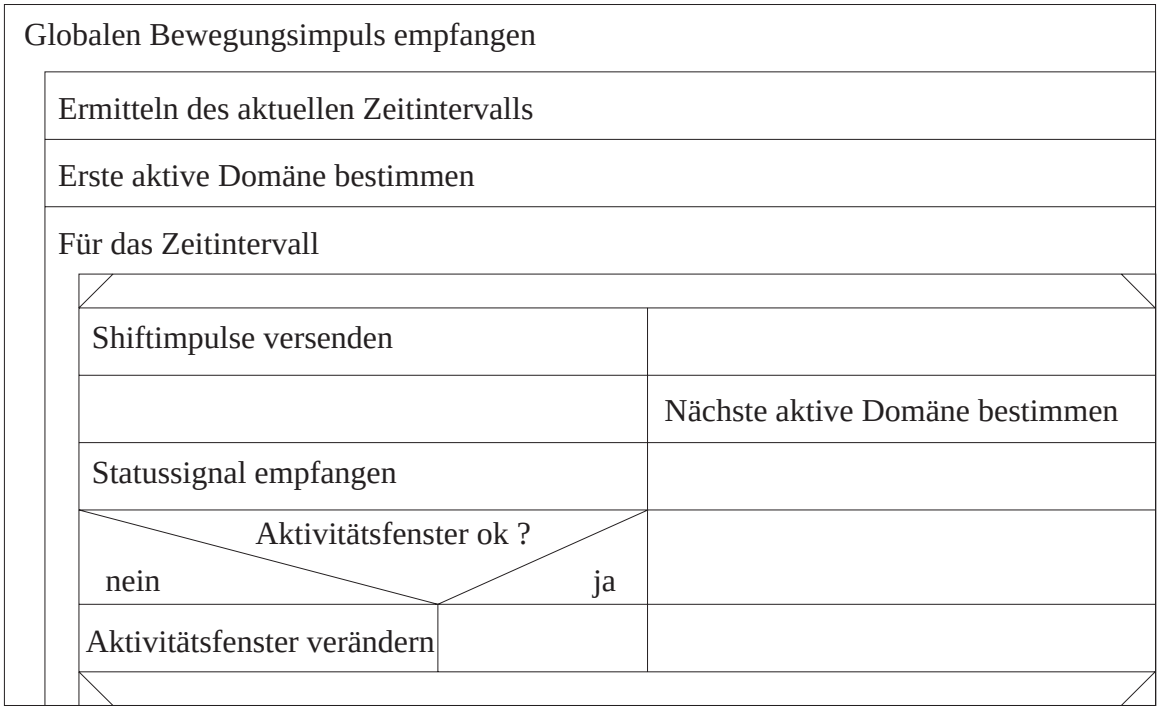


Fig. 9.1: Struktogramm des Masterprozesses

Die zweite Prozedur bestimmt parallel die nächsten Domänen, in denen die Aktivität der Neuronen verschoben werden muß. Da die Aufgabe der ersten Prozedur fast ausschließlich aus dem Senden und Empfangen von Daten besteht, eignet sich die Aufgabe der zweiten Prozedur sehr gut für eine parallele Abarbeitung.

Die beschriebene Aufgabe des Masters ist im Verhältnis zur Verschiebung der Aktivität so gering, daß sie von einem Transputer ausgeführt werden kann. Um die Shiftimpulse auf kürzestem Weg zu den einzelnen Shiftprozessen schicken zu können, wurde als grundlegende Netzwerkstruktur für die Transputer ein Gitter gewählt.

9.2.2 Verteilung der Domänen

Wie bereits erwähnt, werden die Domänen des Aktivitätsfensters auf die anderen Transputer des Netzwerkes verteilt. Auf diesen Transputern laufen Shift-Prozesse. Doch bevor diese beschrieben werden können, muß die Verteilung der einzelnen Domänen vorgestellt werden.

Da bei der Simulation nur jeweils der Bereich betrachtet wird, in dem sich aktive Neuronen befinden, ist es von großer Bedeutung, diesen Bereich sinnvoll auf die Transputer zu verteilen. Hierbei kommt es in erster Linie darauf an, daß bei einer Bewegung alle Transputer gleichmäßig ausgelastet sind, d. h. daß sie die gleiche Anzahl von aktiven Domänen im gleichen Zeitschritt haben. Außerdem soll der Kommunikationsaufwand möglichst klein gehalten werden. Dies soll auch dann noch gelten, wenn sich das Aktivitätsfenster bei einer Bewegung verändert.

Für die translatorischen Bewegungen gilt, daß die Geschwindigkeit überall gleich groß ist und sich die Aktivitätswolke gleichmäßig verschiebt. Somit können alle Neuronen der Domänen gleichzeitig einen Shiftimpuls erhalten. Dies bedeutet, daß es sinnvoll ist, die Domänen gleichmäßig auf alle Transputer zu verteilen. Bei Veränderungen des Aktivitätsfensters kommen entweder Domänen hinzu oder es werden Domänen entfernt. Hinzukommende Domänen müssen wieder gleichmäßig auf alle Transputer verteilt werden, und zwar vorzugsweise auf solche Transputer, auf denen vorher Domänen entfernt wurden. Nach dem Entfernen von Domänen ist es nicht praktikabel, anschließend Domänen zwischen zwei Transputern auszutauschen, um wieder eine Gleichverteilung zu erhalten, Da hierdurch ein unnötiger Kommunikationsaufwand entstünde. Dieser steht in keinem Vergleich zu dem Zeitverlust, der durch die Verwaltung von einer leicht unterschiedlichen Anzahl aktiver Domänen auf den einzelnen Transputern entsteht. Vielmehr ist schon bei der Anfangsverteilung darauf zu achten, daß bei einer Verkleinerung des Aktivitätsfensters Domänen auf allen Transputern gelöscht werden.

Da nur Aktivität zwischen Neuronen von direkt benachbarten Domänen ausgetauscht wird, ist es außerdem wichtig, daß direkt benachbarte Domänen auf dem gleichen oder einem benachbarten Transputer zu liegen kommen. Hierdurch ist nur Kommunikation zwischen benachbarten Transputern notwendig.

Schwieriger wird die Aufteilung der Domänen auf die einzelnen Transputer für die rotatorischen Bewegungen. Hier kann nicht mehr von einer Verschiebung aller Aktivität zum gleichen Zeitpunkt ausgegangen werden. Die Domänen erhalten ihre Shiftimpulse unabhängig voneinander.

Um die Domänen aber trotzdem sinnvoll auf die einzelnen Transputer zu verteilen, muß untersucht werden, ob es zwischen den Feuerzeitpunkten der einzelnen Domänen eine Abhängigkeit gibt. Hierzu betrachten wir zuerst den zweidimensionalen Fall. Allgemein gilt für die anteilige Bewegung der x-Komponente von zwei Punkten mit Radius r_1 und r_2 vom Ursprung und den Winkeln α und β in der gleichen Zeile l nach Formel 6.2 (Fig. 9.2):

$$v_{x1}(\vec{\Omega}) = r_1 \cdot \vec{\Omega} \cdot \sin\alpha \text{ und } v_{x2}(\vec{\Omega}) = r_2 \cdot \vec{\Omega} \cdot \sin\beta. \qquad (9.1)$$

Hieraus ergibt sich mit $r_1 = l / \sin\alpha$ und $r_2 = l / \sin\beta$:

$$\begin{aligned} v_{x1} &= r_1 \cdot \vec{\Omega} \cdot \sin\alpha \qquad (9.2)\\ &= l / \sin\alpha \cdot \vec{\Omega} \cdot \sin\alpha \\ &= l \cdot \vec{\Omega} \\ &= r_2 \cdot \vec{\Omega} \cdot \sin\beta \\ &= v_{x2} \end{aligned}$$

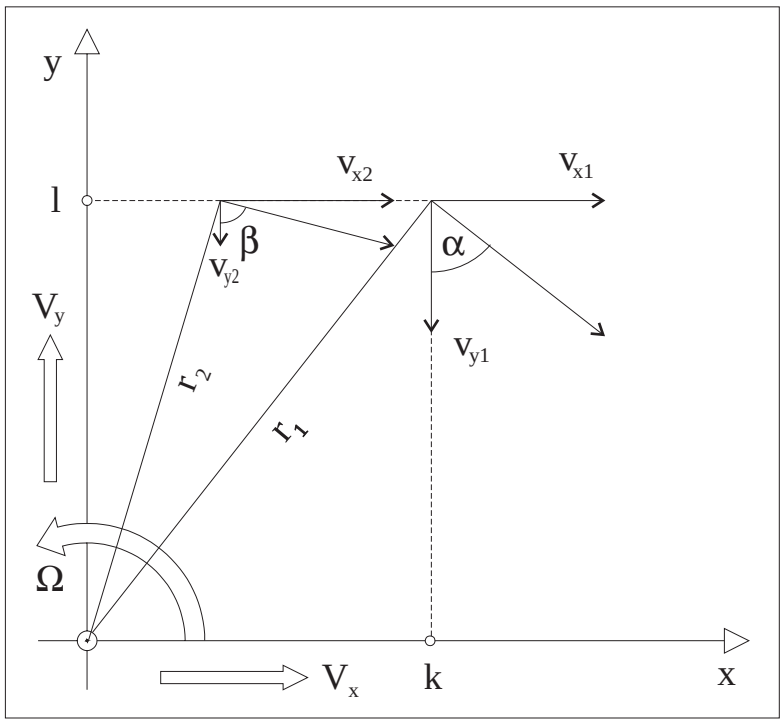


Fig. 9.2: Für alle Punkte in der Zeile l ist die Geschwindigkeit v_x gleich groß.

Es folgt, daß in jeder Domäne in der Zeile l eine Aktivität mit der gleichen Geschwindigkeit verschoben wird. Die gleiche Berechnung läßt sich für die anteilige Bewegung der y-Komponente für jede Zeile l durchführen. Somit kann die Verschiebung der Aktivität in einer Zeile im gleichen Zeitschritt vorgenommen werden. Die gleichen Bedingungen gelten für die Verschiebung in der Spalte k.

Für die Rotationen im dreidimensionalen Raum gilt, daß nicht nur eine Domäne den Abstand r zur Drehachse hat, sondern ein ganzer *Domänenstrang* parallel zur Drehachse steht (vgl. Fig. 9.3a). Zusammengefaßt ergibt sich nun, daß jeweils eine Domänenscheibe für alle Rota-

tionsrichtungen im gleichen Zeitpunkt betrachtet werden kann. In Fig. 9.3b sind die möglichen Domänenscheiben für die drei Drehrichtungen zu sehen.

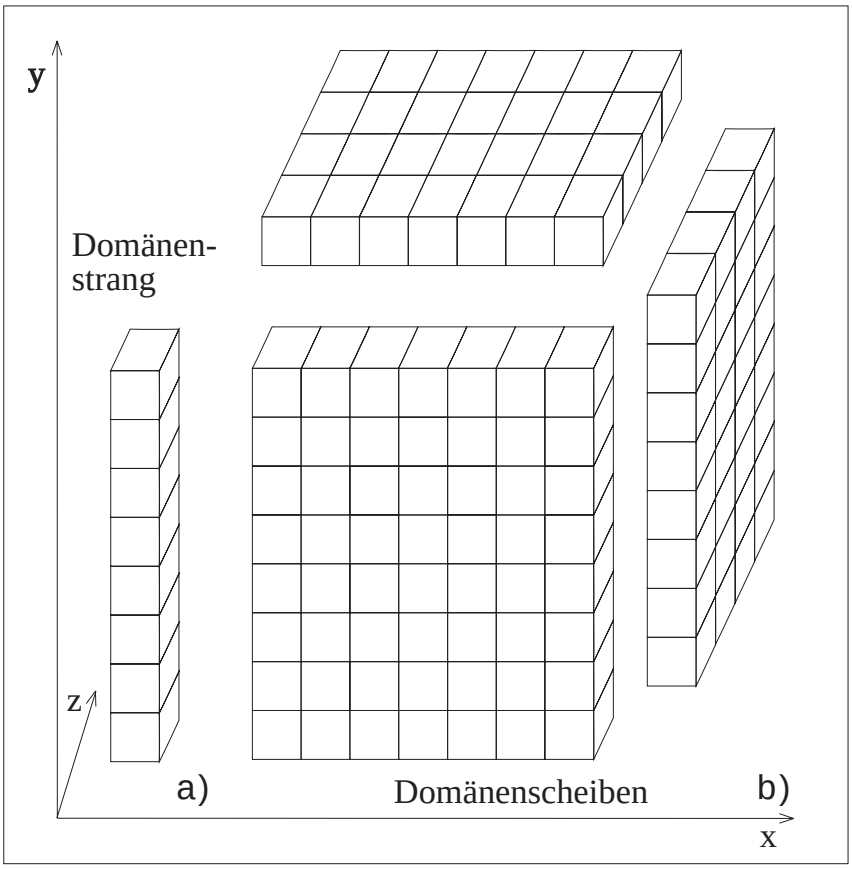


Fig. 9.3: Abhängigkeiten von Domänen

Zwischen den einzelnen Domänenscheiben gibt es keine Abhängigkeit. Deshalb erhalten sie ihre Shiftimpulse unabhängig voneinander. Somit kommt es nun darauf an, die Domänenscheiben für alle Rotationen gleichmäßig auf alle Transputer zu verteilen. Bei der Betrachtung nur einer Rotationsrichtung könnte jeder Strang einer Domänenscheibe auf einen Transputer gelegt werden. Da jedoch die Anzahl der einzelnen Stränge nur im seltensten Fall mit der Anzahl der Transputer übereinstimmt, haben die Transputer eine unterschiedliche Zahl von Domänensträngen und somit auch Domänen zu bearbeiten. Dieses Verhältnis wird besser, wenn die einzelnen Domänen alternierend auf die Transputer verteilt werden (Fig. 9.4). Diese Aufteilung garantiert, daß für jede Rotationsrichtung die Transputer etwa gleichmäßig ausgelastet sind. Außerdem entspricht die Aufteilung den Überlegungen zur Verteilung der Domänen für die translatorischen Bewegungen.

Bei der Betrachtung der benötigten Kommunikationsverbindungen zum Austausch der Aktivitäten zwischen den einzelnen Transputern ergibt sich, daß jeder Transputer nur mit zwei anderen Transputern kommunizieren muß. Beispielhaft ist dies für Transputer T2 in Fig. 9.4 zu

sehen. Seine direkten Nachbardomänen befinden sich immer auf Transputer T1 oder T3, unabhängig von der betrachteten Richtung. Als ideale Verbindungsstruktur bietet sich somit ein Ring an. Es brauchen keine Aktivitätsdaten über mehrere Transputer geschoben, zu werden.

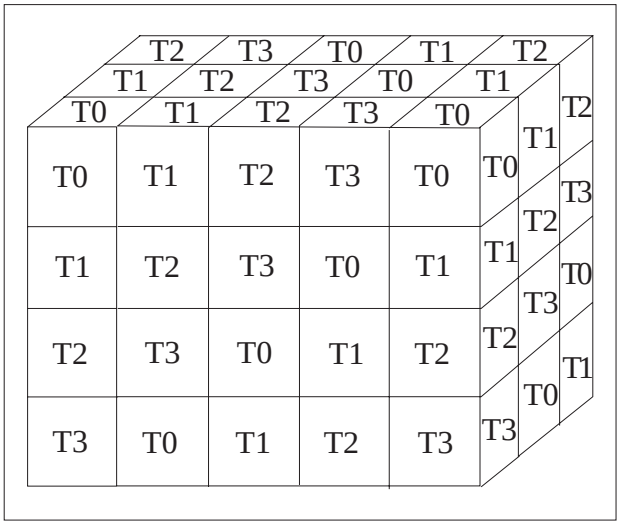


Fig. 9.4: Die Aufteilung der Domänen auf die Transputer T0 bis T3.

9.2.3 Shift-Prozeß

Nachdem die Verteilung der Domänen auf die einzelnen Transputer feststeht, kann der Prozeß Shift besprochen werden (Fig. 9.5). Wenn ein Impuls vom Masterprozeß ankommt, verschiebt sich die Aktivität in den entsprechenden Domänen. Die Verschiebung vollzieht sich in drei Schritten.

Im ersten Schritt wird für alle Domänen, die einen Shiftimpuls erhalten und nicht am vorderen Rand der Aktivitätswolke liegen, die Aktivität der Neuronen, die am vorderen Rand liegen, ermittelt und zwischengespeichert. Diese Werte werden im folgenden als *Randaktivität* bezeichnet. Die Randaktivität muß an andere Domänen auf dem benachbarten Transputer gesendet werden. Da immer alle Neuronen einer Domänenscheibe im gleichen Zeitschritt einen Shiftimpuls erhalten, ist der vordere Nachbar einer Domäne auch immer aktiv und braucht die Randaktivität von seinem Hintermann. Deshalb kann die Randaktivität im voraus ermittelt werden. Außerdem kann durch eine Abfrage in den Domänen am vorderen Rand des Aktivitätsfensters getestet werden, ob das Aktivitätsfenster zu vergrößern ist.

Danach wird die Randaktivität an die entsprechenden Nachbardomänen auf den benachbarten Transputern gesendet und gleichzeitig die Randaktivität von dem anderen benachbarten Tran-

sputer empfangen. Parallel hierzu wird die Aktivität der Neuronen innerhalb der einzelnen aktiven Domänen verschoben. Durch Testen der Aktivitäten der Neuronen der vorletzten Domänenreihe des Aktivitätsfensters wird ermittelt, ob es verkleinert werden kann.

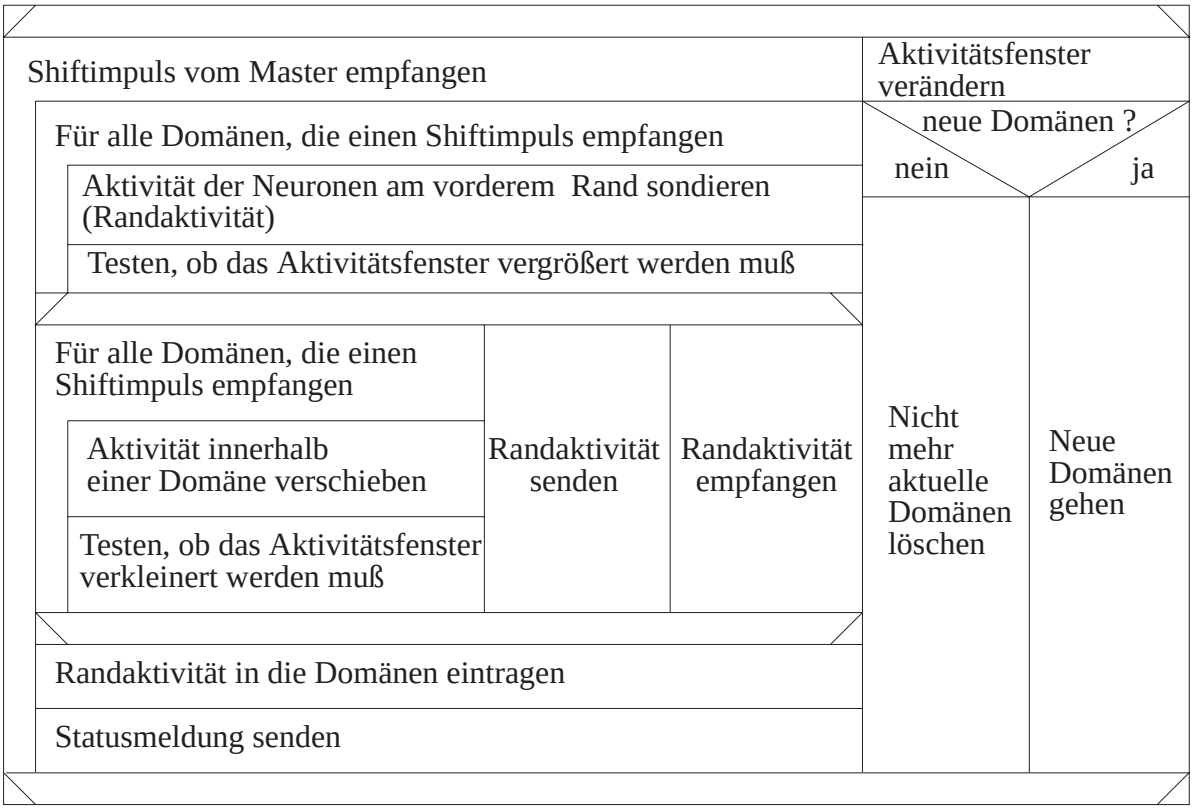


Fig. 9.5: Struktogramm des Shiftprozesses.

Zum Abschluß muß die empfangene Randaktivität den entsprechenden Domänen zugewiesen und ein Statussignal an den Masterprozeß gesendet werden. Dieses Signal dient gleichzeitig zur Synchronisation der einzelnen Shift-Prozesse.

Parallel hierzu läuft auf dem Shift-Prozeß eine Prozedur, die aufgerufen wird, wenn das Aktivitätsfenster verändert werden muß. Dieser Aufruf wird vom Master generiert. Falls das Fenster zu vergrößern ist, gibt der Master an, welche Domänen neu zu setzen sind. Ansonsten sind die Domänen, die nicht mehr aktuell sind, zu löschen.

Außerdem ist es noch möglich, die Aktivität aller Neuronen an den Master zu senden, wo sie dann zu einer graphischen Ausgabe geleitet werden. Die entsprechenden Prozeduren wurden in die beiden Struktogramme nicht mit aufgenommen.

9.3 Sequentielle RRN

Da der T9000-Transputer nicht ausgeliefert wurde, wurde das RRN auf einem sequentiellen Rechner implementiert. Hierbei ging man von der beschriebenen parallelen Version aus, um die beiden Versionen vergleichen zu können. Da nur der parallele Code zu einem sequentiellen Programm vereinigt wurde, soll hier eine kurze Übersicht gegeben werden, die nur auf die markanten Veränderungen eingeht.

Der Master- und der Shiftprozeß werden zu einem Prozeß verschmolzen. Die einzelnen Domänen brauchen nicht mehr verteilt werden, sie werden daher in einem Array gespeichert. Gestartet wird das Programm mit dem ersten Teil des Masterprozesses. Nach dem Versenden der Shiftimpulse wird die Funktion für den Shiftprozeß aktiviert. Diese sondiert die Randaktivität und verschiebt dann die Aktivität innerhalb der Domänen, die einen Shiftimpuls erhalten. Für jede dieser Domänen, die am Rand des Aktivitätsfensters liegen, wird überprüft, ob durch die Verschiebung der Wolke das Aktivitätsfenster zu verändern ist, und gegebenenfalls ein entsprechendes Flag gesetzt. Das Senden und Empfangen der Randaktivität entfällt, da sie direkt in die Domänenscheibe der entsprechenden Neuronen eingetragen wird. Nach dem gesamten Durchlauf wird nun das Flag des Aktivitätsfensters überprüft und gegebenenfalls verändert. Abschließend wird die nächste aktive Domäne bestimmt.

9.4 Grafische Oberfläche

Zu der parallelen und sequentiellen Version des RRN wurde eine einheitliche Oberfläche geschaffen, um die Ergebnisse anzuzeigen. Es ist möglich, abgespeicherte Ergebnisdateien anzuzeigen sowie während des Programmablaufs die Aktivitätswolke zu verfolgen. Hierbei kann die Aktivitätswolke in verschiedenen Auflösungen angezeigt werden.

In Fig. 9.6 ist die schematisch Darstellung der dreidimensionalen Aktivitätswolke zu sehen. Sie wird durch den kleineren Quader in der Mitte repräsentiert. Es werden nicht die genauen Abmessungen der Wolke beschrieben. Vielmehr wir um sie ein Quader gelegt, in den sie genau paßt. Die detaillierte Form der Wolke ist somit nicht zu erkennen. Hierdurch ist es möglich einzelne Bewegungsabläufe der Wolke zu verfolgen. Bei einer detaillierteren Darstellung würde der Aufbau der Grafik zu lange dauern.

Der größere Quader repräsentiert den Raum um den Roboter mit der Kamera. In diesen Bereich sollte die Aktivitätswolke nicht kommen, da es sonst zu einem Zusammenprall zwi-

schen dem Objekt, das durch die Aktivitätswolke beschrieben wird, und dem Roboter bzw. der Kamera kommen könnte. Hierauf wird in Kapitel 9.7.2 noch eingegangen.

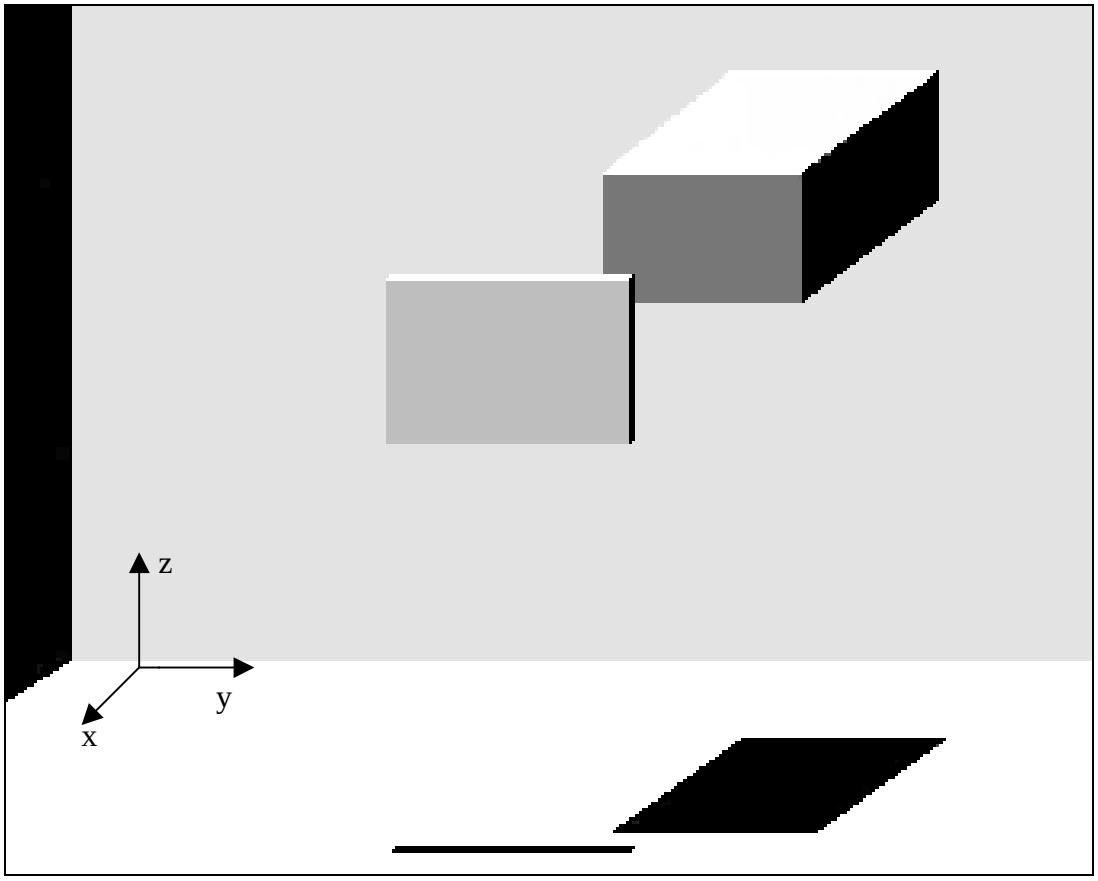


Fig. 9.6: Grafikoberfläche zur Projektion der dreidimensionalen Bewegung.

Um die aktuelle Position der beiden Quader in z-Richtung zu erkennen, wird unter ihnen jeweils ihr Schatten dargestellt. Hierdurch kann gesehen werden, wie weit die Aktivitätswolke vom Roboter entfernt ist.

Die vorgestellte Grafik dient dazu, „online“ Bewegungen zu verfolgen. Bewegungen der Kamera, die von einem Robotersimulationsprogramm erzeugt werden (siehe Kap. 10.1), werden also in Impulse für die Ansteuerung der Aktivitätswolke umgewandelt. In der Grafik wird dann die Aktivitätswolke entsprechend verschoben. Außerdem können Bewegungsabläufe, die in einer Datei abgelegt wurden, nachvollzogen werden. So können einzelne Sequenzen einer Bewegung verfolgt werden.

Als weiteres ist es noch möglich, gezielt einzelne Zustände zu laden, um sie dann detailliert analysieren zu können. Hierzu reicht die vorgestellt Grafik natürlich allein nicht aus. Deshalb kann zu einer Grafik mit Detailinformation über die Aktivitätswolke umgeschaltet werden (Fig. 9.7).

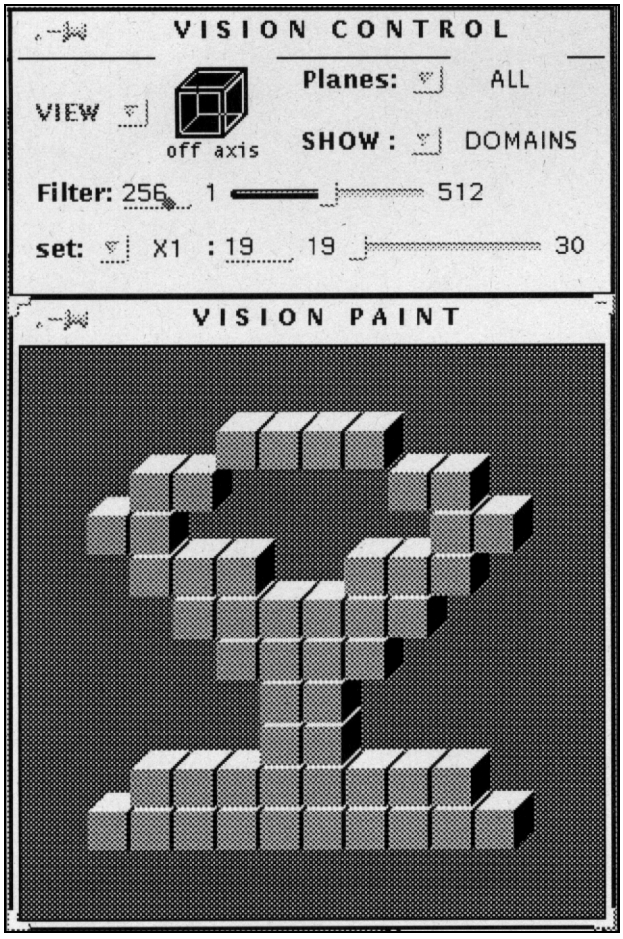


Fig. 9.7: Detailansicht der dreidimensionalen Aktivitätswolke.

In der Detailansicht kann die Aktivitätswolke in der Vorder-, Seiten- und Draufsicht betrachtet werden. Außerdem ist eine dreidimensionalen Ansicht wie in Fig. 9.7 einstellbar. Hierzu dient die Schaltfläche „VIEW“.

In der Figur ist die Aktivitätswolke in der Aufteilung nach Domänen dargestellt. Also wird jeweils eine ganze Domäne als kleiner Quader dargestellt, wenn eine gewisse Anzahl an Neuronen in der Domäne aktiv sind. Die Zahl der Neuronen, die aktiv seien müssen, ist über einen „Filter“ (Schwellwert) einstellbar. In dem Beispiel steht der Filter auf 256. Dies bedeutet, sind mehr als 256 Neuronen der Domäne aktiv, wird die Domäne gezeichnet. Ansonsten wird die Domäne ausgeblendet. Durch den Filter ist es möglich,, unterschiedliche Schwellen auszuwählen und zu testen. Die Praxis hat gezeigt, daß ein Filter von 50%, also 256 aktive Neuronen, praktikabel ist.

Um das genaue Aussehen der Aktivitätswolke zu betrachten, kann zu einer Darstellung gewechselt werden, in der alle aktiven Neuronen gezeichnet werden (Schaltmöglichkeit „SHOW“). Bei dieser Darstellung ist der Grafikaufbau sehr langsam. Deshalb ist es nur bei sehr detaillierten Analyse sinnvoll, diese Betrachtungsweise zu wählen. Fig. 9.8 zeigt ein entsprechendes Beispiel.

Besonders bei umfangreicheren Aktivitätswolken kann es sinnvoll sein, die Aktivitätswolke nur teilweise zu betrachten.. Über die Auswahl „Planes“ können neben der gesamten Wolke („ALL“) auch nur einzelne Scheiben der Wolke betrachtet werden. Eine Scheibe ist hierbei eine Reihe von Domänen. Über die Auswahl „set“ und den angrenzenden Slider kommen die Richtungen (x, y oder z) und die Grenzen der Scheiben festgelegt werden.

9.5 Ergebnisse

In den Simulationen hat sich gezeigt, daß mit dem erstellten System Bewegungen in beliebiger Richtung durchgeführt werden können. Für die translatorischen Bewegungen ist dies nicht weiter verwunderlich, da die Wolke als Block verschoben wird. Interessanter ist es, daß die Wolke auch bei zusammengesetzten Bewegungen kompakt bleibt. Die nachfolgenden Grafiken zeigen dies.

In Fig. 9.8 ist links das Ausgangsobjekt zu sehen. Jedes aktive Neuron der Aktivitätswolke wird durch einen Punkt abgebildet. Es ist zu sehen, daß das Objekt nach einer Drehung um 180° in seiner Grundstruktur erhalten bleibt. Am Rand kommt es zu kleinen Ungenauigkeiten. Durch laterale Kopplungen können diese in einem tolerierbaren Rahmen gehalten werden (vgl. Kap. 9.6)

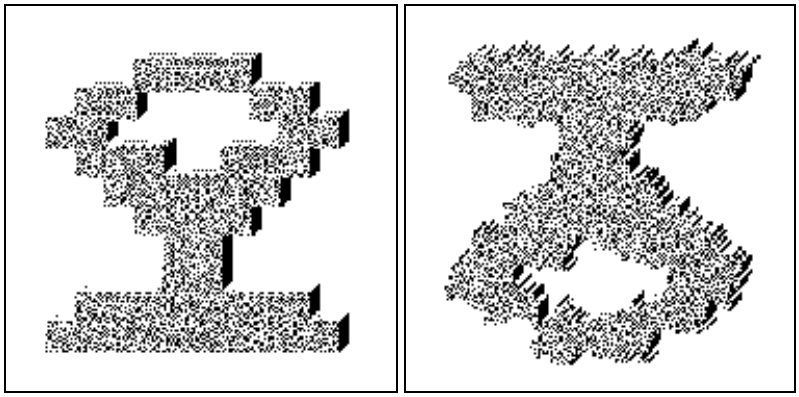


Fig. 9.8: Darstellung aller aktiven Neuronen einer Aktivitätswolke.

Das gleiche Objekt ist nochmals in Fig. 9.9 dargestellt. Nur diesmal sind die Domänen, die aktive Neuronen besitzen, als Quader dargestellt. Eine Domäne wird gezeichnet, wenn mehr als die Hälfte alle Neuronen der Domäne aktiv sind. Zu dem Aktivitätsfenster gehören außerdem noch weitere Domänen, in denen keine Neuronen aktiv sind. Insgesamt ist es 12 x 12 x 3 = 432 Domänen groß. Im rechten Bild ist das Objekt nach einer Drehung von 180° um die z-Achse zu sehen.

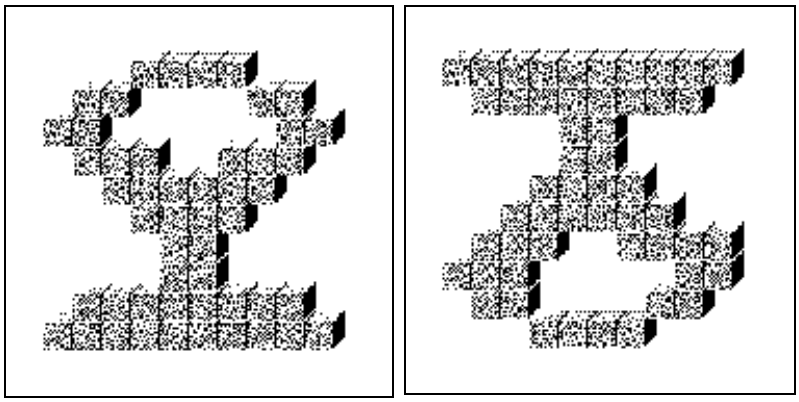


Fig. 9.9: Darstellung einer Aktivitätswolke für ein Objekt.

Dies zeigt, daß bei der Auswertung der Aktivität im RRN nicht jedes Neuron einzeln auszuwerten ist. Vielmehr werden die Domänen als ganzes betrachtet.

9.6 Laterale Kopplung

Wie auch schon im zweidimensionalen Fall bleibt die Aktivitätswolke bei zusammengesetzten Bewegungen nicht kompakt. Es bilden sich isolierte aktive und passive Neuronen. Deshalb koppeln sich Nachbarneuronen gegenseitig, um isolierte aktive Neuronen zu deaktivieren bzw. isolierte passive Neuronen zu aktivieren.

Es wurde untersucht, wie stark sich die Nachbarneuronen beeinflussen müssen, damit die ursprüngliche Form des Objekts erhalten bleibt. Zur Auswertung wird die Aktivität der 26 Nach-barneuronen herangezogen. Gekoppelt wird nach jedem Shiftimpuls. Als Bewertungskriterien, wie gut die Form der Startaktivitätswolke erhalten bleibt, wurden folgende Kriterien untersucht:

- Formerhaltung der Aktivitätswolke,
- Zusammenhalt der Aktivitätswolke,
- Vorhandensein von isolierten aktiven Neuronen,
- Vorhandensein von isolierten passiven Neuronen und

- Erhaltung der Ränder des Objekts.

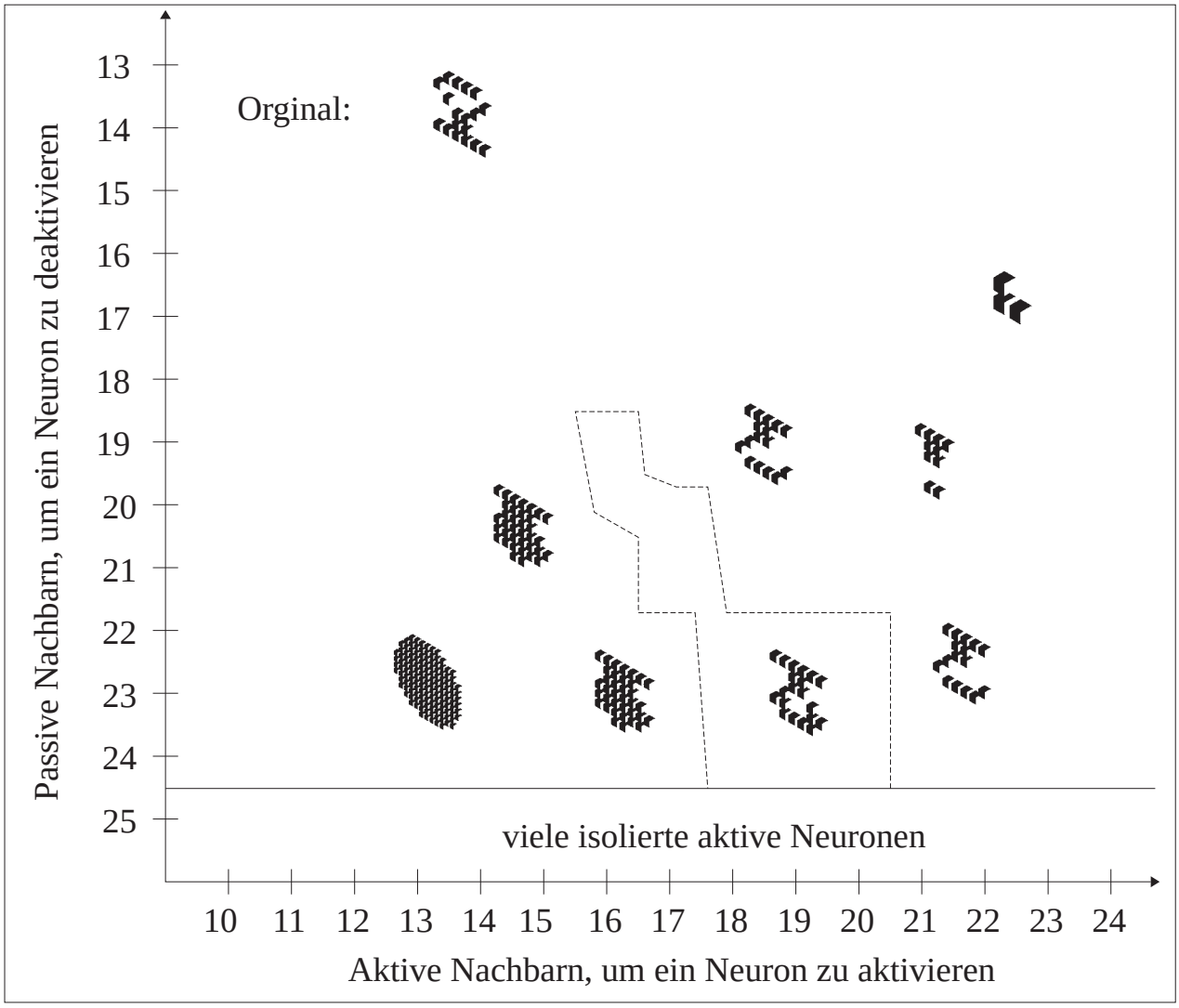


Fig. 9.10: Je nach Stärke der lateralen Kopplung wird das Originalobjekt (linke obere Ecke) in seiner Form verändert.

In Fig. 9.10 sind die Ergebnisse für die Aktivitätswolke einer Seitenplatte, ein Teil des Crandfieldmontagesatzes, zu sehen. Die Startaktivitätswolke ist in der linken oberen Ecke dargestellt. Eine Domäne wird jeweils dargestellt, wenn mehr als die Hälfte ihrer Neuronen aktiv ist. Es ist der Zustand der Aktivitätswolke in Abhängigkeit von der Anzahl der aktiven und passiven Nachbarneuronen, die die Kopplung beeinflussen, dargestellt. Auf der x-Achse ist die Anzahl der Nachbarneuronen aufgetragen, die aktiv sein müssen, um ein passives Neuron

zu aktivieren, und auf der y-Achse ist die Anzahl der Nachbarneuronen dargestellt, die passiv seien müssen, um ein aktives Neuron zu deaktivieren.

Insgesamt ergibt sich ein Bereich, in dem die Seitenplatte in ihrer Grundform erhalten bleibt (innerhalb der gestrichelten Linie). Ist die Kopplung zu stark eingestellt, daß heißt, reichen wenige aktive Nachbarneuronen aus, um ein passives Nachbarneuron zu aktivieren bzw. wird ein Neuron erst deaktiviert, wenn sehr viele Nachbarneuronen passiv sind, vergrößert sich die Aktivitätswolke (linker Teil der Grafik). Im anderen Fall verkleinert sich die Aktivitätswolke, und sie reißt am Loch der Seitenplatte auf (rechter Teil der Grafik).

9.7 Anwendungsbeispiele

Bisher wurde nur davon gesprochen, wie einzelne Objekte im RRN repräsentiert werden. Nun werden zwei Beispiele skizziert, die mögliche Einsatzgebiete aufzeigen.

9.7.1 Beispiel 1: Suchen eines Objektes im Raum

Sobald die Umgebung der Kamera ins RRN aufgenommen wird, kann durch Aktivitätsabfragen bei allen Gitterneuronen des RRNs festgestellt werden, wo sich die einzelnen Objekte nach jeder beliebigen Bewegung befinden. Um nun ein Objekt aufzufinden, ist es nicht notwendig, den gesamten Raum um die Kamera zu untersuchen, sondern die Kamera kann gezielt zu dem Objekt bewegt werden. Erst wenn das Objekt wieder vor der Kamera ist, muß es erneut aufgenommen werden, um seine genaue Struktur festzustellen. Durch diese Vorgehensweise entfällt das zeitaufwendige Suchen nach Objekten im Raum.

9.7.2 Beispiel 2: Kollisionsvermeidung

Darüber hinaus kann das RRN zur Kollisionsvermeidung eingesetzt werden. Da sich die Kamera immer in der Mitte des Koordinatensystems befindet, ist sie immer von den gleichen Neuronen umgeben. Es braucht nur die Aktivität dieser Neuronen abgegriffen werden, um festzustellen, ob sich die Kamera sehr nah an ein Objekt heran bewegt.

Bewegt sich die Kamera auf ein Objekt zu, schiebt sich die Aktivitätswolke immer näher an die Neuronen um die Kamera. Wird nun beispielsweise die Aktivität von zwei Domänenrei-

hen um die Kamera kontrolliert, wird ein Teil dieser Neuronen aktiv, wenn sich die Kamera sehr dicht an das Objekt bewegt hat.

Da es auch sinnvoll sein kann, die Kamera ganz dicht an ein Objekt zu fahren, darf das System an dieser Stelle nicht gestoppt werden. Vielmehr muß von einer übergeordneten Einheit entschieden werden, wie weiter zu reagieren ist. Generell ist es auf diese Weise möglich, die Kollision der Kamera mit einem Objekt zu vermeiden.

Zu Testzwecken wurde ein entsprechender Mechanismus in das RRN eingebaut. Um die Kamera wird ein Schutzraum gebildet, in den die Aktivitätswolke nicht eindringen darf. Berührt die Wolke den Schutzraum, wird eine Warnung ausgegeben. Eine Rückkopplung auf das Robotersystem wurde aber nicht implementiert, da hierzu nicht hinreichend in die Robotersteuerung eingegriffen werden konnte.

10. Kopplung des RRNs mit einem Roboter

Um das RRN mit der Bewegungsinformation einer Kamera zu versorgen, wurde ein sechsachsiger Roboter mit dem RRN gekoppelt, an dessen sechster Achse eine Kamera befestigt ist. Für den zweidimensionalen Fall führt der Roboter Bewegungen in x- und y-Richtung und Drehungen um die z-Achse der Kamera durch. Die Objekte liegen auf einem Tisch, und die Kamera wird in einer festen Höhe über den Tisch geführt. Befindet sich die Kamera über einem der Objekte, dann wird im RRN eine entsprechende Aktivitätswolke erzeugt. Bei allen Bewegungen wird die erzeugte Aktivitätswolke in der entgegengesetzten Richtung verschoben.

Für den dreidimensionalen Fall wurde keine direkte Kopplung des Roboters mit dem RRN erstellt. Vielmehr beschränkte man sich darauf, das RRN durch ein Simulationsprogramm anzusteuern (Fig. 10.1). Im folgenden soll zuerst die Schnittstelle für den zweidimensionalen Fall vorgestellt werden, da dieser sich übersichtlicher darstellen läßt. Anschließend werden die Erweiterungen für die dreidimensionale Bewegung aufgezeigt.

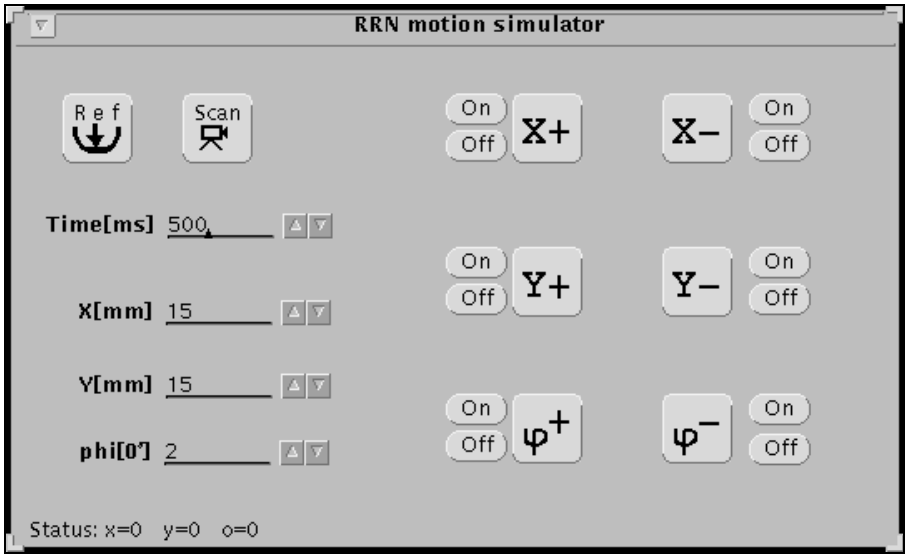


Fig. 10.1: Das Eingabefenster des Robotersimulationsprogramms.

10.1 Schnittstelle zwischen Roboter und RRN

Die Kopplung des RRN mit dem Roboter wird über eine "Remote Procedure Call"-Schnittstelle (RPC) realisiert. So ist es möglich, beide Programme unabhängig voneinander auf verschiedenen Rechnern laufen zu lassen. Dies ist besonders wichtig, denn das RRN benötigt eine hohe Rechenleistung.

Da das RRN seine Bewegungsinformation direkt vom Roboter der Dortmunder Forschungsgruppe erhalten soll und aus der Paderborner Robotersteuerung nicht die entsprechenden Impulse gewonnen werden konnten, wurde das Robotersimulationsprogramm *ROB* geschrieben. Dieses Programm steuert gleichzeitig den Roboter und das RRN an. Da die Ansteuerung des Roboters nicht Gegenstand der vorliegenden Arbeit war, wird hierauf nicht weiter eingegangen.

In Fig. 10.2 ist die Schnittstelle zwischen dem ROB und dem RRN zu sehen. Das Programm ROB dient als Client, während das RRN als Server fungiert. Als erstes kann das gesamte System neu initialisiert werden, d.h. der Roboter wird in seine Ausgangsstellung bewegt, und im RRN wird alle Aktivität gelöscht.

Nimmt nun die Kamera ein Bild auf, wird hieraus ein Binärbild der Größe 64 x 64 erzeugt und an das RRN gesendet. Dort werden dann die Neuronen unter der Kamera entsprechend aktiviert. Jedes Neuron ist einem Binärwert des Bildes zugeordnet. Neuronen, deren Binärwert eins ist, werden aktiv, alle anderen bleiben passiv. Somit ist die Aktivitätswolke für das Objekt erzeugt. Soll später ein zweites Objekt aufgenommen werden, muß die Kamera sich erst über das Objekt bewegen. Hierbei schiebt sich die Aktivitätswolke im RRN unter der Kamera weg, und bei der Aufnahme des neuen Objektes wird eine weitere Aktivitätswolke erzeugt.

Bei jeder Bewegung des Roboters wird die Roboterbewegung in die Bewegungskomponenten V^+_x, V^-_x, V^+_y, V^-_y, Ω^+ und Ω^- umgerechnet. Die Anteile dieser Komponenten werden an das RRN übergeben. Dort werden entsprechend der Beschreibung in Kapitel 8.1 und 8.2 die Spikeraten für die einzelnen Shiftneuronen bestimmt und die Aktivitätswolke dann verschoben.

Normalerweise arbeitet die RPC-Schnittstelle so, daß der Server auf eine Anweisung vom Client wartet und während dieser Wartezeit nicht beschäftigt ist. Erst wenn der Client einen RPC-Aufruf startet, wird der Server aktiviert und arbeitet seinen Programmteil ab. Anschließend sendet er ein Fertigsignal an den Client. Dieser hat in der Zwischenzeit gewartet, bis er das Signal erhält, und arbeitet dann weiter. Dies bedeutet, daß immer nur ein Prozeß aktiv ist.

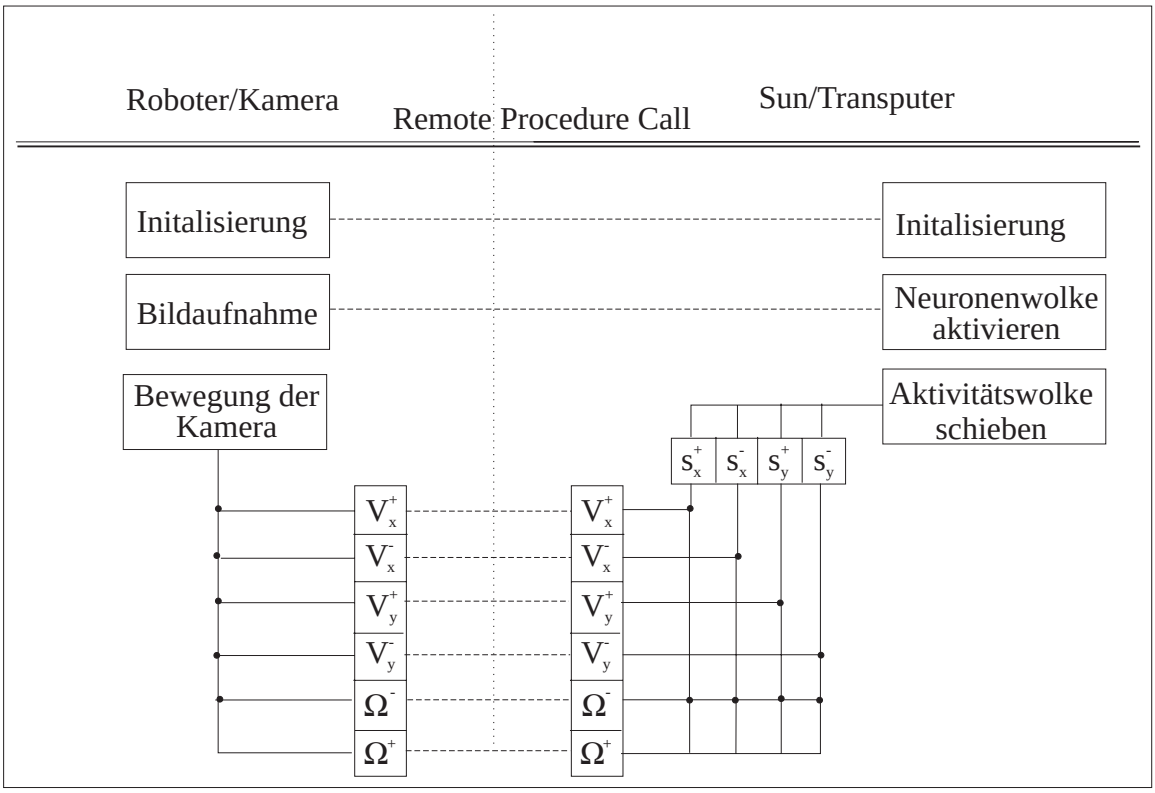


Fig. 10.2: Die Schnittstelle zwischen Roboter und Kamera wird über einen "Remote Procedure Call" realisiert.

Diese Arbeitsweise führt aber dazu, daß sich die Programme gegenseitig behindern. Der Roboter kann sich nur bewegen, wenn das RRN passiv ist und umgekehrt. Ziel war es jedoch, beide Programme parallel nebeneinander laufen zu lassen. Deshalb wurde die ursprüngliche Schnittstelle überarbeitet, ohne aber die Parameter der Schnittstelle zu verändern (Fig. 10.3).

Wichtig bei der Kommunikation ist, daß die gesendete Information schnell abgenommen wird. Die bisherige Schwierigkeit lag darin, daß die Information sofort im RRN verarbeitet wird. Um diese zu lösen, wurde ein Zwischenspeicher eingebaut. Jede ankommende Information vom Robotersimulationsprogramm wird nun erst in den Speicher geschrieben, und danach wird sofort ein Fertigsignal an den Client zurückgesendet, so daß dieser weiterarbeiten kann.

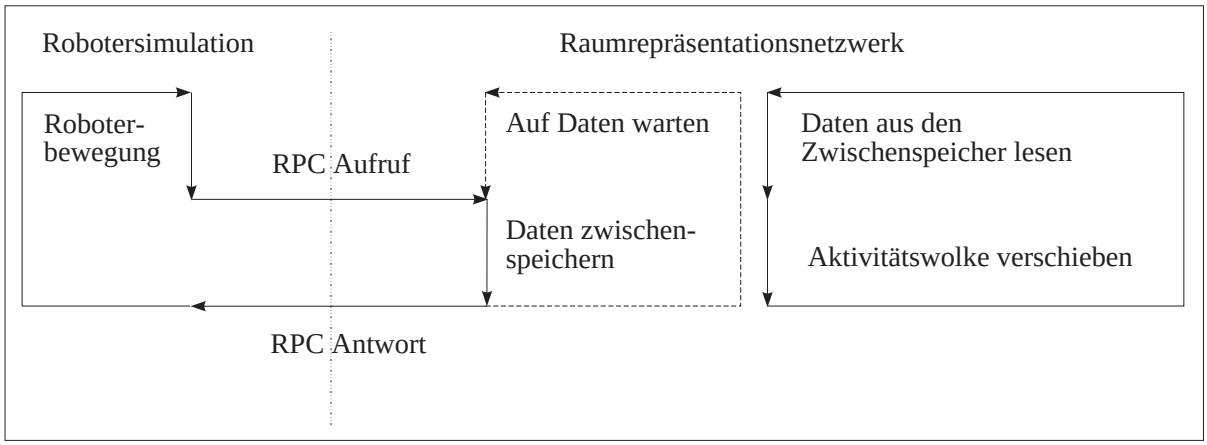


Fig. 10.3: Die Daten von der Robotersimulation werden im RRN erst in einen Zwischenspeicher geschrieben.

Das RRN überprüft kontinuierlich, ob im Zwischenspeicher Informationen vorhanden sind. Ist dies der Fall, wird die entsprechende Aktion ausgeführt. In der Zwischenzeit kann die Robotersimulation schon weitere Informationen im Speicher ablegen, ohne daß das RRN beeinflußt wird.

10.2 Erweiterung auf das dreidimensionale System

Um die Schnittstelle auf die dreidimensionale Bewegung zu erweitern, bleiben die beschriebenen Funktionen gleich. Bei der Bildübertragung wird nun ein Binärbild der Größe 64 x 64 x 64 übertragen und die entsprechenden Neuronen aktiviert. Es kommen die zusätzlichen Bewegungsanteile der dreidimensionalen Bewegung hinzu, die in einer entsprechenden Form die sechs Shiftneuronen für jede Domäne ansteuern.

11. Exploration des Raums um eine Kamera

Bisher wurde die Aktivitätswolke der Objekte im RRN nur statisch gesetzt. Im zweidimensionalen Fall geschieht dies durch eine Kamera, die in einer konstanten Entfernung vom Objekt plaziert ist. Die Fläche des Objekts wird als Binärbild an das RRN gesendet, wo dann die entsprechende Aktivitätswolke initialisiert wird, indem die Neuronen, die sich unter der Kamera befinden, aktiviert werden. Für den dreidimensionalen Fall wurden die Aktivitätswolken am Anfang nur statisch gesetzt.

Das Ziel ist es jedoch, den gesamten Raum um die Kamera zu erkunden und beliebige Objekte in beliebiger Lage erfassen zu können. Die Form eines Objekts soll durch Anschauen aus verschiedenen Blickwinkeln ermittelt werden (Stereo by motion). Für die Untersuchungen wurde das zweidimensionale Polarkoordinatensystem gewählt. Eine Erweiterung ins Dreidimensionale ist leicht möglich, indem das Objekt nicht nur von links und rechts betrachtet wird, sondern auch von oben und unten. Deshalb reicht eine Diskussion im Zweidimensionalen aus. Außerdem wird aus Übersichtsgründen nur ein Objekt betrachtet.

Der Raum um die Kamera wird nun in zwei Bereich eingeteilt. Der Bereich vor der Kamera, in dem die Kamera Bilder aufnehmen kann, wird als *Erfassungsbereich* bezeichnet. Nur hier wird ein Objekt erfaßt und kann somit die Aktivitätswolke verändern. Der andere Bereich ist die Fläche, die nicht von der Kamera eingesehen werden kann und folgerichtig mit *außerhalb des Erfassungsbereich* bezeichnet wird.

Bewegt sich die Kamera, aktiviert ein Objekt, das in dem Erfassungsbereich liegt oder kommt, einen *Neuronenstrahl*. Als solcher wird eine Reihe von Neuronen bezeichnet, die ausgehend vom Zentrum bis zum äußeren Rand des RRNs reicht. Ein Neuronenstrahl wird deshalb aktiviert, da die Kamera keinerlei Tiefeninformation über das Objekt besitzt. Erst durch Bewegung der Kamera wird die Information über die Tiefe des Objekts gewonnen.

11.1 Regeln zur Exploration des Raumes

Um den Raum zu erkunden, wurden mehrere Regeln aufgestellt, inwieweit das ursprüngliche RRN zu modifizieren ist, damit zu einem Objekt im Raum die zugehörige Aktivitätswolke erzeugt wird. Die Regeln lauten wie folgt:

Bewegung der Aktivitätswolke **außerhalb** des Erfassungsbereich:

- Verschiebung wie bisher

Bewegung der Aktivitätswolke **innerhalb** des Erfassungsbereiches:
1) (Objekt im Domänenstrahl) & (Domänenstrahl aktiv):
   - Verschiebung wie bisher
2) Objekt nicht im Domänenstrahl:
   - keine Aktivierung von Neuronen am Anfang der Wolke
   - keine Rückkopplung der Neuronen
3) (Objekt im Domänenstrahl) & (Domänenstrahl nicht aktiv):
   - Neuronenstrahl aktivieren
4) (Objekt im Nachbarneuronenstrahl) & (entsprechender Domänenstrahl nicht aktiv):
   - keine Hemmung von Neuronen am Ende der Wolke

Es wird zwischen Regeln für die Verschiebung der Wolke außerhalb und innerhalb des Erfassungsbereiches unterschieden. Außerhalb des Erfassungsbereiches wird die Aktivitätswolke wie bisher verschoben, da hier kein sensorischer Input durch die Kamera erfolgt.

Nur innerhalb des Erfassungsbereiches wird die Verschiebung durch den sensorischen Input beeinflußt. Die Betrachtung, wie die Wolke innerhalb einer Domäne verschoben wird, geschieht über den jeweiligen *Domänenstrahl*. Ein Domänenstrahl besteht aus einer Reihe von Domänen vom Ursprung bis zum Rand des betrachteten Raums.

Für alle Domänen, die zu einem Domänenstrahl gehören, in dem ein Objekt und die Aktivitätswolke liegt, gilt, daß sich die Aktivitätswolke in der richtigen Position befindet (Regel 1). Somit wird die Wolke in der ursprünglichen Form verschoben.

Auf der anderen Seite gilt für alle Domänen, die zu einem Domänenstrahl gehören, in dem kein Objekt vorhanden ist, daß dort auch keine Aktivitätswolke sein darf (Regel 2). Somit müssen die Neuronen dieser Domänen deaktiviert werden. Dies geschieht dadurch, daß bei der Verschiebung der Wolke keine neuen Neuronen am Anfang der Wolke aktiviert werden und die Rückkopplung der Neuronen auf sich selbst unterdrückt wird.

Bisher wurde immer davon ausgegangen, daß bereits eine Aktivitätswolke vorhanden ist. Ist nun aber in einem Domänenstrahl ein Objekt, aber noch kein aktives Neuron, werden die Neuronen eines Neuronenstrahls durch den sensorischen Input aktiviert (Regel 3). Somit ist dieser Domänenstrahl aktiv. Damit aber mehr als ein Neuronenstrahl aktiviert wird, um eine Aktivitätswolke zu erzeugen, muß Regel 4 angewandt werden. Hier wird der Nachbarneuronenstrahl betrachtet, wobei bei einer Bewegung der Wolke in φ^+ -Richtung die Nachbarneuronen der Domäne in φ^- -Richtung beobachtet werden. Ist in diesem Neuronenstrahl das Objekt plaziert, der Domänenstrahl jedoch nicht aktiv, heißt dies, daß die Aktivitätswolke

noch nicht breit genug ist und somit die Hemmung der Neuronen am Ende der Wolke unterdrückt wird.

Mit diesen Regeln kann durch Anschauen des Objekts von mehreren Seiten die Aktivitätswolke für ein Objekt erstellt werden. Betrachtet man mehrere Objekte, gelten die gleichen Regeln. Es müssen nur mehr Bewegungen um die Objekte erfolgen.

11.2 Modifizierung der neuronalen Verschaltung

Es lassen sich viele neuronale Verschaltungen finden, um das RRN so zu modifizieren, daß es nach den oben genannten Regeln arbeitet. Ziel war es aber, mit möglichst wenigen zusätzlichen Verbindungen auszukommen.

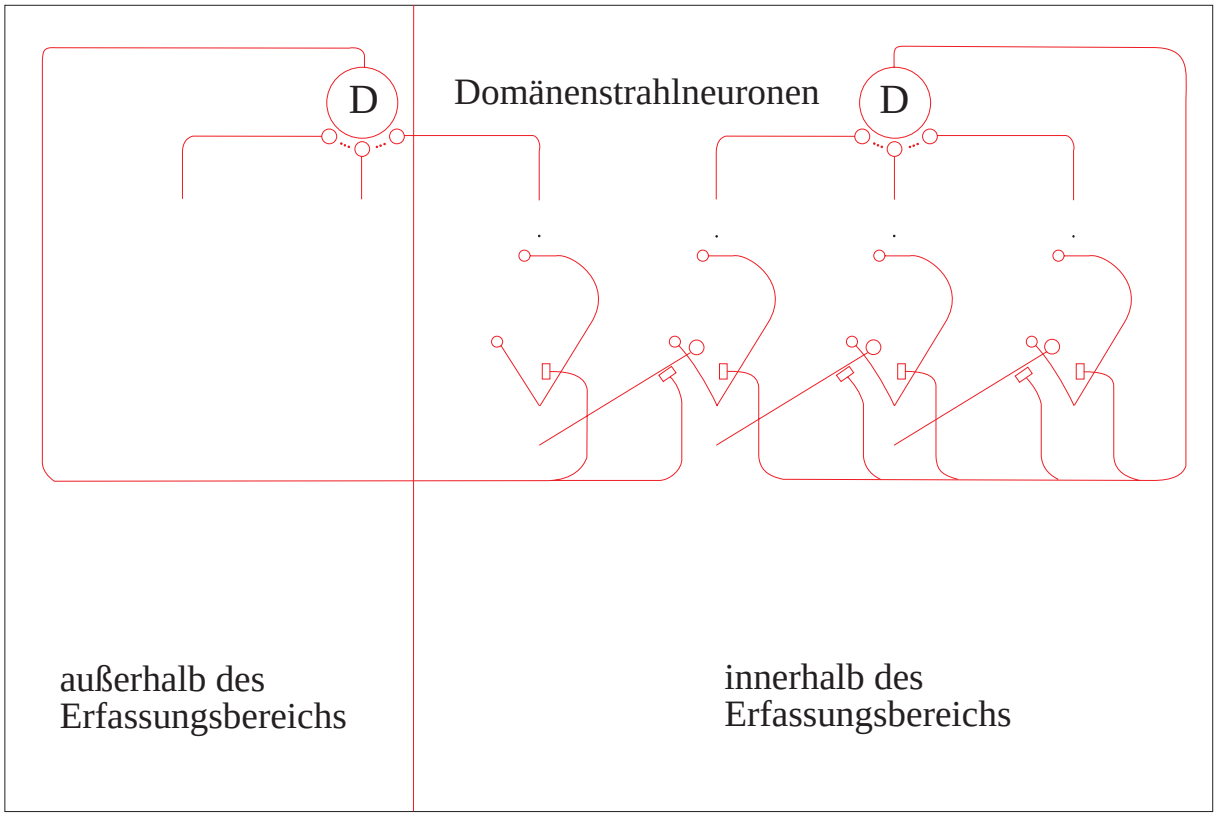


Fig. 11.1: Zusätzlichen Neuronen und neuronale Verbindungen zur Raumerkundung.

In Figur 11.1 ist zu sehen, wie die Modifikation des RRN mit wenigen zusätzlichen Neuronen und neuronalen Verbindungen möglich ist. Wie bereits erwähnt, ist es nötig, das Netzwerk in einen Bereich außerhalb des Erfassungsbereiches und einen innerhalb des Erfassungsbereiches aufzuteilen. Nur innerhalb des Erfassungsbereiches sind zusätzliche Verbindungen notwendig.

Außerdem werden zusätzliche Neuronen (Neuronen D in Fig. 11.1) eingeführt, die ihren Input von den Neuronen eines Domänenstrahls erhalten. Ist die Aktivitätswolke innerhalb des Domänenstrahls, sind die D-Neuronen aktiv.

Außerhalb des Erfassungsbereiches bekommen die Neuronen keinen sensorischen Input mehr, deshalb werden die entsprechenden Verbindungen nicht mehr benötigt (Fig. 11.2 links). Wird ein Neuron durch einen sensorischen Input aktiviert, erregt es zusammen mit den anderen Neuronen des Domänenstrahls sein Domänenstrahlneuron. Das Domänenstrahlneuron wird seinerseits aktiv und hemmt über eine präsynaptische Verbindung den sensorischen Input. Somit können durch den Input keine weiteren Neuronen aktiviert werden (Regel 3).

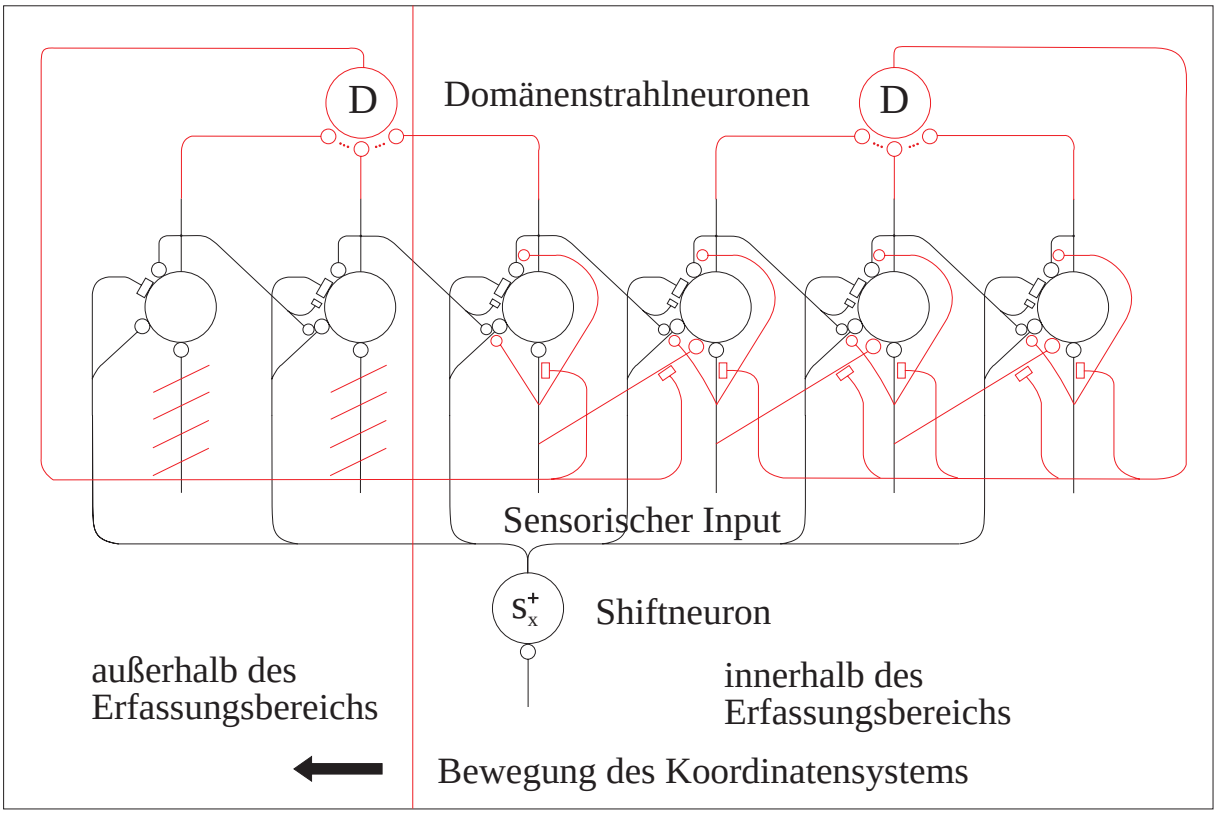


Fig. 11.2: Die komplette neuronale Verschaltung für die Bewegung der Aktivitätswolke in (x^+)-Richtung.

Darüber hinaus wirkt der sensorische Input auf die erregende Verbindung vom linken Nachbarneuron und auf die Rückkopplung. Ist der sensorische Input aktiv, beeinflussen diese zusätzlichen Synapsen die Verschiebung der Aktivität nicht (Regel 1). Sobald aber kein sensorischer Input an den entsprechenden Koordinaten vorhanden ist, hemmt die untere Synapse, die von dem Shiftneuron kommt, und das Neuron kann nicht aktiviert werden; am Anfang der Wolke wird daher keine Aktivität von einem Neuron auf das nächste übertragen.

Die Synapse, die auf der Rückkopplungssynapse endet, unterdrückt die Rückkopplung, so daß das Neuron seine Aktivität nicht mehr erhalten kann und passiv wird (Regel 2). Zuletzt gibt es eine zusätzliche Verbindung vom sensorischen Input auf das rechte Nachbarneuron. Das synaptische Gewicht ist nicht so hoch, daß es ein passives Neuron aktivieren kann. Es ist aber groß genug, um die Hemmung durch den inhibitorischen Input zu kompensieren. So bleibt das Nachbarneuron weiterhin aktiv, und die Wolke verliert an ihrem Ende keine Aktivität (Regel 4).

Die vorgestellten synaptischen Verbindungen realisieren die Regeln zur Erzeugung einer Aktivitätswolke für die Bewegung des Koordinatensystems nach x^+. Durch die spiegelbildlichen Verbindungen lassen sich die Regeln für eine x^- -Bewegung anwenden. Für die Verschiebung der Aktivität in r-Richtung werden keine zusätzlichen Verbindungen gebraucht, da die Aktivität immer auf einem Neuronenstrahl entlang läuft und dort der sensorische Input für alle Neuronen gleich ist. Erweitert man das System auf den dreidimensionalen Fall, ist der Erfassungsbereich nach oben und unten auszudehnen. In diesem Bereich gelten dann wieder die aufgestellten Regeln, die durch eine entsprechende spiegelbildliche Verschaltung zu realisieren sind.

11.3 Implementierung

Um die beschriebene Methode zur Exploration des Raums zu testen, wurde sie in das Programmpaket für das zweidimensionale RRN im Polarkoordinatensystem integriert. Als zusätzliches Modul wurde der Standort des Objekts mit in das System aufgenommen. In dieser Version wurde davon ausgegangen, daß sich nur ein Objekt im Raum befindet.

Vor der Kamera wird der Erfassungsbereich definiert (Fig. 11.3). Da am Anfang noch keine Aktivitätswolke vorhanden ist, müßte das Aktivitätsfenster den gesamten Raum umfassen. Deshalb wird auf das Aktivitätsfenster verzichtet. Alle anderen Programmteile wurden aus dem vorgegebenen Programm übernommen.

Für alle Domänen außerhalb des Erfassungsbereiches ändert sich nichts. Dort wird der normale Algorithmus angewandt. Nur für jede aktive Domäne im Erfassungsbereich wird eine Betrachtung nach den Regeln erforderlich.

11.4 Ergebnisse

Mit diesem Netzwerk können Objekte als Aktivitätswolke in das RRN aufgenommen werden. In Fig. 11.3 ist die Ausgangssituation zu sehen. Die Kamera, die durch einen Kopf dargestellt

wird, hat einen Erfassungsbereich, in dem sie Objekte erfassen kann. Innerhalb des Erfassungsbereiches befindet sich kein Objekt und somit auch noch keine Aktivitätswolke. Links neben der Kamera befindet sich ein Objekt.

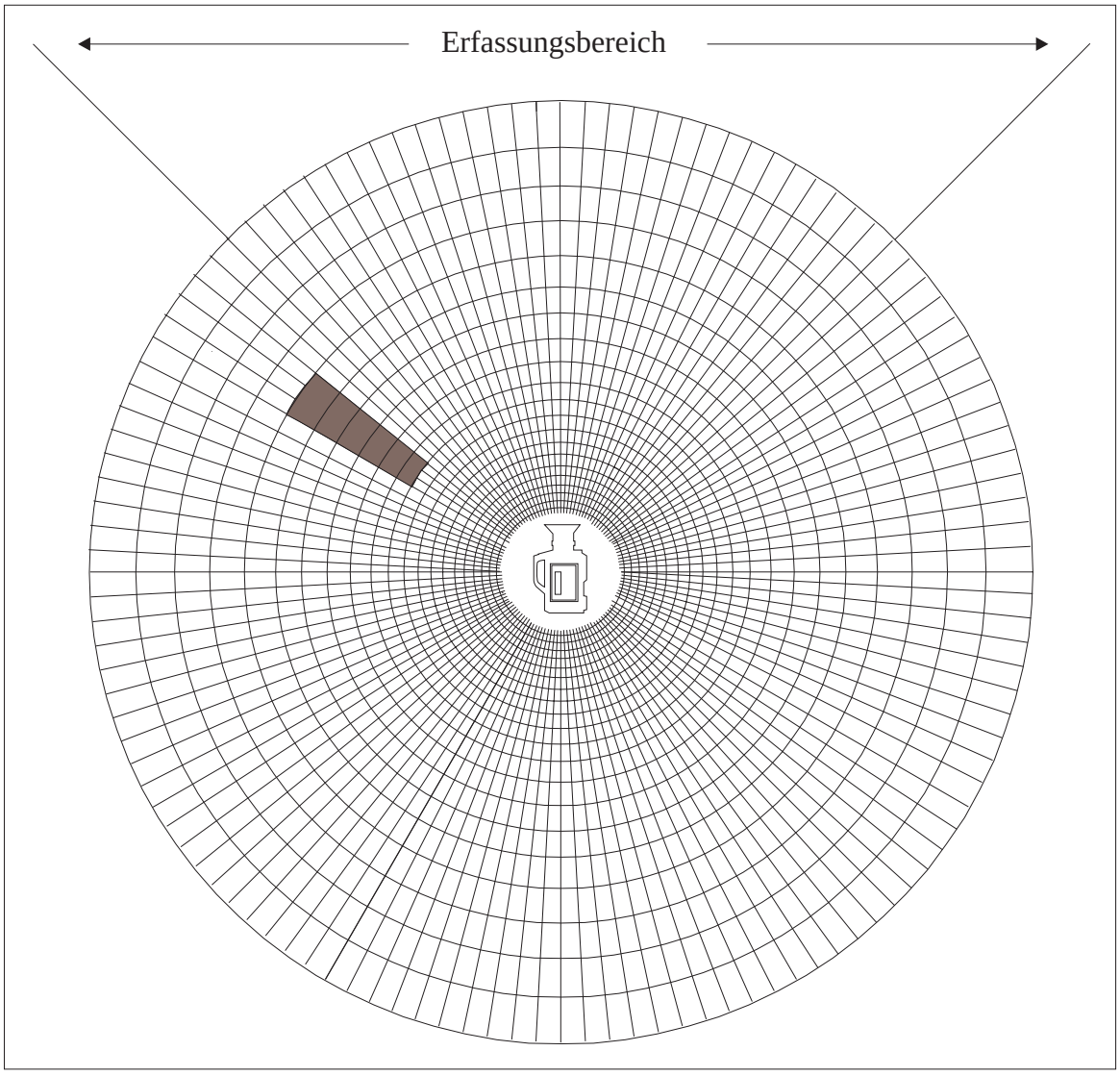


Fig. 11.3: Im Anfangszustand existiert im Netzwerk keine Aktivitätswolke. Außerhalb des Erfassungsbereiches befindet sich ein Objekt (schwarz).

Bewegt sich nun die Kamera in Richtung des Objekts, gelangt das Objekt nach einiger Zeit in den Erfassungsbereich. Hierdurch werden die Neuronen des ersten Domänenstrahls am Rand des Erfassungsbereiches aktiviert (Fig. 11.4).

Dreht sich die Kamera nun weiter um die eigene Achse und rückt das Objekt weiter in den Erfassungsbereich, werden die Neuronen der Domänenstrahlen, in deren Bereich das Objekt gelangt, nach und nach aktiv, während die bereits aktiven Neuronen aktiv bleiben. Erst wenn

sich das Objekt vollständig im Erfassungsbereich befindet, werden die Neuronen in den Neuronenstrahlen, in denen kein Objekt mehr vorhanden ist, passiv. Die Aktivitätswolke besteht nun aus mehreren aktiven Domänenstrahlen. Überlagert man die aktuelle Position des Objektes, dann ist zu sehen, daß die Aktivitätswolke so breit wie das Objekt ist (Fig. 11.5).

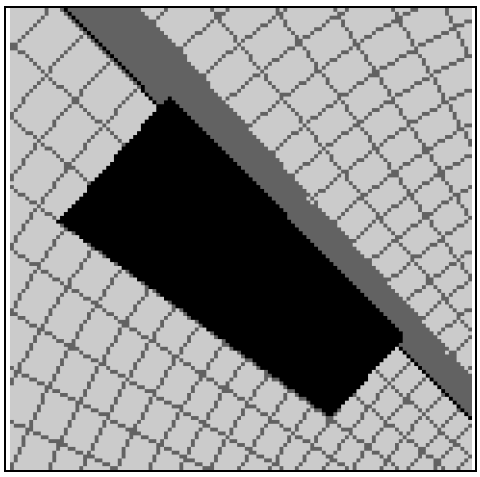


Fig. 11.4: Das Objekt (schwarz) wandert in den Erfassungsbereich.

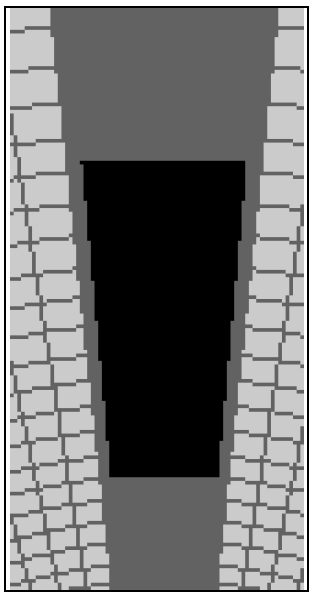


Fig. 11.5: Das Objekt (schwarz) ist vollständig in den Erfassungsbereich.

Bisher dehnt sich die Aktivitätswolke vom Zentrum bis zur Peripherie aus. Um das zu betrachtende Objekt genau zu erfassen, muß nun das Objekt von den Seiten angeschaut werden. Fig. 11.6 zeigt die Aktivitätswolke, nachdem das Objekt von der linken Seite betrachtet wurde. Die Aktivitätswolke ist fast auf die Größe des Objekts reduziert worden. Da jede Domäne mit mehr als 32 aktiven Neuronen dargestellt ist, ist die Aktivitätswolke etwas größer als das Objekt.

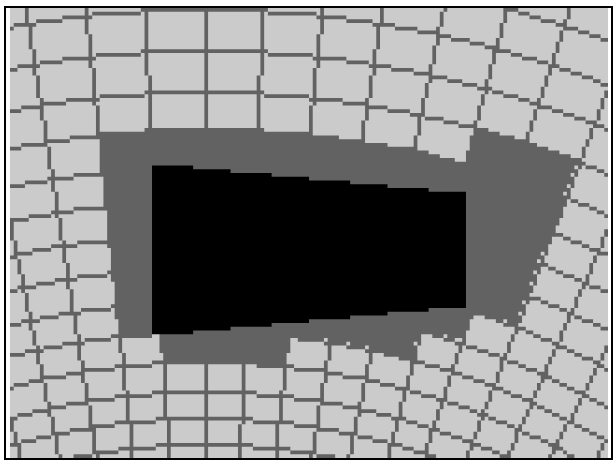


Fig. 11.6: Die Aktivitätswolke (dunkelgrau), nachdem das Objekt (schwarz) von mehreren Seiten angeschaut wurde.

Betrachtet man jedes Neuron der Aktivitätswolke (Fig. 11.7), ist zu sehen, daß die Aktivitätswolke sich dem Objekt gut angenähert hat. Nur auf der rechten Seite sind einige Neuronen zuviel aktiv, die durch Betrachtung dieser Seite noch deaktiviert werden können.

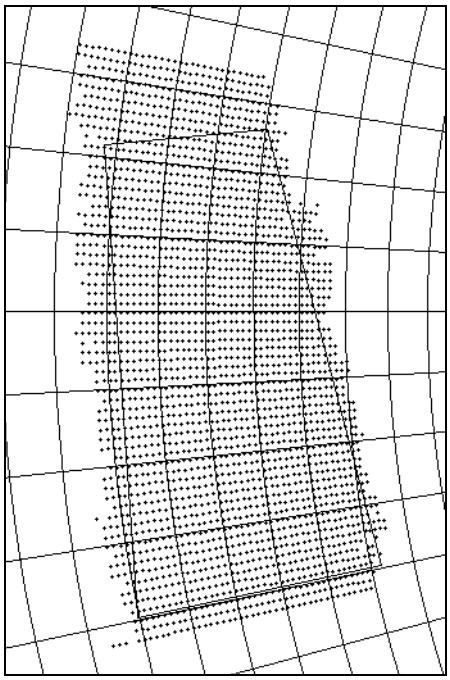


Fig. 11.7: Detaillierte Darstellung der resultierenden Aktivitätswolke.

12. Vergleich mit der Anatomie des Cerebellums

Bisher wurde die Repräsentation von Objekten durch eine Aktivitätswolke und die Verschiebung der Aktivität diskutiert. Nun soll die Anzahl der benötigten Neuronen abgeschätzt werden. Außerdem wird die entstandene dreidimensionale Struktur auf eine zweidimensionale Struktur abgebildet, um sie mit der Anatomie des Cerebellums zu vergleichen.

12.1 Einige Definitionen und Schlußfolgerungen

Wie schon in Kapitel 8.9 beschrieben, erscheint es plausibel, daß die neuronale Wolke in einem Polarkoordinatensystem in ebenen bzw. in einem Kugelkoordinatensystem im räumlichen Fall repräsentiert wird. Nahe Objekte werden nämlich mit einer großen Genauigkeit wahrgenommen, was sich durch eine höhere und dichtere Zahl von Neuronen beschreiben läßt. Weit entfernte Objekte werden hingegen nur mit geringerer Genauigkeit gesehen. Die Dichte der Neuronen braucht hier also nicht so groß zu sein. Es werden weniger Neuronen pro Volumenelement benötigt.

Aus den Simulationen ist bekannt, daß die sich bewegende Wolke am Rand zerfällt. Deshalb erscheint es nicht sinnvoll, ein Volumenelement durch ein einzelnes Neuron zu repräsentieren. Stattdessen bildet eine Gruppe von 1000 Neuronen ein Volumenelement. Es wird weiterhin festgelegt, daß eine Domäne im folgenden aus 10 x 10 x 10 Neuronen besteht.

Außerdem ist bekannt, daß pro Domäne nur je ein Satz an Shiftneuronen notwendig ist. Deshalb ist es auf der anderen Seite ebenfalls vernünftig, daß ein einzelnes Neuron die Aktivität der Gitterneuronen ausliest. Erst wenn genügend Gitterneuronen aktiv sind, wird das entsprechende auslesende Neuron, im folgendem mit *Leseneuron* bezeichnet, aktiv. Daher können kleine Fehler toleriert werden.

Eine Anwendung der neuronalen Wolke in einem biologischen System ist die Kollisionsvermeidung. Dies ist aber nur zweckmäßig, wenn das Objekt bereits vor dem Zusammenstoß erkannt wird. Aus diesem Grund erscheint es plausibel, daß ein Leseneuron nicht nur die Aktivität in einer Domäne ausliest, sondern auch die Nachbardomänen mit betrachtet. Es wird somit mit einem Quader von 3 x 3 x 3 = 27 Domänen verbunden, die es mit Informationen versorgen, und besitzt dadurch ein dreidimensionales rezeptives Feld.

Nun wird die Struktur des Koordinatensystems beschrieben (Fig. 12.1). In der Mitte befindet sich der Kopf. Eine Position wird durch die Winkel φ und ϑ, sowie den Abstand r vom Zen-

trum beschrieben. Der Winkel φ liegt zwischen –180° und 180° und ϑ liegt zwischen -90° und 90°.

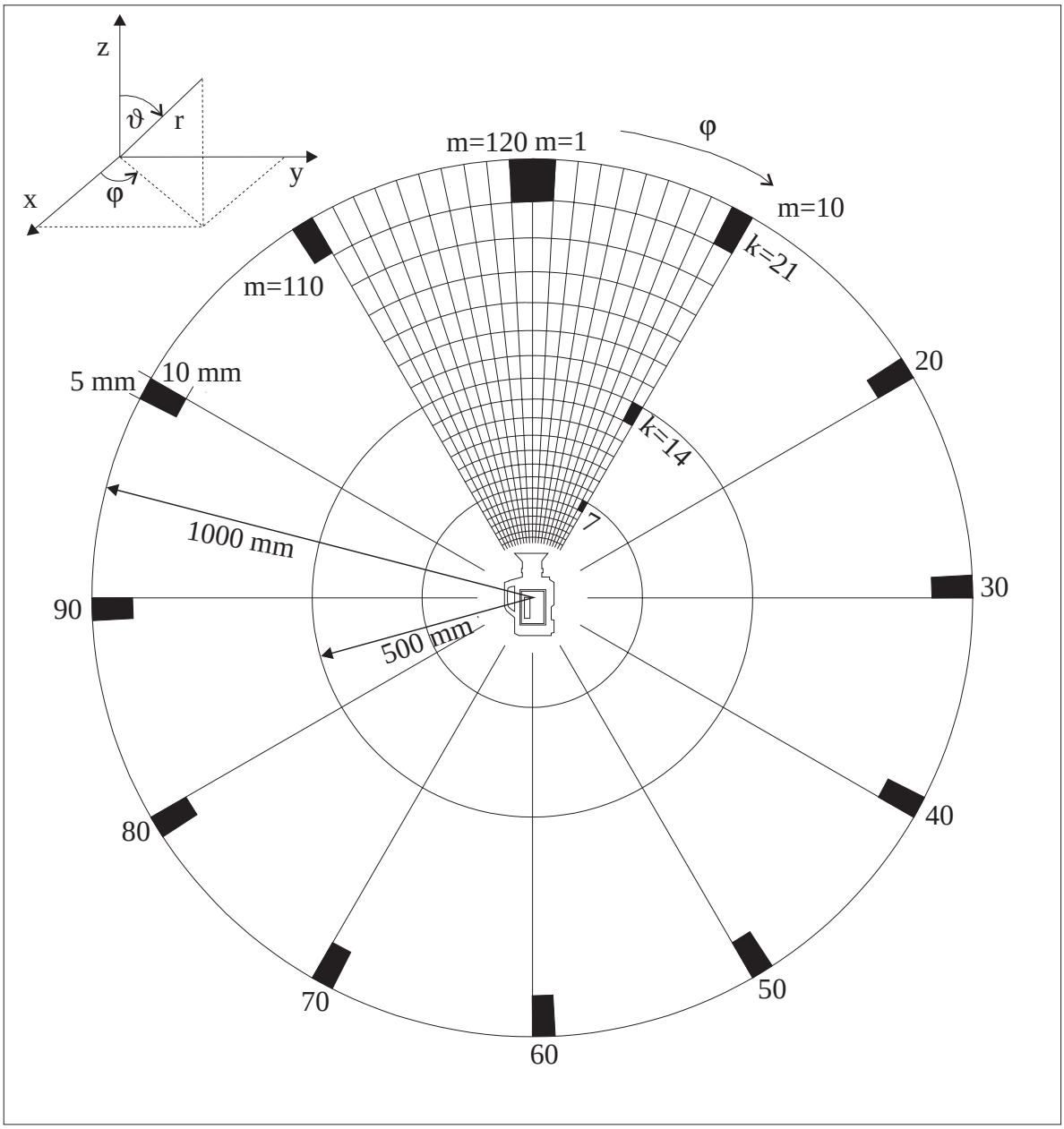


Fig. 12.1: Der Raum wird in kleine Domänen der Größe 5 cm x 5 cm x 10 cm aufgeteilt.

Von aufgenommenen Objekten, die sich in einer Entfernung von 1 m zum Kopf befinden, ist bekannt, daß der seitliche Fehler bei 5 cm und der Entfernungsfehler bei Abschätzungen mit geschlossenen Augen bei 10 cm liegt. Auf dieser Grundlage wird nun eine Domäne der Größe 5 cm x 5 cm x 10 cm für den Abstand von einem Meter konstruiert. Dies ergibt 120 Domänen in der φ-Richtung und 60 Domänen in der ϑ-Richtung. In der radialen Ausdehnung wird eine

Domäne durch die Grenzen r = 1 m und r = 1,1 m eingegrenzt. Der nächste Ring von Domänen wird durch den Radius r = $1{,}1^2$ m und der dritte Ring durch den Radius r = $1{,}1^3$ m begrenzt. Nach dem siebten Ring ergibt sich ein Radius von r = $1{,}1^7$ m. Der Radius hat sich somit fast verdoppelt. Um den gesamten Raum zu erfassen, wären unendlich viele Ringe notwendig.

Normalerweise gibt es aber eine innere und eine äußere Grenze. Die Innengrenze ergibt sich durch das Vorhandensein der Kamera, wobei der Innenradius auf 1/16 m festgelegt wird. Das Koordinatensystem innen abzuschneiden, ist zulässig, da keine Aktivitätswolke durch das Zentrum geschoben werden kann. Dies hieße nämlich, daß sich das Objekt durch die Kamera Kopf hindurch bewegen würde.

Die Außengrenze wird willkürlich auf 60 Ringe begrenzt. Berechnet man nun, ausgehend vom Innenradius r = 1/16 m und bedenkt, daß sich der Radius nach sieben Ringen jeweils verdoppelt, die Gesamtausdehnung, so erhält man eine Ausdehnung von über 25 m.

Nachdem nun die Eckwerte festgelegt sind, kann die Gesamtzahl der Domänen bestimmt werden. Es ergeben sich $1 \leq k \leq 60$ Domänen in r-Richtung, $1 \leq l \leq 60$ in ϑ-Richtung und $1 \leq m \leq 120$ in φ-Richtung. Eine Domäne wird durch das Trippel (k, l, m) beschrieben.

12.2 Abbildung der 3D- auf eine 2D-Struktur

Bisher wurden die Neuronen bezogen auf das Gitter des Koordinatensystems gesehen. Nun soll das neuronale Modell in eine reale Struktur eingebettet werden. Würden die Neuronen nebeneinander plaziert werden, dann ergäbe sich ein Quader von $60 \cdot 120 \cdot 60$ Domänen (Fig. 12.2 (a)). Hierbei bliebe die Struktur erhalten, jedoch ergäben sich zwei Brüche: Einer an den Polen ($\vartheta = \pm 90°$) und der andere in φ-Richtung ($\varphi = \pm 180°$). Während das polare Gitter in ϑ- und φ-Richtung geschlossen ist, geht die Repräsentation im Quader nur von einer Ecke zur anderen. Dieser Punkt wird im Moment vernachlässigt. Er wird jedoch später nochmals aufgegriffen.

Das wahre Problem ist vielmehr die Größe des Quaders. Ein Quader mit 60 Domänen oder 1000 Gitterneuronen würde eine Platte mit einer Dicke von 3 bis 6 mm ergeben, wenn sehr kleine Neuronen mit einer Ausdehnung von 5 bis 10 µm benutzt würden. Doch steht dies im Widerspruch zu allen bisher gefundenen Neuronengruppen. Außerdem wäre es sehr schwierig, alle Neuronen mit Eingangssignalen zu versorgen. Aus diesem Grund wird der Quader in kleine Scheiben mit der Dicke einer Domäne aufgeteilt (Fig. 12.2 (b)). Alle Domänen mit k = l = m = 1 bilden die erste Scheibe, die Domänen mit k = l = m = 2 bilden die zweite Scheibe

und so weiter. Dies bedeutet, daß eine Scheibe in φ-Richtung eine Dicke von einer Domäne und eine Ausdehnung von 60 x 60 Domänen in ϑ- und r-Richtung besitzt. Ein Nachteil dieser Aufteilung ist, daß die Neuronen an der rechten Seite der Scheibe m mit Neuronen auf der rechten Seite der Scheibe m+1 verbunden sein müssen. Verglichen mit der Anzahl alle Verbindungen, machen diese Verbindungen aber nur ca. 2% aus.

Nun werden die einzelnen Scheiben um 90° gedreht und dann nebeneinander gelegt (Fig. 12.2 (b)). Wir erhalten eine Struktur mit der Dicke einer Domäne. Da nun alle Domänen in der gleichen Schicht liegen, kann über jeder Domäne ein Leseneuron plaziert werden (Fig. 12.3 (a)).

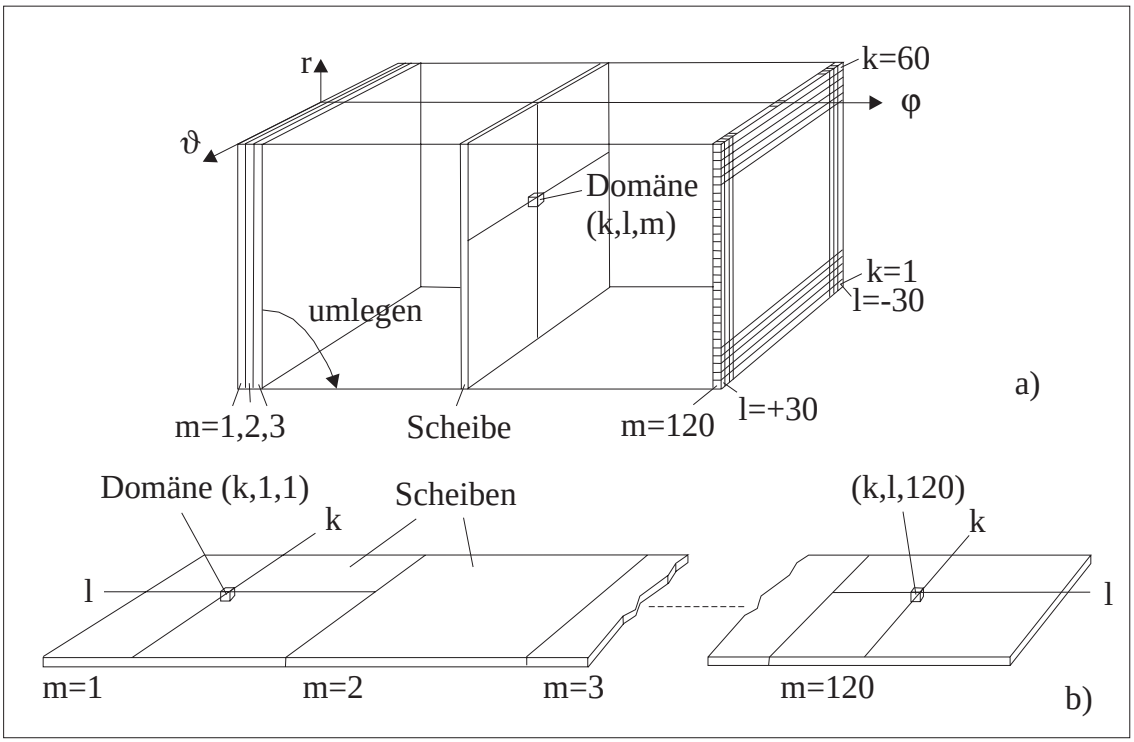


Fig. 12.2: Die dreidimensionale Domänenstruktur (a) wird in Scheiben mit der Dick einer Domäne zerlegt, und die einzelnen Scheiben werden nebeneinander gelegt (b).

Wie beschrieben, sammelt ein Leseneuron die Signale von 3 x 3 x 3 Domänen. Bei der Betrachtung der Ausgangssignale einer einzelnen Domäne hat jedes Gitterneuron der Domäne (k,l,m) eine Verbindung zu den gleichen Leseneuron (Fig. 12.3 (a)). Aus einer Domäne heraus ergeben sich 1000 Verbindugen. Doch jedes Leseneuron muß nach dem Modell auch die Information von den Nachbardomänen $k \pm 1$ und $l \pm 1$ einsammeln. Außerdem müssen die restlichen 18 Domänen verbunden werden. Hierzu ist eine kompakte und „ordentliche“ Vernetzung vorteilhaft.

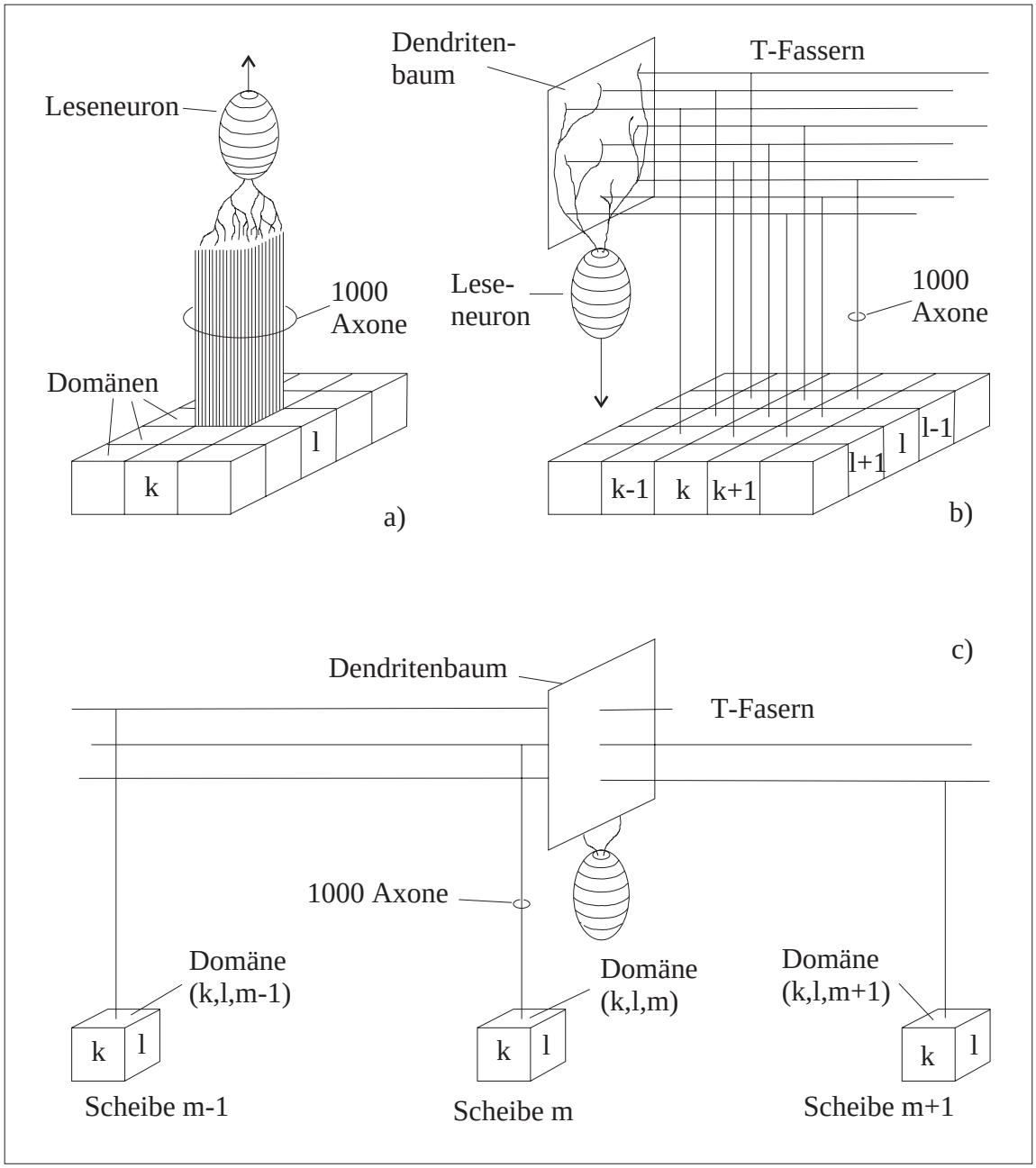


Fig. 12.3: Versorgung der Leseneuronen durch die Neuronen der einzelnen Domänen.

Um dies zu erreichen, werden die Axone der Gitterneuronen T-förmig ausgerichtet. Sie bilden ein Bündel paralleler Fasern, die entlang der Fläche der Scheiben verlaufen. Im folgenden werden diese Fasern als *T-Fasern* bezeichnet. Ein Leseneuron mit einem weit ausgedehntem fächerförmigen Dendritenbaum kann nun alle seine Eingangssignale aufsammeln. Da für jede Domäne ein Leseneuron existiert, kann eine Schicht von ihnen oberhalb der Gitterneuronen angeordnet werden. Die Leseneuronen können sehr dicht und ordentlich gepackt werden, wenn die Fächer parallel zueinander sind und senkrecht auf den T-Fasern stehen.

Um alle benachbarten Domänen einer Scheibe, sagen wir m, miteinander zu verbinden, sind die T-Fasern praktisch. Sie können die Signale von den vorbeikommenden T-Fasern nehmen (Fig. 12.3 (b)). Außerdem sind sie notwendig, wenn die Ausgangssignale von den Domänen in benachbarten Scheiben (m-1) und (m+1) gesammelt werden müssen. Ein Leseneuron muß Informationen von den Scheiben (m-1), m und (m+1) aufnehmen. Dies geschieht dadurch, daß die Äste der T-Fasern so lang wie eine Scheibe sind und so den Dendritenbaum des entsprechenden Leseneurons erreichen (Fig. 12.3(c)).

Somit bildet die Kombination der langen T-Fasern mit den fächerförmigen Dendritenbäumen ein hochwertiges Verbindungssystem, um alle Ausgangssignale zu den entsprechenden Leseneuronen zu führen. Die T-Fasern verbinden außerdem die Gitterneuronen einer Domäne mit den Gitterneuronen der entsprechenden Nachbardomäne in der anderen Scheibe, wenn nur 2% von den T-Fasern Verbindungen bis zu den Gitterneuronen besitzen.

12.3 Parallelitäten zur Anatomie des Celebellums

Es ist zu beachten, daß die im letzten Kapitel vorgestellten Strukturen sich direkt aus dem vorgestellten Modell des RRNs und der Notwendigkeit, die Struktur zweidimensional darzustellen, ergeben. Es ist überraschend, daß sich bestimmte Analogien zu Kleinhirnstrukturen ergeben, wenn die Gitterneuronen durch Körnerzellen und die Leseneuronen durch Purkinjezellen ersetzt werden.

Die Annahme, daß die Gitterneuronen Körnerzellen sind, bedeutet, daß die Moosfasern die Shiftimpulse und die Mustersignale liefern. Dies stimmt mit den in der Literatur gefundenen Angaben über Moosfasern überein [Bloedel und Courville1981]. Die Impulse der Moosfasern werden von sechs oder sieben Dendriten der Körnerzellen aufgenommen. Dies entspricht den sechs Shiftleitungen und der einen Eingangsleitung. Die dichtgepackte, gleichmäßige Anordnung der Körnerzellen in der Körnerschicht paßt zur Struktur der Gitterneuronen im RRN. Außerdem wurden die Parallelfasern der Körnerzellen durch die T-Fasern in dem Modell beschrieben.

Der Durchmesser einer Körnerzelle beträgt 5 bis 8µm. Legt man 10 Zellen übereinander (eine Domäne) und berechnet noch den Platz für die Glumerus-Synapsen (20µm) sowie der eingestreuten Golgizellen und der Verbindungen, ergibt sich eine Schicht, die 300 bis 400µm dick ist (entsprechend der Angaben in der Literatur). Die Festlegung, daß die Gitterneuronen Körnerzellen sind, steht somit nicht im Widerspruch zu Ergebnissen der Biologie ([Eccles 1967] [Eccles 1977] und [Fox 1967]).

Die Annahme, daß die Leseneuronen Purkinjezellen sind, findet sich ebenfalls in biologischen Erkenntnissen wieder. Es ist bekannt, daß die Purkinjezellen Signale von einer großen Zahl an Körnerzellen erhalten. Die für die Purkinjezellen typischen Dendritenbäume werden in dem Modell durch die fächerförmigen Dendritenbäume beschrieben.

Als weiteres können die T-Fasern, die an einer Zelle vorbeiführen, berechnet werden. Jeden Zweig einer T-Faser kreuzt eine Scheibe mit 60 Domänen. Die beiden Enden überdecken somit 120 Domänen, dies bedeutet, daß sich die Fasern von ca. 120 Domänen überlappen und an einer Position die T-Fasern von 120 Domänen bzw. 120000 Körnerzellen parallel verlaufen. Da eine Dendrite von drei dieser Bündel ihre Eingangssignale erhält, ergeben sich insgesamt 360000 T-Fasern. Auf der anderen Seite erhält eine Purkinjezelle nur Signale von 27 Domänen. Somit braucht eine Zelle nur mit 27000 Fasern verbunden sein. Die berechneten Werte stimmen gut mit den biologischen Ergebnissen überein ([Eccles, 1967], [Eccles, 1977] und [Fox, 1957]).

Die nächste Gegenüberstellung betrifft die Anzahl und die Distanz der Purkinjezellen. Vergleichbar mit Ergebnissen der Anatomie gibt es 1000 Körnerzellen (eine Domäne) pro Purkinjezelle in dem Modell. Wenn wir annehmen, daß die Körnerzellen zu dichten Scheiben aufgeschichtet sind, benötigen die $10 \cdot 10$ Neuronen einer Dömäne ca. 2500 $(\mu m)^2$. Mit einer Purkinjezelle pro Domäne bzw. pro 2500 $(\mu m)^2$, erhalten wir 400 Purkinjezellen pro mm^2. Dies ist eine gute Annäherung an die gefundenen Ergebnisse, wo von 500 Zellen pro mm^2 ausgegangen wird. ([Benninghoff, 1985], [Eccless, 1967]).

Abschließend kann die Ausdehnung der T-Fasern aus dem beschrieben Model berechnet werden. Werden die Purkinjezellen zu einem hexagonalen Gitter wie in Fig. 12.4 angeordnet, ergeben sich ca. 400 bis 500 Purkinjezellen pro mm^2. Nun kann die Distanz der einzelnen Zellen berechnen werden. Es ergibt sich eine Distanz von 90µm in ϑ-Richtung und 25µm in r-Richtung. Eine Scheibe mit $120 \cdot 60$ Domänen oder $120 \cdot 60$ Purkinjezellen dehnt sich über $10800\mu m \cdot 1500\mu m$ bzw. $10{,}8mm \cdot 1{,}5mm$ aus. Die Länge von 1,5mm muß von einem Ausläufer der T-Fasern überwunden werden. Somit ergibt sich eine Gesamtausdehnug von 3mm. Dieser Wert findet sich in der Anatomie ebenfalls wieder. ([Eccles, 1977] [Fox, 1957]).

Nun kann über die grundlegenden Konsequenzen gesprochen werden. Im Model des Neuronengitters wird lokale Information ausgelesen, ohne daß das ganze Gitter zu betrachten ist. Somit ist es eine gute Idee, die Information aus den Purkinjezellen nur aus lokalen Teilen zu bestimmen. Bei [Eccles 1966a], [Eccles 1966b] und [Voorhoeve 1967] wird erwähnt, daß die Aktivität der Purkinjezellen mit Hilfe der Kletterfasern geschieht. Nach dieser Vorstellung können die Körnerzellen das Membranpotential der Purkinjezellen zwar anheben, aber kein

Aktionspotential erzeugen. Hingegen können die Kletterfasern eine Purkinjezelle sofort aktivieren. Der erzeugte Spike wird zum aktuellen Membranpotential hinzugerechnet. Ist das Membranpotential vorher niedrig, daß heißt, keine oder sehr wenige der entsprechenden Körnerzellen sind aktiv, so wirkt sich der Spike nicht aus. Ist das Membranpotential durch die entsprechenden Körnerzellen bereits angehoben worden, wird die Purkinjezelle aktiv.

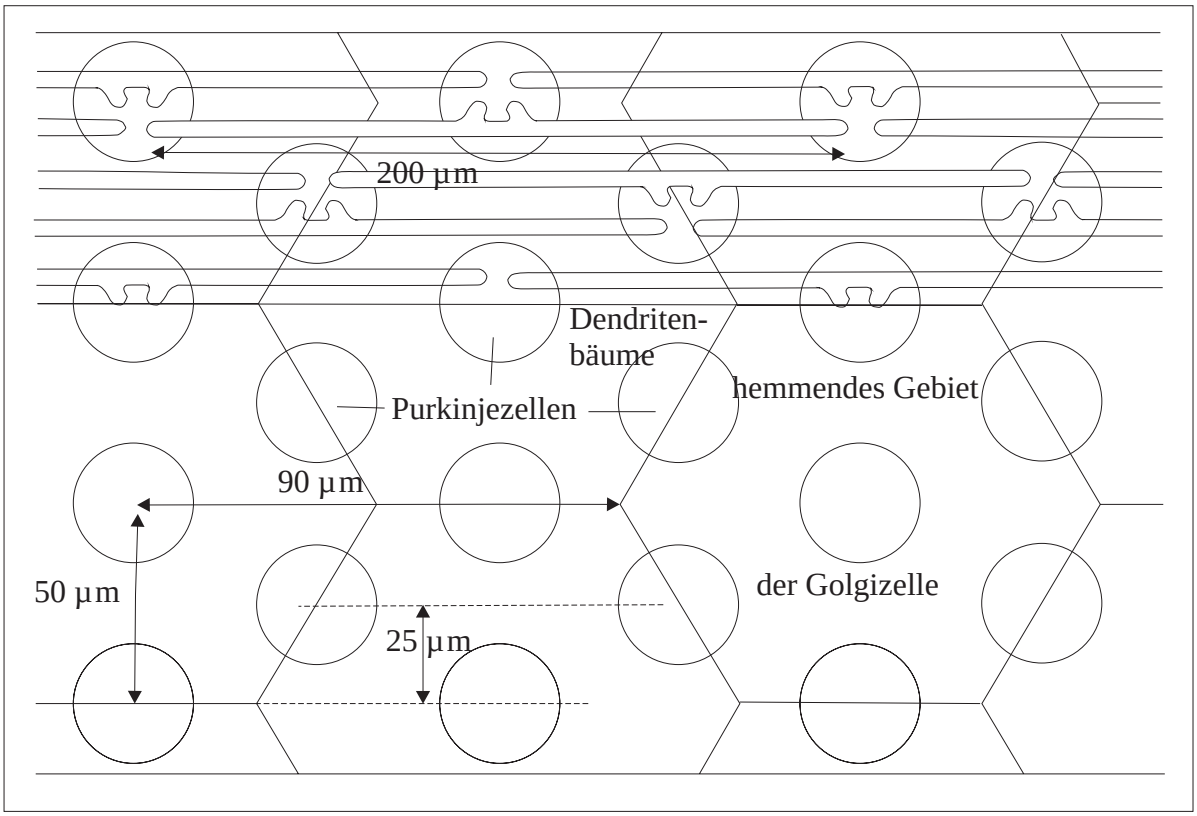


Fig.12.4: Anordnung der Purkinjezellen

Der Mechanismus zum Auslesen der Aktivität soll feststellen, ob im dreidimensionalen rezeptiven Feld (27 Domänen) der Purkinjezellen ein Objekt ist oder nicht. Eine kleine Aktivitätswolke, die nur einige hundert Neuronen in einer der 27 Domänen aktiviert, muß genausogut bemerkt werden wie eine Wolke, die fast alle Neuronen der 27 Domänen aktiviert. Aber es ist fast unmöglich, daß 100 aktive Neuronen das Membranpotential so erhöhen, daß die Zelle überschwellig wird, es gleichzeitig aber auch möglich ist, daß die Zelle durch 27000 Neuronen gleichzeitig aktiviert wird. Eine Lösung dieses Problems liegt darin, die Purkinjezellen zu hemmen, wenn die Anzahl der aktiven Körnerzellen steigt. Dies geschieht durch zusätzliche Zellen, die die Aktivität benachbarter Domänen messen, und dementsprechend die Purkinjezellen der betreffenden Region hemmen. Es gibt mehrere Arten von hemmenden Zellen im cerebellaren Cortex.

Etwas spekulativer ist die Erklärung des Shift-Mechanismus. Der Mechanismus der Simulation ist nicht kompatibel mit den bisher festgestellten Strukturen des Cerebellums. Besonders gibt es keine eingehenden hemmenden Fasern oder Interneuronen. Abhilfe könnte ein Shiftmechanismus schaffen, der verwandt ist mit der synaptischen Struktur des Glomeruli. Im Unterschied zum ursprünglichen Modell gibt es eine Hemmung nicht wegen eines Shiftimpulses, sondern dadurch, daß ein Shiftimpulse erst erzeugt wird. Diese hemmende Aktivität wird durch Golgi- Zellen bewirkt, welche nur solange aktiv sind, wie ein Muster in der Domäne vorhanden ist.

An dieser Stelle sei erwähnt, daß die vorliegenden Ausführungen nur ein Versuch sind, das Modell der Simulation auf Strukturen des Kleinhirnes abzubilden. Es ist sehr erfreulich, daß Ähnlichkeiten gefunden werden konnten. Doch auch dieses Modell kann sich nur in die lange Reihe der Modelle einreihen, die ebenfalls versuchen, die Struktur des cerebellaren Cortexes zu beschreiben (z.B: [Braitenberg 1958], [Eccles 1977], [Albus 1979] und [Houk 1987]).

13. Zusammenfassung

Die vorliegende Arbeit beschreibt zwei neuronale Netze, die beide mit pulscodierten Neuronen arbeiten. Das CLAN ist ein Klassifizierer, der unüberwacht Muster lernt und in wohldefinierte Klassen unterteilt. Wird ein Muster angelegt, so kann das Netzwerk selbständig ohne Lehrer entscheiden, ob das angelegte Muster unbekannt ist, oder ob es schon eine Musterklasse gibt, in die das Muster gehört. So ist kein explizites Umschalten zwischen Erkennen und Lernen notwendig. Ein unbekanntes Muster braucht nur einmal gezeigt werden, und die sonst üblichen Trainingssequenzen entfallen.

Da die Simulationen des neuronalen CLANs sehr zeitaufwendig ist, wurde untersucht, in wie weit es möglich ist, es zu parallelisieren.. Durch die Verteilung je eines CLANs auf einen Transputer konnte eine passende Parallelisierungstrategie gefunden werden.

Für den Einsatz in einem Erkennungssystem war dieser Ansatz jedoch noch zu langsam. Deshalb wurde eine sequentielle Version implementiert, die als letztes Glied in das Erkennungssystem PDIS eingebaut wurde. Mit diesem System werden Erkennungsraten von 1-2 s erreicht.

Als zweites Netz wurde das Raumrepräsentationsnetzwerk RRN betrachtet. Im RRN werden Objekte in einem neuronalen Gitter in einem kamerafesten Koordinatensystem durch Aktivitätswolken dargestellt. Eine Bewegung des Koordinatensystems führt zu einer Verschiebung der Wolke in die entgegengesetzte Richtung. Die Verschiebung wird nur durch Erzeugen und Löschen von Aktivitäten vollzogen. Der vorgestellte Mechanismus garantiert eine stabile Repräsentation der Objekte.

Das RRN wurde im ein-, zwei- und dreidimensionalen kartesischen Koordinatensystem simuliert. Da die Betrachtung des zweidimensionalen Polarkoordinatensystem biologisch plausibler erscheint, wurde dieses ebenfalls betrachtet, wobei speziell auf die Größeninvarianz eingegangen wurde.

Um die nötige Rechenleistung zu erhalten, wurde das RRN im dreidimensionalen Fall parallelisiert. Die globalen Bewegungsimpulse werden an den ersten Transputer gesendet, welcher hieraus berechnet, in welchen Domänen wann Aktivität verschoben werden muß. Die Daten der einzelnen Domänen sind auf die anderen Transputer verteilt, wo jeweils die Aktivität zu verschieben ist. Durch eine geschickte Verteilung sind alle Transputer etwa gleichmäßig ausgelastet.

Für den zweidimensionalen Fall wurde das RRN mit einem Roboter gekoppelt, an dessen sechster Achse eine Kamera befestigt ist. Von der Robotersteuerung erhält das neuronale Netz als Eingabe Bewegungsimpulse entsprechend der Bewegung des Roboters bzw. der Kamera. Hieraus werden die Shiftimpulse der Shiftneuronen bestimmt und die Aktivitätswolke diesbezüglich verschoben. Es ist möglich, kleinere Bewegungen mit der Kamera im zweidimensionalen Raum zu simulieren.

Außerdem wurde ein Mechanismus vorgestellt, der die Aktivitätswolke durch Anschauen von Objekten erzeugt. Hierzu ist es notwendig, das Objekt von mehreren Seiten zu betrachten. In einem Ausflug in die Anatomie des Cerebellums ergaben sich Parallelitäten zwischen Strukturen und Funktionsweisen des Kleinhirns und des RRNs.

Literatur

[Benninghoff 1985]
Benninghoff, A.: *Makroskopische und mikroskopische Anatomie des Menschen. 3. Band. Nervensystem, Haut und Sinnesorgane.* Urban & Schwarzenberg, München, 1985.

[Benninghoff 1994]
Benninghoff, A.; Hrsg. von D. Drenckhahn und W. Zenker: *Anatomie: Makroskopische Anatomie, Embryologie und Histologie des Menschen*. Band 2. Urban & Schwarzenberg, München, 1994.

[Bredenbals 1993]
Bredenbals, R.: *Implementierung eines 2-dimensionalen neuronalen kamerafesten Raumrepräsentationsnetzwerk im Polarkoordinatensystem.* Diplomarbeit an der Universität Paderborn, Fachbereich Grundlagen der Elektrotechnik, unveröffentlicht, 1993.

[Busemann 1993]
Busemann, M.; Dunker, J.; Hartmann, G.; Kräuter, K.O.; Seidenberg E.; Wiemers, H.: *Building an Artificial Retina for Distance- and Orientation-Invariant Pattern Recognition.* Gielen, S. Kappen, B. (Hg.): Proc Intern Conf on Artificial Neural Networks. ICANN 93. S. 888-893, Berlin Springer 1993.

[Busemann 1994]
Busemann, M.; Hartmann, G.; Drüe, S.: *Simulation eines neuronalen Raumrepräsentationsnetzwerks auf einem Transputernetz.* Grebe, R.; Hektor, J. : Parallele Datenverarbeitung mit dem Transputer (TAT '93). Informatik aktuell. S. 176-194, Springer Berlin, 1994.

[Carpenter 1987]
Carpenter, G. A.; Grossberg, S.: *ART2: Self-organisation of stable category recognition code for analog input patterns*. Applied Optics, Vol. 26 (23) (1987), S. 4919 – 4930.

[Eccles 1966]
Eccles, J. C.; Llinas, R.; Sasaki, K.: *The excitatory synaptic action of climbing fibres on the Purkinje cells of the cerebellum.* J. Physiol, 182 (1966), S. 269 – 296.

[Eccles 1967]
Eccles, J. C.; Ito, M.; Szentagothai, J.: *The Cerebellum as a Neuronal Machine.* Springer Berlin, 1967.

[Eccles 1977]
Eccles, J. C.; *An instruction-selection theory of learning in the cerebellar cortex.* Brain Res. 127 (1977), S. 327 – 352.

[Eckhorn 1989]
Eckhorn, R. et al.: *Feature linking via stimulus-evoked oscillations: Experimental results from cat visual cortex and funktional implications from a network model.* Proc. IJCNN 89, IEEE, 1.723-1.730 (1989)

[Fox 1957]
Fox, C. A.; Barnard, J. W.: *A quantitative study of the Purkinje cell dendritic brachlets and their relationship to afferent fibres.* J. Anat. 91 (1957), S. 299 – 313.

[Franke 1993]
Franke, J. et al. *Experiments with the CENPARMI Data Base Combining Different Classification Approaches.* Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition, Buffalo 1993, S. 305-311

[French 1970]
French, A.S.; Stein, R.B. *A flexible neural analog using integrated circuits.* IEEE Trans. Biomed. Eng., 17 (1970) S. 248-253

[Grossberg 1976]
Grossberg, S: *Adaptive pattern classification and universal recoding, II: Feedback expectation, olfaction and illusions.* Biological Cybernetics 23, S. 187 – 202 (1976).

[Grossberg 1993]
Grossberg, S.; Guenther, F.; Bullock, D. and Greve, D.*; Neural Representations for Sensory-Motor Control, II*: *Learning a Head-Centered Visuomotor Representation of 3-D Target Position.* Neural Networks, 6(1), S. 43-67 (1993).

[Hartmann 1991a]
Hartmann, G.: *Learning in a Closed Loop Antagonistic Network.* Kohonen, T. et al. (Hg.): Artificial Neural Networks. Proc. ICANN 91. Amsterdam (Elsevier Science Publishers/North Holland) 1991, S. 239-244

[Hartmann 1991b]
Hartmann, G.: *Hierarchical Neural Representation by Synchronized Aktivity: A Conzept for Visual Pattern Recognition.* Taylor, J.G. et al. (Hg.): Neural Network Dynamics S. 356-370, London Springer-Verlag 1991.

[Hartmann 1992a]
Hartmann, G.: *Neural space representation in a moving frame.* Proc International Joint Conference on Neural Networks (IJCANN), Bd. 1 S. 92-97, Baltimore 1992.

[Hartmann 1992b]
Hartmann, G.: *Motion Induced Transformations of Spatial Representations: mapping 3D Information onto 2D.* Shun-Ichi, A.; Grossberg, St.; Tayer, J.(Hg.): Neural Networks, Bd. 5, S. 823-834, (1992)

[Hartmann 1992c]
Hartmann, G.: *Architectural Consequences of Mapping 3D Space Representations onto 2D.* In: Aleksander, I., Taylor, J. (Hg.): Artificial Neural Networks. 2. S. 899-902, Amsterdam (Elsevier Science Publishers/North-Holland) 1992.

[Hartmann 1993]
Hartmann, G.; Kräuter, K.O.; Wiemers, H.; Seidenberg, E.; Drüe, S.: *Ein distanz- und orientierungsinvariantes lernfähiges Erkennungssystem für Robotikanwendungen.* Pöppl, S.J.; Handels, H. (Hg.): Mustererkennung 1993. Informatik aktuell. S. 375-382, Berlin Springer Verlag 1993.

[Hebb 1949]
Hebb, D. O.; *The organisation of behavior.* Wiley, New York, 1949.

[Heitkkonen 1993]
Heitkkonen, J.; Oja, E.: *Self-Organizing Maps for Visually Guided Collision-free Navigation*. Proc Intern Joint Conf on Neural Networks. Vol 1, S. 669-672, Nagoya 1993.

[Hosaka 1993]
Hosaka, R.; Nagano, T.; *A Neural Network Model for Spatial Information Representation.* In: Gielen, St.; Kappen, B.: Proce Intern Conference on Artificial Neural Networks, S. 111-114, Springer Verlag, Amsterdam, 1993.

[Hubel 1990]
Hubel, D. H.: *Auge und Gehirn.* Spektrum der Wissenschaft Verlagsgesellschaft, Heidelberg 1990.

[Irrgang 1994]
Irrgang, M.: *Parallelisierung eines 2-dimensionalen neuronalen kamerafesten Raumrepräsentationsnetzwerkes im Polarkoordinatensystem.* Diplomarbeit Universität Gesamthochschule Paderborn, Fachbereich Grundlagen der Elektrotechnik, unveröffentlicht, 1994.

[Kahle 1986]
Kahle, W.: *Nervensystem und Sinnesorgane.* Hrsg W. Kahle, H.Leonhardt, W. Platzer: *Taschenbuch der Anatomie Band 3*. Georg Thieme Verlag, Stuttgart, 1986.

[Kalthoff 1990]
Kalthoff, B.: *Antagonistischer Assoziativspeicher mit pulscodierten Neuronen.* Diplomarbeit Universität Gesamthochschule Paderborn, Fachbereich Grundlagen der Elektrotechnik, unveröffentlicht, 1990.

[Kohonen 1984]
Kohonen, T.: *Self-Organization and Associative Memory.* Springer Verlag, Berlin, 1984.

[Kräuter 1995]
Kräuter, K.O: *Distanz- und orientierungsinvariante Extraktion von Konturinformation aus den Kameradaten eines Robotor-Vision-Systems.* (Dissertation) Universität Paderborn, Fachbereich Grundlagen der Elektrotechnik, 1995.

[Malsburg 1973]
von der Malsburg, C.: *Self-organization of orientation sensitive cells in the striate cortex.* Kybernetik 14, S. 85 – 100, (1973).

[Pötzsch 1994]
Pötzsch, H.: *Implementierung einer schnellen algorithmischen Lösung eines Musterklassifikators auf Basis des Closed Loop Antagonistic Network.* Diplomarbeit Universität Gesamthochschule Paderborn, Fachbereich Grundlagen der Elektrotechnik, unveröffentlicht, 1994.

[Rauber 1987]
Rauber; Kopsch: *Anatomie des Menschen.* Hrsg. H. Leonhardt, G. Töndury und K. Zilles*:*, *Lehrbuch und Atlas,* Band III, Nervensystem, Sinnesorgane, Georg Thieme Verlag Stuttgart, 1987.

[Reitböck 1984]
Reitböck, H.J: Altmann, J.: *A Model for Size- and Rotation-Invariant Pattern. Processing in the Visual System.* Biol. Cybernetics 51 (1984), S. 113-121

[Rummelhart 1996]
Rummelhart, D. E.; Hinton, G. E.; Williams, R. J.; *Learning internal representations by backpropagating errors.* Nature, 323: S. 533 –536 (1986).

[Schwarz 1981]
Schwarz, E.L.: *Cortial Anatomy, Size Invariance and Spatial Frequency Analysis.* Perception, 10 (1981), S. 831-835

[Thompson 1990]
Thompson, Richard F.: *Das Gehirn: Von der Nervenzelle zur Verhaltenssteuerung.* Spektrum der Wissenschaft 1990.

[Wiemers 1994]
Wiemers, H.: *Architektur eines Robot-Vision-Systems für ein distanz- und orientierungsvariantes Greifen auf basis neuronaler Verarbeitungsmechanismen.* (Dissertation) Universität Paderborn, Fachbereich Grundlagen der Elektrotechnik, 1994.

[Willshaw 1972]
Willshaw, D.; Longuet-Higgins, H. C.; Buneman, P.: *A simple network capable of inductive generalization.* Proc. R. Soc. London, 182, S. 233 – 247 (1972).

[Zillers 1993]
Zillers, K.; Rehkämper, G.: *Funktionelle Neuroanatomie, Lehrbuch und Atlas*. Springer-Verlag, Berlin, Heidelberg, 1993.