Technical Report

Pattern Recognition and Image Processing Group
Institute of Automation
Vienna University of Technology
Treitlstr. 3/1832
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801-8161
Fax: +43 (1) 5054668
E-mail: melzer@prip.tuwien.ac.at

PRIP-TR-48 September 24, 1997

Adaptive Robotersteuerung mittels visueller Rückkopplung

Thomas Melzer

Abstract

An adaptive system for kinematic robot control based on visual feedback is presented. The system is capable of moving the effector of an industrial robot to the position of an target object, whose coordinates are extracted from a pair of stationary mounted CCD cameras. The adaptive component consists of an extended neural gas network, which will - without any prior knowledge about camera orientation or robot arm architecture - eventually learn the mapping from stereo image coordinates to associated robot joint angles, the so called hand eye transform. Following the discussion of self organizing systems and the description of the system components, the results of four software experiments are presented, which shall illustrate the impressive performance, but also some weaknesses of the extended neural gas model. Finally, the results of an experiment conducted in a real hardware environment are presented.

Inhalt

1. Einleitung

1.1. Motivation und Zielsetzung

Das korrekte Anfahren eines vorgegebenen Punktes im Arbeitsraum, um ein dort befindliches Objekt zu manipulieren - die *Endeffektorpositionierung* -, ist eine in typischen Anwendungen der Robotik häufig zu bewältigende Aufgabe. Die vom Roboter während des Arbeitsprozesses anzufahrenden Positionen werden normalerweise zuvor durch das sogenannte *teach in* festgelegt: der Effektor - z.B. ein Greifer - wird dabei manuell in die gewünschte Position und Orientierung gebracht, anschließend werden die aktuellen Positionsparameter abgefragt und gespeichert. Es wird also der Roboter selbst als Meßwerkzeug eingesetzt.

Jedoch können sich im Laufe der Zeit - infolge von Last und Verschleißerscheinungen - die kinematischen Parameter[1] des Roboters verändern, was dazu führt, daß zuvor *geteachte* Positionen nicht mehr korrekt (d.h. nicht mehr innerhalb einer vorgegebenen Wiederholgenauigkeit) angefahren werden. Das mathematische Modell des Roboterarms, welches von der Roboterfirmware dazu verwendet wird, Weltkoordinaten[2] in korrespondierende Winkel der einzelnen Roboterachsen umzurechnen (die **inverse Kinematik**), stimmt in diesem Fall nicht mehr mit der physikalischen Realität überein. Dies macht es notwendig, das Robotersystem regelmäßig zu rekalibrieren.

Gegenstand dieser Diplomarbeit ist eine adaptive, auf einem neuralen Netzwerk basierende Robotersteuerung, welche unter anderem imstande ist, die oben beschriebene Rekalibrierung selbsttätig durchzuführen. Die Idee dazu wurde ursprünglich von Thomas Martinetz [Martinetz, 1992] im Rahmen seiner Dissertation entwickelt und geht von folgendem Grundaufbau - welcher auch in Abbildung 1.1 und 1.2 dargestellt ist - aus: der relevante Ausschnitt des Arbeitsbereichs des Roboters wird von zwei festmontierten Kameras beobachtet.

---

[1] Die kinematischen Parameter umfassen u.a. die Länge der einzelnen Armsegmente, die Orientierung der Gelenkachsen usw.

[2] Position und Orientierung eines Objekts im Arbeitsbereich (also auch des Effektors) werden typischerweise in 6-dimensionalen *kartesischen Koordinaten* relativ zu einem feststehenden Roboterkoordinatensystem (dem sogenannten *base frame*) angegeben; die ersten 3 Koordinaten geben dabei den Translationsvektor und die restlichen 3 die Euler-Winkel der Rotationsmatrix an. Im folgenden wird angenommen, daß Welt- und Roboterkoordinatensystem zusammenfallen.

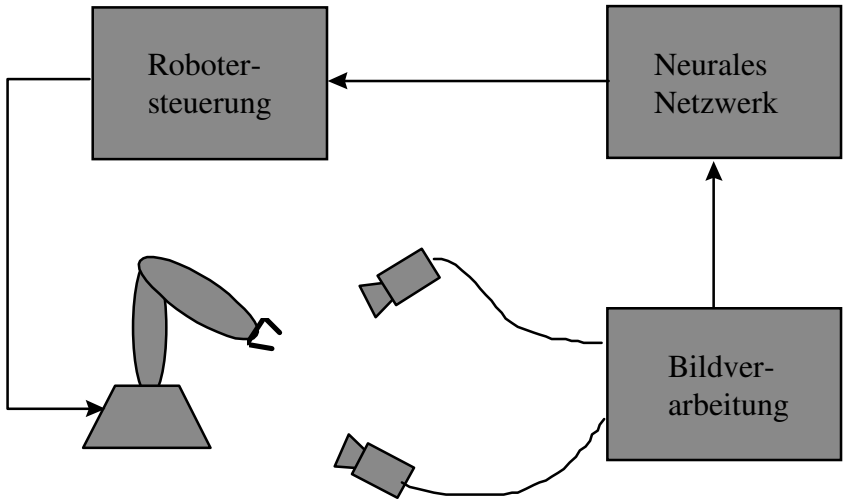

Abb 1.1: Logischer Aufbau des im Rahmen der Diplomarbeit implementierten Systems (siehe Text).

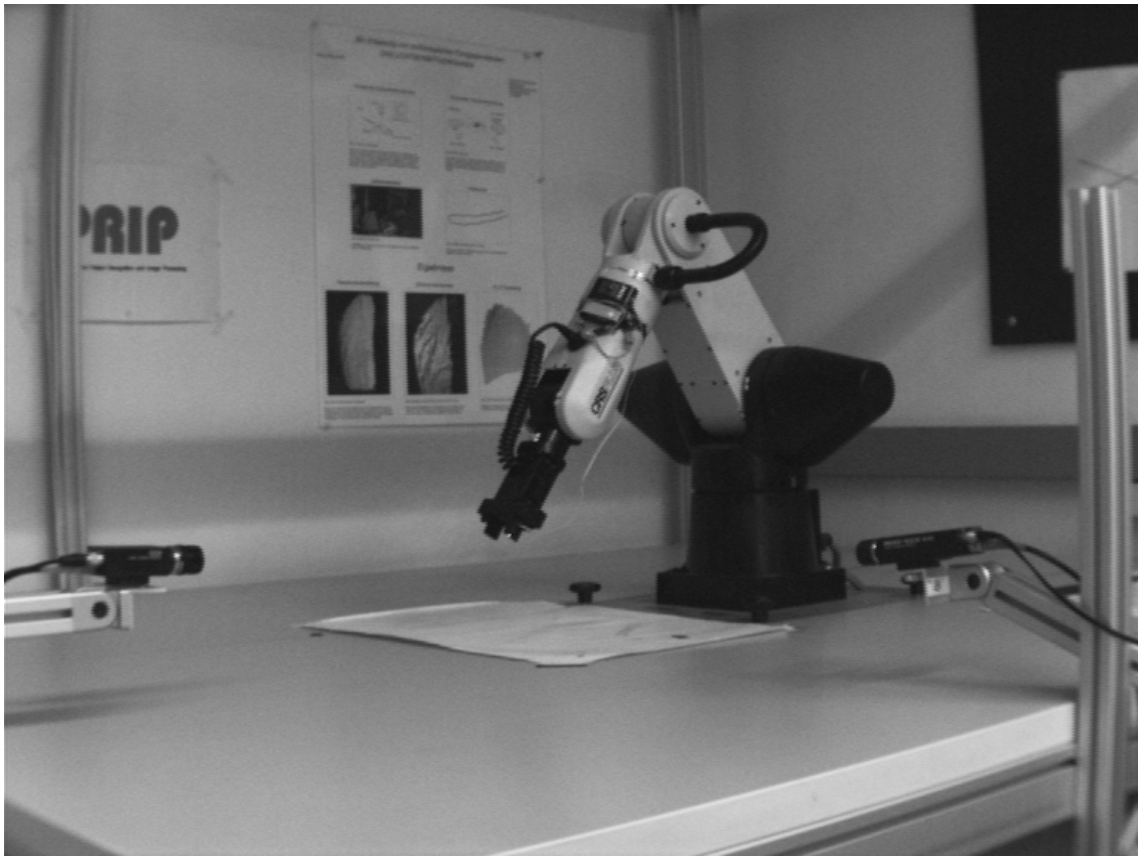

Abb. 1.2: Der ihm Rahmen der Diplomarbeit verwendete Industrieroboter vom Typ A-465. Der relevante Ausschnitt des Arbeitsbereichs wird von zwei CCD-Kameras erfaßt.

Zunächst wird die Position des anzufahrenden Objekts in beiden Kamerabildern ermittelt und dem neuralen Netzwerk übergeben, welches einen Satz von Gelenkwinkeln für die Grobpositionierung des Effektors berechnet. Nachdem der Roboter mit den so ermittelten Gelenkwinkelwerten angesteuert wurde, wird in den Kamerabildern die aktuelle Position des Effektors ermittelt; stimmt die aufgrund der berechneten Gelenkwinkel angefahrene Ist-Position nicht mit der Soll- (= Objekt-) Position überein, so veranlaßt das neurale Netz wiederholt Korrektur-bewegungen des Roboterarms, bis eine zufriedenstellende Positioniergenauigkeit erreicht wurde. Die Korrekturbewegungen dienen allerdings nicht nur der unmittelbaren Erhöhung der Positioniergenauigkeit, sondern liefern auch die für das Training der Ausgabeeinheiten des Netzwerks notwendige Information.

Die Aufgabe, die visuelle Repräsentation eines Objektes in korrespondierende Gelenkwinkel umzurechnen, wird in der Robotik und im Bereich der Computer Vision als *Hand-Eye Coordination* (siehe z.B. [Horn, 1986], S 312 ff.) bezeichnet. Der erste Schritt besteht in der Transformation der Bild- in Weltkoordinaten; dies setzt im Fall eines Stereobildverfahrens die exakte Kalibrierung sowohl der äußeren (Orientierung und Position beider Kamerakoordinatensysteme bezüglich des Weltkoordinatensystems) als auch der inneren (Linsenverzerrung, effektive fokale Länge etc.) Parameter beider Kameras voraus. Anschließend müssen die so gewonnenen Weltkoordinaten mittels der **inversen Kinematik** in korrespondierende Gelenkwinkel des Roboterarms umgerechnet werden. Beide Transformationsschritte arbeiten mit statischen, einmalig gewonnenen mathematischen Modellen der Kameras und des Roboters, was, falls sich ein oder mehrere Systemparameter ändern, zu fehlerhaftem Verhalten des Systems führt und eine Rekalibrierung erforderlich macht.

Bei dem ihm Rahmen der Diplomarbeit eingesetzten adaptiven Verfahren - einem um Ausgabewerte erweiterten *Neural Gas*-Netzwerk - ist es hingegen nicht nötig, die Kamerakoordinaten mittels eines Stereo-Verfahrens explizit in Weltkoordinaten umzurechnen; die Kalibrierung der Kameras - und ebenso des Robotersystems - erfolgt im Verlauf des Trainings implizit durch das Netzwerk selbst. Spätere kleinere Veränderungen der Position oder Orientierung der Kameras oder der kinematischen Parameter des Robotersystems werden durch die Korrekturbewegungen sofort ausgeglichen und im weiteren Verlauf des Trainings selbsttätig nachkalibriert. Das neurale Netzwerk ist also in der Lage, die oben beschriebene *Hand-Eye*-Transformation ohne Vorwissen über die Position und Orientierung der Kameras oder die Roboterarchitektur ausschließlich aufgrund wiederholter Präsentation von Objekten an

verschiedenen Punkten des Arbeitsbereichs zu erlernen und - bei Veränderung eines oder mehrerer Systemparameter - an die veränderte physikalische Realität anzupassen.

1.2. Überblick

Kapitel 2 widmet sich ausführlich der Theorie und Funktionsweise des im Rahmen der Diplomarbeit eingesetzen erweiterten *Neural Gas*-Netzwerks.

Abschnitt 2.1 führt für die weitere Diskussion wichtige Begriffe und Konzepte ein und behandelt zu diesem Zweck auch einige historische Vorgänger des *Neural Gas*-Verfahrens. Eine kurze Übersicht über Vektorquantisierung und die Beschreibung des *competetive learning* - welches die „Urform" aller neuroinformatischen Ansätze zur Vektorquantisierung darstellt - sind in Abschnitt **2.1.1** und **2.2.2** enthalten. Eng verbunden mit Vektorquantisierungsaufgaben ist das Problem der Bildung topologieerhaltender Karten, welche eine kompakte Repräsentation der topologischen Struktur von Signalverteilungen ermöglichen; während sich Abschnitt **2.2.3** mit der zugrundeliegenden Theorie befaßt, beschreibt Abschnitt **2.2.4** das „klassische" selbstorganisierende Netzwerk-Modell zur Bildung topologieerhaltender Karten, den *Kohonen*-Algorithmus.

Abschnitt 2.2 beschäftigt sich mit dem eigentlichen *Neural Gas*-Modell; während Abschnitt **2.2.1** die einfachste Form des Verfahrens, in welcher es lediglich zur Vektorquantisierung eingesetzt werden kann [Martinetz & Schulten, 1991], beschreibt, stellt Abschnitt **2.2.2** das *competititve Hebbian learning* vor, welches in Verbindung mit dem *Neural Gas*-Vektorquantisierungsverfahren ein außerordentlich leistungsfähiges und dem *Kohonen*-Algorithmus weit überlegenenes Werkzeug zur Bildung topologieerhaltender Karten darstellt. Abschnitt **2.2.3** streift kurz einige neuere, verwandte Verfahren aus dem Bereich der selbstorganisierenden Systeme.

Abschnitt 2.3 hat das im Rahmen der Diplomarbeit eingesetzte, um vektorielle Ausgabewerte erweitere *Neural Gas*-Modell zum Gegenstand. Abschnitt **2.3.1** beschreibt die Architektur des erweiterten Modells, während sich Abschnitt **2.3.2** mit dem auf Fehlerrückkopplung basierenden Lernverfahren auseinandersetzt. Eine Herleitung der Lernregeln wird schließlich in Abschnitt **2.3.2** gegeben.

Kapitel 3 beschreibt den Aufbau des Gesamtsystems sowie dessen einzelne Komponenten und deren Zusammenwirken.

Abschnitt 3.1 befaßt sich mit dem eigentlichen Robotersystem. In Abschnitt **3.1.1** wird ein Überblick über die Hardware des verwendeten Industrieroboters vom Typ A465 gegeben, während sich Abschnitt **3.1.2** mit der Softwareschnittstelle des Robotersystems auseinandersetzt.

Abschnitt 3.2 beschreibt den Aufbau des Bildverarbeitungssystems. Die verwendete Hardware wird in Abschnitt **3.2.1** besprochen; die zur Ermittlung der Effektorposition und der Kalibrierung der Kameras verwendeten Verfahren sind Gegenstand von Abschnitt **3.2.2** und Abschnitt **3.2.3**.

Abschnitt 3.3 geht auf die Details der Implementierung des erweiterten *Neural Gas*-Verfahrens ein. Die gegenüber dem Originalverfahren notwendig gewordenen Modifikationen und Erweiterungen des Verfahrens werden in Abschnitt **3.3.1** besprochen, während Abschnitt **3.3.2** mögliche Fehlerursachen sowie die implementierten Fehlerbehandlungsmechanismen diskutiert.

Kapitel 4 enthält die Beschreibung und Diskussion der mit dem erweiterten *Neural Gas*-Netzwerk durchgeführten Experimente.

Abschnitt 4.1 diskutiert die für die verschiedenen Simulationen verwendeten Parameter und deren Einfluß auf das Lernverhalten.

Abschnitt 4.2 beschäftigt sich mit den verschiedenen Arten von Ausgabefehlern, welche in den nachfolgenden Abschnitten zur Beurteilung der Qualität der vom Netzwerk erlernten Abbildung herangezogen werden, und setzt diese miteinander in Beziehung.

Abschnitt 4.3 präsentiert vier auf einer emulierten Umgebung durchgeführte Softwareexperimente, welche die potentielle Leistungsfähigkeit, aber auch die Grenzen des erweiterten *Neural Gas*-Modells aufzeigen. In Abschnitt **4.3.1** wird das „Referenzexperiment“ beschrieben, welches eine Grundlage für die Beurteilung der Ergebnisse der restlichen Experimente bieten soll. Die in den Abschnitten **4.3.2** und **4.3.3** präsentierten Experimente untersuchen das Verhalten des erweiterten *Neural Gas*-Modells für verschiedene Verteilungen von Eingabesignalen. Abschnitt **4.3.4**

demonstriert die Sensitivität des Netzwerks bezüglich eines seiner Parameter, der Nachbarschaftsreichweite $\lambda(t)$.

Abschnitt 4.4 präsentiert die Resultate des *online*-Experiments, welches auf einer realen Hardwareumgebung durchgeführt wurde.

Abschnitt 4.5 faßt die Resultate der Experimente nochmals kurz zusammen und stellt diese einander gegenüber.

Kapitel 5 enthält eine abschließende Zusammenfassung und Diskussion der Ergebnisse.

2. Das Neural Gas Modell

In diesem Kapitel sollen Aufbau und Funktionsweise des im Rahmen der Diplomarbeit verwendeten neuralen Netzwerks beschrieben werden; es handelt sich um eine um vektorielle Ausgabewerte erweiterte Version des *Neural Gas*-Modells, welche von Thomas Martinetz [Martinetz, 1992] entwickelt wurde.

Das *Neural Gas*-Verfahren eignet sich vorzüglich zur Lösung von Vektorquantisierungsaufgaben [Linde & Buzo & Gray, 1984; Gray, 1991] und zur Erstellung sogenannter **topologischer Karten** [Martinetz & Schulten, 1994]; dies ist Gegenstand von Abschnitt **2.1**, der auch kurz auf einige verwandte Verfahren aus dem Bereich der Neuroinformatik eingeht. Der *Neural Gas*-Basisalgorithmus [Martinetz & Schulten, 1991] ist in Abschnitt **2.2** beschrieben, während das im Rahmen der Diplomarbeit eingesetzte, um Ausgabewerte erweiterte *Neural Gas*-Modell in Abschnitt **2.3** diskutiert wird.

2.1. Grundlagen und verwandte Verfahren

2.1.1. Vektorquantisierung

Die Aufgabe eines Vektorquantisierungsverfahrens besteht darin, eine Teilmenge S des $\Re^n$ durch eine endliche und im Vergleich zu |S| kleine Anzahl von Referenzvektoren $\mathbf{w}_i \in W$, $W \subset \Re^n$, $1 \le i \le m$, zu kodieren. Einem Element $\mathbf{x} \in S$ wird durch die Kodierungsfunktion $\phi: S \to I$, $I = \{1, .., m\}$, jener Referenzvektor $\mathbf{w}_i$ zugeordnet, der $\mathbf{x}$ bezüglich eines gegebenen Abstandsmaßes $d_S(. , .)$ - typischerweise der euklidischen Metrik - am ähnlichsten ist, wodurch $\Re^n$ in m sogenannte Voronoi-Polyhedra oder Diskretisierungsparzellen V_i zerlegt[3] wird:

$$V_i = \{ \mathbf{x} \in \Re^n \mid d_S(\mathbf{x}, \mathbf{w}_i) \le d_S(\mathbf{x}, \mathbf{w}_j), \forall j \in I \setminus \{i\} \} \qquad (2.1)$$

[3] Zu beachten ist, daß es sich bei der durch Gleichung (2.1) gegebenen Zerlegung des $\Re^n$ um keine Partitionierung im streng mathematischen Sinn handelt: es gilt zwar $\cup V_i = \Re^n$, allerdings sind die Mengen V_i - wie direkt aus (2.1) folgt - nicht paarweise disjunkt; man spricht in diesem Zusammenhang von einer Voronoi-Tessellation des Raumes $\Re^n$. Aus demselben Grund ist auch ϕ i.a. nicht eindeutig und somit - streng genommen - keine Funktion.

In Abbildung 2.2 werden eine durch 14 Referenzvektoren gegebene Voronoi-Tessellation des $\Re^n$ sowie der zum Voronoi-Diagramm duale Graph - die Delaunay-Triangulation $\mathbf{D}_{(W)}$ - gezeigt. Letztere erhält man, indem man alle Referenzvektoren, deren zugeordnete Voronoi-Polyhedra aneinandergrenzen, durch eine Kante verbindet. Beide Arten von Graphen werden uns in den folgenden Abschnitten noch häufiger begegnen.

Die Qualität der durch Gleichung (2.1) gegebenen Diskretisierung hängt bei festem m = |W| ausschließlich von der Lage der $\mathbf{w}_i$ ab und läßt sich durch den mittleren Beschreibungsfehler

$$E = \int_S d_S(\mathbf{x}, \mathbf{w}_{\phi(\mathbf{x})})\, p(\mathbf{x})\, d\mathbf{x} \tag{2.2}$$

charakterisieren, wobei $p(\mathbf{x})$ die Dichtefunktion der Signalmenge S beschreibt. Die Aufgabe des Vektorquantisierungsalgorithmus besteht nun darin, die $\mathbf{w}_i$ derart zu wählen, daß E minimal wird.

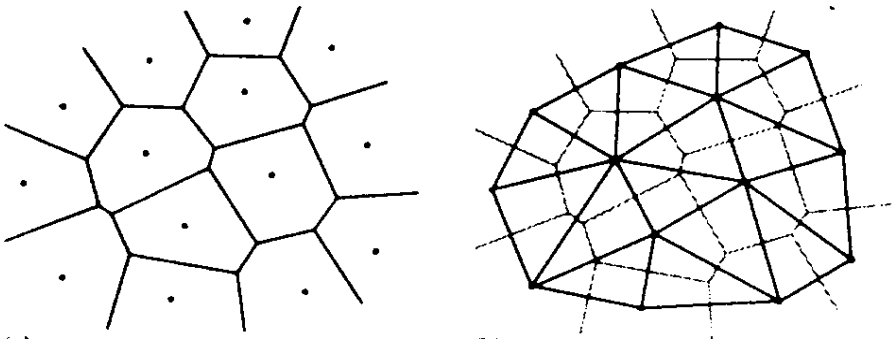

Abb. 2.2: Die linke Abbildung zeigt das Voronoi-Diagramm einer durch 14 Referenzvektoren induzierten Tessellation des $\Re^2$. Die Kanten ergeben sich als Schnittmengen jeweils zweier aneinandergrenzender Voronoi-Polyhedra; die Eckpunkte des Graphen sind genau jene Punkte des $\Re^2$, die im Durchschnitt dreier Voronoi-Polyhedra liegen. Die rechte Abbildung zeigt die zum Voronoi-Diagramm duale Delaunay-Triangulation, in der alle Referenzvektoren, deren zugeordnete Polyhedra aneinandergrenzen, durch eine Kante verbunden sind.

2.1.2. Competitive Learning

Vektorquantisierungsverfahren, die auf neuralen Netzwerken basieren, sind - wie auch der *Neural Gas*-Algorithmus - meist Erweiterungen und Modifikationen des sogenannten *competetive learning*. Obwohl es sich - zumindest in der hier vorgestellten Version - um ein relativ simples Verfahren handelt, ist es doch hervorragend geeignet, die in den folgenden Abschnitten benötigten Konzepte und Ideen auf einfache und anschauliche Weise zu präsentieren, und soll daher an dieser Stelle etwas ausführlicher beschrieben werden. Wir interpretieren dazu die Menge I als Schicht (*layer*) von m Neuronen (*competing units*) und die Referenzvektoren $\mathbf{w}_i$ als Eingangsgewichte dieser *units*; jene *unit* c_i, $1 \leq i \leq m$, deren Gewichtsvektor $\mathbf{w}_i$ dem im *input layer* präsentierten Eingabesignal $\mathbf{x}$ bezüglich der Metrik $d_S(.,.)$ am ähnlichsten ist, gilt als am stärksten aktiviert und somit als Sieger (*winning unit*)[4]; Abbildung 2.3 soll diese Zusammenhänge veranschaulichen.

Der Name des Verfahrens erklärt sich unter anderem daraus, daß nach jeder Präsentation eines Eingabesignals $\mathbf{x}$ nur die *winning unit* eine positive Aktivierung aufweist und die Aktivierungen aller anderen *units* auf 0 fallen; wir haben somit: $\phi(\mathbf{x}) = i$ g.d.w. c_i ist die *winning unit*. Da die Menge aller Eingabesignale $\mathbf{x} \in \Re^n$, für welche die *unit* c_i als Sieger hervorgeht, gleich V_i ist, wird V_i auch als **rezeptives Feld** von c_i bezeichnet. Aufgrund der gefundenen Verteilung der $\mathbf{w}_i$ wird also die Signalmenge S in m Klassen $S_i \subseteq V_i$, $1 \leq i \leq m$, partitioniert, wobei die Klassenzugehörigkeit eines Vektors $\mathbf{x} \in S$ durch das Aktivierungsmuster der *competing units* - hier durch einen 1:m Code - repräsentiert wird.

[4] Die „klassische“ Alternative zur expliziten Verwendung einer Metrik stellt die Berechnung der Netto-Inputs als Inneres Produkt der Eingangsgewichte und des Eingangssignals: $\mathbf{w}_i \cdot \mathbf{x}$ dar; diese Vorgangsweise entspricht zwar eher der Intuition und dem biologischen Vorbild (Gewichtung der Eingangssignale entsprechend der Stärke der jeweiligen präsynaptischen Verbindung und anschließende Summation), hat allerdings den Nachteil, daß die Gewichte während des Trainings ständig renormalisiert werden müssen, um zu verhinderen, daß im weiteren Verlauf einzelne *units* ständig aufgrund der Größe ihres Gewichtsvektors (und nicht aufgrund maximaler Ähnlichkeit mit dem Eingabesignal) gewinnen.

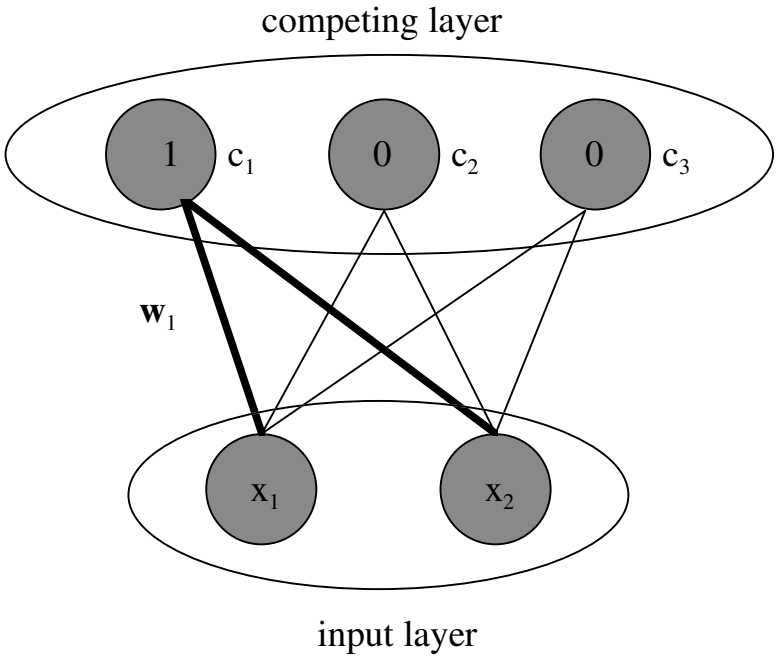

Abb. 2.3: Architektur eines einfachen kompetitiven 2-3 Netzwerks. Die *competing units* c_1, c_2, c_3 sind eingangsseitig über die Gewichtsvektoren $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$ mit den beiden *input units* verbunden; der der *competing unit* c_1 zugeordnete Gewichtsvektor $\mathbf{w}_1 = (w_{11}, w_{12})$ ist fett hervorgeben. Die Aktivierungen der *input units* ergeben sich einfach als Komponenten des Eingabevektors $\mathbf{x} = (x_1, x_2)$. Die *winning unit* c_1 hat ihre Aktivierung auf den Wert 1 gesetzt, die Aktivierungen der anderen *competing units* haben den Wert 0 angenommen (es muß also $ds(\mathbf{w}_1,\mathbf{x}) \leq ds(\mathbf{w}_2,\mathbf{x})$ und $ds(\mathbf{w}_1,\mathbf{x}) \leq ds(\mathbf{w}_2,\mathbf{x})$ gelten).

Beim *competitive learning* fließt keinerlei Information über die Klasseneinteilung der Eingabevektoren ein, es bleibt vielmehr dem Netzwerk selbst überlassen, eine solche zu finden - man spricht in diesem Zusammenhang von **„Lernen ohne Unterweisung“** oder „unüberwachtem Lernen“ (*unsupervised learning*). Unüberwachte Verfahren werden verwendet, um in den Trainingsdaten vorhandene, bisher noch nicht bekannte Strukturen aufzufinden. Soll ein Netzwerk hingegegen eine vorgegebene Ein- Ausgaberelation erlernen, so muß diesem mit jedem Trainingsvektor der gewünschte Sollwert - im konkreten Fall die Klassenzugehörigkeit - präsentiert werden; diese Vorgangsweise bezeichnet man als **„Lernen mit Unterweisung“**[5]. Überwachte Lernverfahren versuchen typischerweise, die Gewichte des Netzwerks im Verlauf des Trainings so zu adaptieren, daß der mittlere quadratische Ausgabefehler (Sollwert - Ausgabewert) minimal wird.

[5] Beispiele für überwachte Lernverfahren sind der bekannte *Backpropagation*-Algorithmus (siehe z.B. [Rumelhart et al., 1986], [Werbos, 1974]) oder das in Abschnitt **2.3** vorgestellte erweiterte *Neural Gas*-Modell.

Wir kommen nun zur algorithmischen Darstellung des *competitive learning*; im einfachsten Fall wird nur der *winning unit* erlaubt, ihre Eingangsgewichte auf den Trainingsvektor hin zu verschieben; diese Vorgangsweise wird als *winner takes all* (*WTA*) bezeichnet:

Algorithmus 2.1: Competetive Learning (WTA)

1. Berechnung der Netto-Inputs
   Setze die Netto-Inputs net_j aller *competing units* c_j gleich $net_j = d_S(\mathbf{x}, \mathbf{w}_j)$.
2. Berechnung der Aktivierungen
   Ermittle die *winning unit* c_i, deren zugeordneter Referenzvektor $\mathbf{w}_i$ dem Eingabesignal $\mathbf{x}$ am ähnlichsten ist, für die also $(\forall j \in I)$ $net_i \leq net_j$ (oder - äquivalent - $\phi(\mathbf{x}) = i$) gilt. Setze die Aktivierung der *winning unit* gleich 1, die aller anderen *units* gleich 0.
3. Lernen
   Verschiebe die Gewichte (den Referenzvektor) der *winning unit* auf das Eingabesignal hin:

$$\mathbf{w}_i^{neu} = \mathbf{w}_i^{alt} + \varepsilon(t) * (\mathbf{x} - \mathbf{w}_i^{alt}), \qquad (2.3)$$

   wobei $\varepsilon(t)$ ein zeitabhängiger Lernparameter (Lernrate) ist.

Die Lernrate $\varepsilon(t)$ wird sinnvollerweise als monoton fallende Funktion der Zeit t (entspricht der Anzahl der durchgeführten Adaptionsschritte) mit Wertebereich]0..1[gewählt. Ein anfänglich hoher Wert von $\varepsilon(t)$ führt dazu, daß sich die Referenzvektoren schnell in einen Bereich mit großer Signaldichte bewegen, wohingegen eine kleine Lernrate eine abschließende Feinjustierung der Referenzvektoren ermöglicht. Ein Beispiel für die zeitliche Entwicklung der Referenzvektoren ist in Abbildung 2.4 zu sehen.

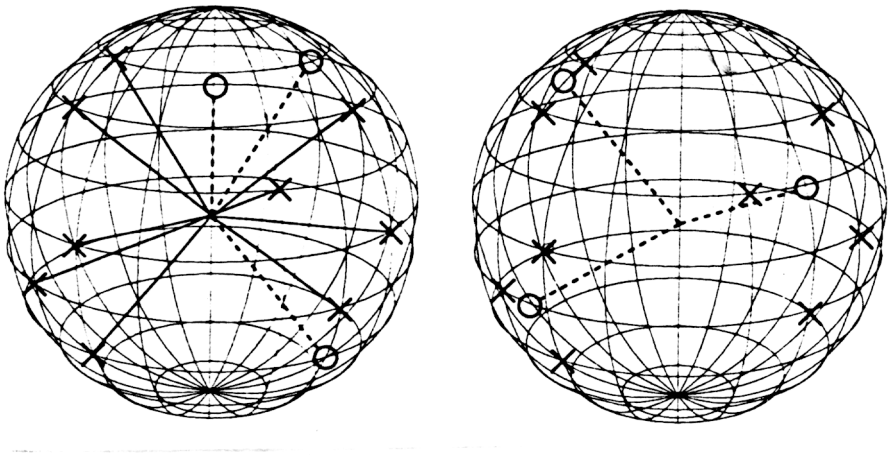

Abb. 2.4: Geometrische Darstellung der Vorgänge beim *competetive learning* (der Anschaulichkeit halber wurden alle Vektoren normiert, sodaß sie auf der Oberfläche einer Kugel mit Radius 1 zu liegen kommen). Trainingsvektoren sind durch ein X, Referenzvektoren durch ein O gekennzeichnet. Die mit zufälligen Werten initialisierten Referenzvektoren (linke Hälfte) bewegen sich im Verlauf des Trainings in Richtung räumlich zusammengehöriger Mustergruppen (rechte Hälfte). In diesem Beispiel wird keine zufriedenstellende Quantisierungsgüte erreichbar sein, da für 4 deutlich unterscheidbare Cluster von Trainingsdaten nur 3 Referenzvektoren zur Verfügung stehen.

Das *competitive learning* in seiner hier dargestellten, einfachsten Form weist allerdings einige Schwächen auf, deren gravierendste ist, daß *units*, deren Referenzvektoren in einem hinreichend großen Bereich mit nicht positiver Signalwahrscheinlichkeit $p(\mathbf{x})$ liegen, niemals gewinnen und daher auch niemals lernen können; diese sogenannten *dead units* stehen also für die Lösung der Vektorquantisierungsaufgabe effektiv nicht mehr zur Verfügung (ϕ ist nicht surjektiv). Es wurden verschiedene Techniken vorgeschlagen, um dieses Problem zu vermeiden (einen Überblick findet man z.B. in [Hertz & Krogh & Palmer, 1991]). Von besonderer Bedeutung ist hier das sogenannte *leaky* oder *soft competetive learning*, welches in jedem Trainingsschritt nicht nur für die *winning unit*, sondern auch für zu dieser benachbarte *units* eine Gewichtsänderung gemäß Gleichung (3) - wenn auch mit deutlich kleinerer Lernrate - durchführt. Den wohl bekanntesten Vertreter dieser Kategorie von Lernverfahren stellt der *Kohonen*-Algorithmus dar, welcher Gegenstand von Abschnitt **2.1.4** ist.

2.1.3. Topologieerhaltende Abbildungen

Mit der obigen Aufgabenstellung eng verbunden ist das Problem der Bildung **topologischer Karten**. Die Idee ist, topologische Beziehungen - sprich Nachbarschaftsrelationen -, welche zwischen den Elementen der Eingabesignalmenge S respektive den Referenzvektoren $\mathbf{w}_i$, $1 \leq i \leq m$, bestehen, auch unter der Abbildung ϕ im Bildraum I zumindest teilweise zu erhalten. Zu diesem Zweck wird I als ungerichteter Graph mit Knotenmenge {1, .., m} aufgefaßt, wobei zwei Elemente i, j, $i \neq j$ (Knoten) aus I genau dann als benachbart gelten, wenn sie durch eine Kante verbunden sind. Die Menge der Kanten wird im folgenden durch die symmetrische m×m-Verbindungsmatrix $\mathbf{A} = (a_{ij})$, $1 \leq i, j \leq m$, dargestellt, wobei für $i \neq j$ gilt: $a_{ij} \in \{0, 1\} = 1$ g.d.w. i und j sind durch eine Kante verbunden[6]. Darüberhinaus ist es sinnvoll, eine Metrik $d_T(. , .)$ auf I einzuführen, welche eine quantitative Aussage über den „Grad" der Nachbarschaft zwischen zwei Elementen i, j erlaubt; wir gehen im folgenden davon aus, daß $d_T(i , j)$ die minimale Pfadlänge zwischen den Knoten i und j angibt und somit insbesondere

$$d_T(i , j) = 1 \Leftrightarrow \text{benachbart}(i, j) \Leftrightarrow a_{ij} = 1, \text{ für } i, j \in I, i \neq j,$$

gültig ist.

Zu klären bleibt noch, unter welchen Voraussetzungen zwei Referenzvektoren $\mathbf{w}_i$, $\mathbf{w}_j \in W$, $i \neq j$[7], als benachbart gelten sollen. Martinetz und Schulten geben in [Martinetz & Schulten, 1994] eine exakte Definition, die hier übernommen werden soll:

Definition 2.1a

Zwei Referenzvektoren $\mathbf{w}_i$, $\mathbf{w}_j \in W$, $i \neq j$, sind genau dann **benachbart in $\Re^n$**, wenn ihre zugeordneten Voronoi-Polyhedra V_i, V_j in der durch die Menge der Referenzvektoren W gegebenen Tessellation des die Signalmenge S einbettenden Raumes $\Re^n$ aneinandergrenzen, sie also in der zur Voronoi-Tessellation dualen Delaunay-Triangulation $\mathbf{D}_{(W)}$ durch eine Kante verbunden sind.

---

[6] Jeder Knoten ist zu sich selbst benachbart; wir klammern diesen trivialen Fall hier jedoch aus, da wir ansonsten ($a_{ii} = 1$) Schleifen im Graphen einführen müßten, was die nachfolgende Diskussion sehr erschweren würde.

[7] Die Nachbarschaftsrelation auf W ist, wie auch auf I, reflexiv, d.h. jeder Referenzvektor ist zu sich selbst benachbart.

Da in praktischen Problemstellungen die topologische Struktur der Signalmenge S (und nicht des einbettenden Raumes) von Interesse ist, ist es jedoch sinnvoll, eine stärkere Bedingung für die Nachbarschaft von Referenzvektoren in S zu fordern. Wir führen dazu zunächst den Begriff des **maskierten Voronoi-Polyhedrons** $V_i^{(S)} = V_i \cap S$ ein; jedes der $V_i^{(S)}$ liegt in S, und die Vereinigung aller $V_i^{(S)}$ ergibt S. Im Gegensatz zu den Voronoi-Polyhedra legen die maskierten Voronoi-Polyhedra also eine Tessellation der Signalmenge S und nicht des gesamten S einbettenden Raumes $\mathfrak{R}^n$ fest. Unter der **induzierten Delaunay-Triangulation** $\mathbf{D}_{(W,S)}$ verstehen wir jenen Graphen, in welchem genau jene Referenzvektoren $\mathbf{w}_i$, $\mathbf{w}_j \in W$ durch eine Kante verbunden sind, deren maskierte Voronoi-Polyhedra $V_i^{(S)}$, $V_j^{(S)}$ aneinandergrenzen ($V_i^{(S)} \cap V_j^{(S)} \neq \emptyset$); die Menge der Kanten der induzierten Delaunay-Triangulation $\mathbf{D}_{(W,S)}$ ist stets eine Untermenge der Kanten der Delaunay-Triangulation $\mathbf{D}_{(W)}$.

Definition 2.1b

Zwei Referenzvektoren $\mathbf{w}_i$, $\mathbf{w}_j \in W$, $i \neq j$, sind genau dann **benachbart in S**, wenn ihre zugeordneten maskierten Voronoi-Polyhedra $V_i^{(S)}$, $V_j^{(S)}$ aneinandergrenzen, sie also in der induzierten Delaunay-Triangulation $\mathbf{D}_{(W,S)}$ durch eine Kante verbunden sind.

Für zwei in S benachbarte Referenzvektoren $\mathbf{w}_i$, $\mathbf{w}_j$, $i \neq j$, gilt somit insbesondere, daß sowohl V_i als auch V_j Elemente aus S enthalten $(V_i^{(S)} \cap V_j^{(S)} \neq \emptyset \Rightarrow (V_i \cap S) \neq \emptyset \wedge (V_j \cap S) \neq \emptyset)$ und die Menge S entlang der gemeinsamen Grenze von V_i und V_j nicht unzusammenhängend ist ($V_i^{(S)} \cap V_j^{(S)} \neq \emptyset \Leftrightarrow (V_i \cap V_j) \cap S \neq \emptyset$). Die eben gegebene Definition für Nachbarschaft läßt sich in natürlicher Weise auf beliebige Elemente aus S erweitern: $\mathbf{u}, \mathbf{v} \in S$ sind benachbart g.d.w. $\mathbf{w}_{\phi(\mathbf{u})}$, $\mathbf{w}_{\phi(\mathbf{v})}$ sind benachbart in S.

Alle in Abbildung 2.5 durch eine Kante verbundenen Referenzvektoren sind im Sinne von Definition 2.1a benachbart; wendet man jedoch die strengere Definition 2.1b an, so ist $\mathbf{w}_4$ in S zu keinem der anderen Referenzvektoren benachbart, da $V_4^{(S)} \cap V_i^{(S)} = \emptyset$ für $i \in \{1, 2, 3\}$. Jene Kanten, welche zur Delaunay-Triangulation $\mathbf{D}_{(W)}$, aber nicht zur induzierten Delaunay-Triangulation $\mathbf{D}_{(W,S)}$ gehören, wurden strichliert dargestellt.

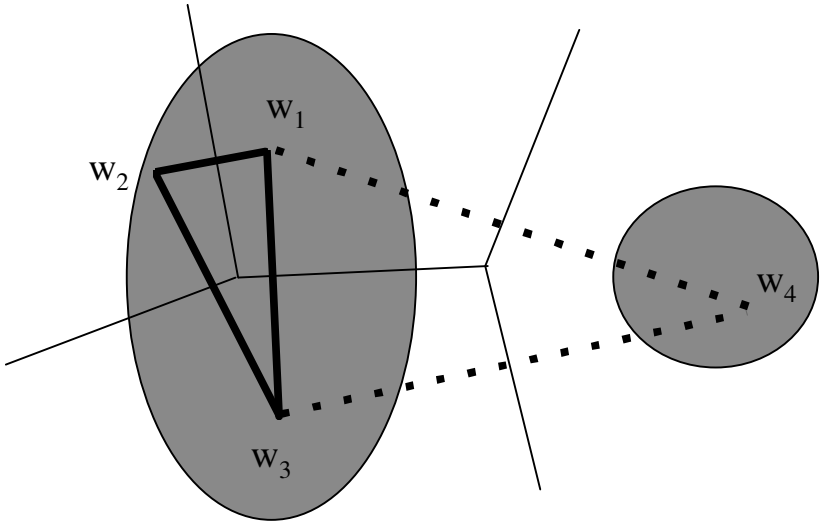

Abb. 2.5: Nachbarschaftsbeziehungen von vier Referenzvektoren im $\Re^2$. Die Signalmenge S setzt sich aus zwei unzusammenhängenden Bereichen (hier durch die beiden grau unterlegten Ellipsen dargestellt) zusammen. Die dünnen Linien gehören zum Voronoi-Diagramm, die dicken - durchgezogenen und strichlierten - Linien entsprechen den Kanten der Delaunay-Triangulation (siehe Text).

Wir kommen nun zu den beiden zentralen Begriffen der **nachbarschaftserhaltenden** respektive der **topologieerhaltenden Abbildung**, welche ebenfalls aus [Martinetz & Schulten, 1994] übernommen wurden (im folgenden verstehen wir unter der Nachbarschaft von Referenzvektoren - sofern nicht anders angegeben - immer deren Nachbarschaft in S, gehen also von Definition 2.1b aus):

Definition 2.2

Die Abbildung ϕ: S $\rightarrow$ I ist **nachbarschaftserhaltend**, wenn benachbarte Elemente aus S auf benachbarte Knoten in I abgebildet werden; ebenso ist die Umkehrabbildung ϕ^{-1}: i $\mapsto \mathbf{w}_i$ **nachbarschaftserhaltend**, wenn sie benachbarte Knoten aus I auf benachbarte Referenzvektoren aus W abbildet.[8]

[8] Im trivialen Fall i = j verhalten sich - unabhängig von der Gestalt der Verbindungsmatrix **A** - sowohl ϕ als auch ϕ^{-1} nachbarschafterhaltend, da wir die Nachbarschaftsrelation auf S und I als reflexiv vorausgesetzt haben und $\phi(\mathbf{w}_i) = \phi(\mathbf{w}_j) = i$ sowie $\phi^{-1}(i) = \phi^{-1}(j) = \mathbf{w}_i$ gilt.

Definition 2.3

Eine nachbarschaftserhaltende ϕ Abbildung ist genau dann **topologieerhaltend**, wenn auch ihre Inverse nachbarschaftserhaltend ist; in diesem Fall wird der Graph I als **topologieerhaltende Karte** von S bezeichnet.[9]

Wir wollen nun Abbildung 2.5 noch einmal im Licht der beiden neuen Definitionen betrachten. Vereinbaren wir, daß der Graph I sowohl die durchgezogenen als auch die strichlierten Kanten enthält - und somit der Delaunay-Triangulation $\mathbf{D}_{(W)}$ der Menge der Referenzvektoren W entspricht[10] -, so ist ϕ nicht **topologieerhaltend**, da ϕ^{-1} z.B. die beiden in I benachbarten Knoten 4 und 1 auf die in S nicht benachbarten Referenzvektoren $\mathbf{w}_4$ und $\mathbf{w}_1$ abbildet und somit nicht **nachbarschaftserhaltend** ist. Enthält I hingegen nur die zur induzierten Delaunay-Triangulation $\mathbf{D}_{(W,S)}$ gehörigen - durchgezogen dargestellten - Kanten, so ist durch ϕ tatsächlich eine **topologieerhaltende Abbildung** von S in I gegeben. Allgemein ist Forderung, daß die Menge der Kanten des Graphen I eine Untermenge der Kanten der Delaunay-Triangulation ist, eine notwendige (aber nicht hinreichende) Bedingung dafür, daß I eine **topologieerhaltende Karte** von S darstellt. Wir geben im folgenden eine notwendige und hinreichende Bedingung (der Beweis folgt im wesentlichen jenem in [Martinetz & Schulten, 1994]):

Theorem 2.1

Sei I ein ungerichteter Graph mit m Knoten (*units*) {1, .., m} und Verbindungsmatrix $\mathbf{A} = (a_{ij})$, $a_{ij} \in \{0, 1\}$. Sei weiters S eine Teilmenge eines n-dimensionalen einbettenden Raumes $\Re^n$ und $W = \{\mathbf{w}_1, .. ,\mathbf{w}_n\}$ eine Menge von Referenzvektoren $\mathbf{w}_i \in \Re^n$, wobei jedes $\mathbf{w}_i$ eineindeutig dem Knoten i zugeordnet ist. I ist eine **perfekt topologieerhaltende Karte** von S g.d.w. I die induzierte Delaunay-Triangulation $\mathbf{D}_{(W,S)}$ ist.

---

[9] Um zu unterstreichen, daß man von der strengeren Definition von Nachbarschaft (Definition 2.1b) ausgeht, wird I auch als **perfekt topologieerhaltende Karte** bezeichnet.

[10] Streng genommen setzen wir nicht $I = (\{1, .., m\}, \mathbf{A})$ und $\mathbf{D}_{(W)}$ gleich, sondern $I' = (\{\mathbf{w}_1, .., \mathbf{w}_n\}, \mathbf{A}')$ und $\mathbf{D}_{(W)}$, wobei $\mathbf{A}' = (a'_{ij})$ wie folgt definiert ist: $a'_{ij} = 1 \Leftrightarrow \phi(\mathbf{w}_i) = i$ und $\phi(\mathbf{w}_j) = j$ sind benachbart in $I \Leftrightarrow a_{ij} = 1$, und somit $\mathbf{A}' = \mathbf{A}$.

Beweis
Ist I die induzierte Delaunay-Triangulation, so sind, wie sich direkt aus der Definition von $\mathbf{D}_{(W,S)}$ ergibt, genau jene Knoten i, j, $i \neq j$, in I benachbart - also durch eine Kante verbunden ($a_{ij} = 1$), für welche gilt, daß die ihren korrespondierenden Referenzvektoren $\mathbf{w}_i$,$\mathbf{w}_j$ zugeordneten maskierten Voronoi-Polyhedra $V_i^{(S)}$, $V_j^{(S)}$ aneinandergrenzen. Somit ist sowohl ϕ als auch ϕ^{-1} nachbarschaftserhaltend, und I ist somit eine **perfekt topologieerhaltende Karte** von S.

Ist I eine **perfekt topologieerhaltende Karte** von S, dann muß sowohl ϕ als auch ϕ^{-1} nachbarschaftserhaltend sein. Dies ist der Fall, wenn in I benachbarte Knoten i, j, $i \neq j$, ($a_{ij} = 1$) auf Referenzvektoren $\mathbf{w}_i$, $\mathbf{w}_j$, deren zugeordnete maskierte Voronoi-Polyhedra $V_i^{(S)}$, $V_j^{(S)}$ aneindergrenzen, abgebildet werden, und in S benachbarte Referenzvektoren $\mathbf{w}_i$, $\mathbf{w}_j$, $i \neq j$, auf in I benachbarte Knoten i, j abgebildet werden. I muß also, wie sich wiederum direkt aus der Definition von $\mathbf{D}_{(W,S)}$ ergibt, die induzierte Delaunay-Triangulation sein.

Die Terminologie auf diesem Gebiet ist leider keineswegs einheitlich. So wird insbesondere der *Kohonen*-Algorithmus immer noch gerne als prototypisches Verfahren zur Erzeugung **topologieerhaltender Karten** zitiert, obwohl die mittels dieses Verfahrens erzeugten Abbildungen i.a. nicht topologie-, sondern nur nachbarschaftserhaltend sind (siehe Abschnitt **2.1.4)**. Wir werden die Menge I daher im folgenden als **topologische Karte** bezeichnen, wenn ϕ oder ϕ^{-1} nachbarschaftserhaltend ist; insbesondere ist somit jede **topologieerhaltende Karte** von S auch eine **topologische Karte** von S.

Topologieerhaltende Karten stellen eine kompakte Repräsentation der topologischen Struktur des Eingabesignalraums S dar und geben somit Aufschluß über die Verteilung wichtiger Signalmerkmale. Die Güte einer solchen Karte hängt klarerweise auch davon ab, wie gut die Referenzvektoren $\mathbf{w}_i$ die Verteilung von S abdecken, so daß ein Verfahren zur Bildung topologieerhaltender Karten auch Vektorquantisierung auf der Menge S durchführen muß. Andererseits läßt sich durch die Berücksichtigung von Nachbarschaftsbeziehungen die mittels eines Vektorquantisierungsverfahrens erzielbare Diskretisierungsgüte und Konvergenzgeschwindigkeit erheblich steigern, wie in Abschnitt **2.1.4** und **2.2** gezeigt werden wird.

2.1.4. Kohonen Algorithmus

Viele der neuralen Strukturen im zerebralen Neokortex von Säugetieren sind im wesentlichen zweidimensional angeordnete Schichten (*layer*) von Verarbeitungseinheiten (*units*), welche durch **laterale Interaktion** miteinander verschaltet sind [Kohonen, 1989]: jede *unit* ist sowohl ein- als auch ausgangsseitig mit den anderen *units* ihrer Schicht verbunden, wobei die Verbindungen innerhalb eines Radius von 50 bis 100 µm anregender, im angrenzenden Bereich, der sich bis zu einem Radius von 200 bis 500 µm erstreckt, jedoch hemmender Natur sind. Im äußeren Bereich ≥ 500 µm sind die synaptischen Verbindungen wieder schwach anregender Natur, was im folgenden jedoch vernachlässigt werden soll.

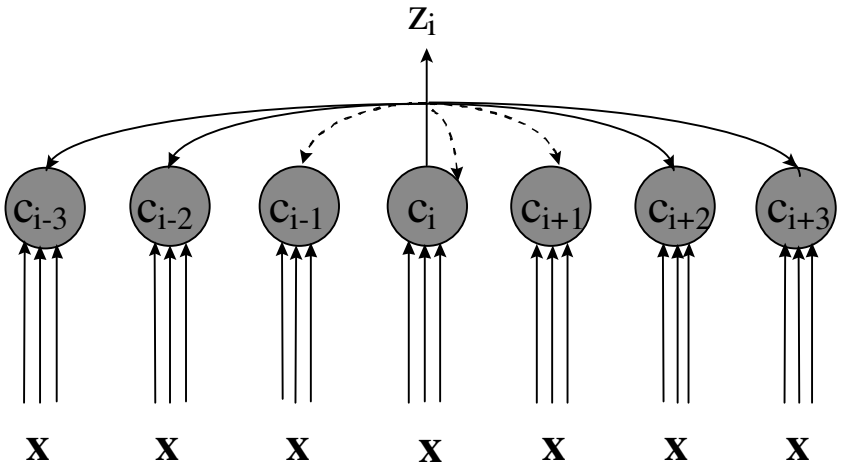

Abb. 2.6: Laterale Interaktion in einer eindimensional angeordneten Neuronenschicht (siehe Text).

Abbildung 2.6 zeigt einen Ausschnitt aus einer eindimensionalen Anordung von *units*, welche alle dasselbe externe Eingabesignal **x** erhalten. Der Übersichtlichkeit halber sind nur die von der *unit* c_i wegführenden lateralen Verbindungen eingezeichnet. Gestrichelte Linien symbolisieren anregende, durchgezogene Linien hemmende Verbindungen; die Stärke der Verbindungen (und somit der **lateralen Interaktion**) hängt von der Distanz zwischen den beiden *units* ab und wird üblicherweise durch die *Mexican Hat* Funktion (siehe Abbildung 2.7) modelliert. Man beachte, daß c_i durch eine anregende Verbindung mit sich selbst verbunden ist. Je größer die Aktivierung z_i von c_i, desto größer ist

klarerweise auch der anregende respektive hemmende Einfluß auf die Aktivierungen der anderen *units*.

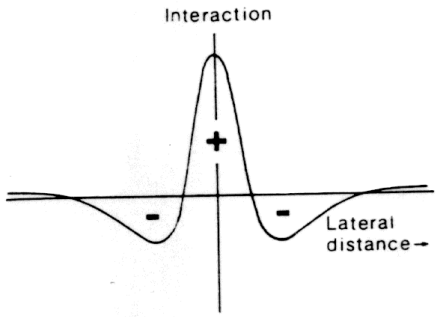

Abb. 2.7: Zur mathematischen Beschreibung der lateralen Interaktion wird meist die *Mexican Hat* Funktion herangezogen. Die Funktion trägt ihren Namen aufgrund ihrer ausgeprägten Änlichkeit mit einem mexikanischen Sombrero.

Erhalten die solcherart innerhalb einer Schicht verschalteten Einheiten nun ein externes Eingangssignal, so nimmt die Netzantwort der Schicht mit fortschreitender Zeit aufgrund des **lateralen Feedbacks** eine höchst lokale Form an: es ensteht ein kreisförmiger Bereich von *units* mit positiven Aktivierungen (*activation bubble*), in dessen Zentrum die aufgrund des externen Inputs am stärksten erregte *unit* liegt; die Aktivierungen aller anderen *units* gehen gegen Null.

Die topologische Anordnung der *units* innerhalb der Schicht ist, wie bereits ausgeführt, zweidimensional, wohingegen das Eingabesignal aus einer Teilmenge eines n-dimensionalen Raums, $n \geq 2$, stammt. Hängt nun der Ort des lokalen Erregungszentrums in stetiger und gesetzmäßiger Weise von einigen wenigen Signalmerkmalen ab, so wirkt die Schicht als **topologische Karte**, welche Aufschluß über die Verteilung dieser Signalmerkmale gibt; es wird also versucht, eine n-dimensionale Verteilung unter möglichst geringem Informationsverlust auf eine 2-dimensional angeordnete Schicht von *units* abzubilden; man spricht daher in diesem Zusammenhang auch von dimensionsreduzierenden **Merkmalskarten** (*feature maps*).

Der von dem Finnen Teuvo Kohonen entwickelte und nach ihm benannte *Kohonen*-Algorithmus [Kohonen, 1989] versucht im wesentlichen, das oben beschriebene biologische Modell nachzuahmen: die einer Menge von *competing units* zugeordneten Referenzvektoren (diese entsprechen hier wiederum den vom externen Eingabesignal zu den *units* hinführenden Eingangsgewichten) sollen dergestalt innerhalb der Signalmenge S angeordnet werden, daß sie - zusammen mit einer auf der Menge der *units* vorgegebenen Topologie - eine **Merkmalskarte** der Eingabesignalmenge S bilden. Im Gegensatz zum biologischen Vorbild erfolgt die Ermittlung der aufgrund des externen Eingangssignals am stärksten erregten *unit* (der *winning unit*) nicht mittels **lateralem Feedback**, sondern durch Berechnung des Abstands zwischen dem Eingabesignalvektor und dem Gewichtsvektor aller *units*; dies entspricht der in Abschnitt **2.1.1** beschriebenen Vorgangsweise.

Vor Beginn des Trainings muß vom Benutzer explizit eine topologische Anordnung der m *competing units* sowie eine zugehörige Metrik $d_T(.,.)$ festgelegt werden, welche die Distanz zwischen zwei *units* im *competing layer* angibt. Die Topologie kann prinzipiell beliebig gewählt werden, muß also nicht unbedingt 2-dimensional sein; in der Praxis werden jedoch zumeist 2- oder 3-dimensionale gitterförmige Anordnungen mit der Taxi-Metrik[11] als zugeordnetem Abstandsmaß verwendet. Man kann sich den *competing layer* als Graphen veranschaulichen, dessen Knoten die *competing units* sind; zwei *units* gelten genau dann als benachbart, wenn sie im Graphen durch eine Kante verbunden sind, wobei alle Kanten als ungerichtet angenommen werden. Die Metrik $d_T(.,.)$ wird normalerweise so gewählt, daß sie die minimale Pfadlänge zwischen zwei *units* angibt und somit für benachbarte *units* einen Wert ≤ 1 annimmt. Abbildung 2.8 zeigt ein Beispiel für einen zweidimensional angeordneten *layer* mit dem für *Kohonen*-Netzwerke typischen „netzähnlichen“ Erscheinungsbild.

Das Lernverfahren unterscheidet sich nun vom *competitive learning* darin, daß nicht nur die *winning unit* c_i, sondern auch alle *units*, die bezüglich $d_T(.,.)$ hinreichend nahe bei c_i liegen, ihre Referenzvektoren auf das Eingabesignal hinbewegen, allerdings mit einer mit zunehmender Distanz stark abfallenden Lernrate; dadurch soll erreicht werden, daß bezüglich $d_T(.,.)$ benachbarten *units* zugeordnete Referenzvektoren bezüglich $d_S(. , .)$ in S ebenfalls benachbart sind oder, anders ausgedrückt, benachbarte *units* ähnliche

[11] Die Taxi-Metrik ist im 2-dimensionalen Fall wie folgt definiert: $d_M((x_1, y_1), (x_2 ,y_2)) = (|x_1 - x_2| + |y_1 - y_2|)$ (analog im 3-dimensionalen Fall); die Menge $\{ \mathbf{x} \mid r \geq d_M(\mathbf{x}, a) \}$ beschreibt also ein Parallelogramm (im 3-dimensionalen Fall ein Oktaeder) mit Seitenlänge $r\sqrt{2}$ und Mittelpunkt a. Die Argumente von $d_M(.,.)$ sind bei array- oder gitterförmiger Anordnung der *units* deren Array-Indizes.

Ausprägungen von Signalmerkmalen kodieren; das Verfahren versucht also, eine **nachbarschaftserhaltende Abbildung** ϕ^{-1}: $i \mapsto \mathbf{w}_i$ von $I = \{1, .., m\}$[12] in die Menge $W = \bigcup_{i=1}^{m} \mathbf{w}_i$ zu finden. Darüberhinaus approximiert die Verteilung der Referenzvektoren die Dichtefunktion der Signalverteilung von S, d.h. Gebiete mit hoher Signaldichte werden i.a. feiner diskretisiert als solche mit niedriger Signaldichte.

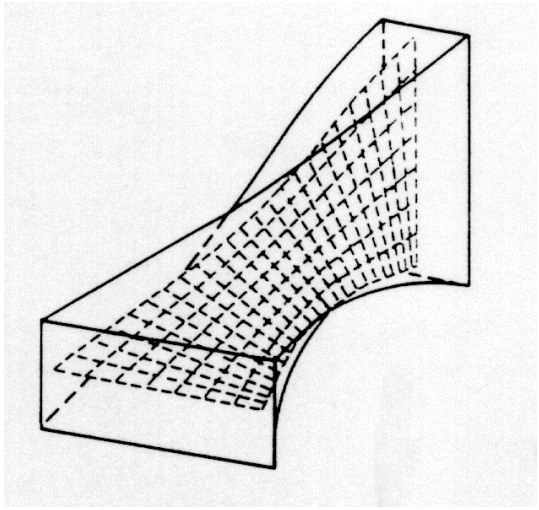

Abb. 2.8: Projektion einer Teilmenge des $\Re^3$ auf einen zweidimensional angeordneten *competing layer*. Die Gitterpunkte entsprechen den *competing units*, benachbarte *units* sind durch eine Kante verbunden. Die einzelnen *units* wurden am Ort ihres zugeordneten Referenzvektors $\in \Re^3$ eingezeichnet.

Nachdem die wichtigsten theoretischen Grundlagen des *Kohohen*-Algorithmus behandelt wurden, soll nun das Verfahren selbst beschrieben werden:

[12] Die Abbildung ϕ^{-1} erfolgt genaugenommen von der Menge der *competing units* in **W**; wir setzen hier allerdings - wie in der Literatur zu diesem Thema durchaus üblich - die *competing units* c_i mit ihrem Index i gleich.

Algorithmus 2.2: Kohonen-Algorithmus

1. Berechnung der Netto-Inputs
   Setze die Netto-Inputs net_j aller *competing units* c_j gleich $net_j = d_S(\mathbf{x}, \mathbf{w}_j)$.

2. Berechnung der Aktivierungen
   Ermittle die *winning unit* c_i, deren zugeordneter Referenzvektor $\mathbf{w}_i$ dem Eingabesignal $\mathbf{x}$ am ähnlichsten ist, für die also $(\forall j \in I)$ $net_i \leq net_j$ (oder - äquivalent - $\phi(\mathbf{x}) = i$) gilt. Setze die Aktivierung der *winning unit* gleich 1, die aller anderen *units* gleich 0.

3. Lernen
   Verschiebe die Gewichte (den Referenzvektor) aller *units* auf das Eingabesignal hin:

$$\mathbf{w}_j^{neu} = \mathbf{w}_j^{alt} + \varepsilon(t) * h(c_i, c_j) * (\mathbf{x} - \mathbf{w}_j^{alt}), \tag{2.4}$$

   wobei $\varepsilon(t)$ wiederum ein zeitabhängiger Lernparameter ist. Die Funktion $h(.,.)$ modifiziert die Lernrate für den Referenzvektor $\mathbf{w}_j$ in Abhängigkeit von der Distanz $d_T(c_j, c_i)$ der *unit* c_j zur *winning unit* c_i; eine beliebte Wahl ist z.B. die Gaussglocke:

$$h(c_i, c_j) = e^{-(d_T(c_j, c_i)^2 / 2*\sigma(t)^2)}. \tag{2.5}$$

   Die Nachbarschaftsreichweite $\sigma(t)$ ist zeitabhängig und konvergiert im Verlauf des Trainings gegen 0; das Verfahren geht dann wieder in das klassische *competitive learning* über, um eine abschließende Feineinstellung der Referenzvektoren zu ermöglichen.

Die Lernrate $\varepsilon(t)$ wählt man - wie beim *competetive learning* - als langsam und monoton fallende Funktion der Zeit mit Wertebereich]0..1[. In der stochastischen Approximation (siehe [Kohonen, 1989], S 71) werden an $\varepsilon(t)$ außerdem die Anforderungen

$$\sum_{t=1}^{\infty} \varepsilon(t) = \infty \text{ und } \sum_{t=1}^{\infty} \varepsilon(t)^2 < \infty.$$

gestellt. Eine Klasse von Funktionen, die diese Forderungen erfüllt, wäre z.B: $\varepsilon(t) \propto t^{-\alpha}$, $\alpha \in]0.5 ..1]$.

Abbildung 2.9 zeigt die zeitliche Entwicklung eines zweidimensional angeordneten *Kohonen*-Netzwerks mit 10×10 *competing units*. Die Eingabesignalmenge S ist ein rechteckiger Ausschnitt des $\Re^2$ mit konstanter Signaldichte. Die Referenzvektoren werden, ausgehend von zufällig gewählten Startwerten (a), im Verlauf des Trainings (b - d) über S verteilt. Konfiguration (d), in welcher das Netzwerk die topologische Struktur von S bereits gut adaptiert hat (der *competing layer* bildet, wie man leicht sieht, eine **topologieerhaltende Karte** von S), wird typischerweise nach einigen 100 Trainingsschritten erreicht.

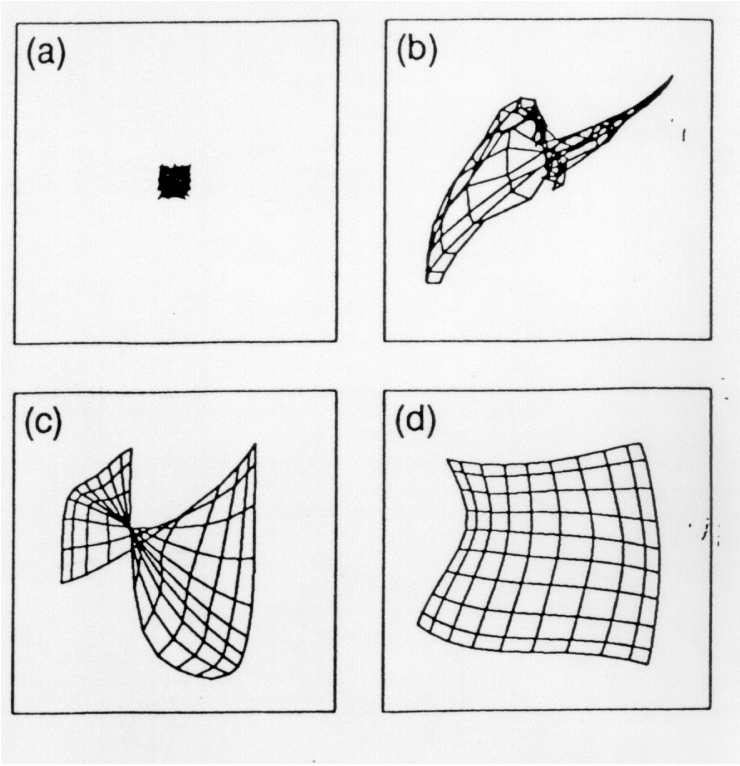

Abb. 2.9: Ein zweidimensional angeordnetes *Kohonen*-Netzwerk adaptiert einen rechteckigen Ausschnitt des $\Re^2$.

Man beachte die Verdrehung in (c): derartige Phänomene treten häufig in der Anfangsphase des Trainings auf; gelingt es im weiteren Verlauf jedoch nicht, diese Verdrehung wieder aufzulösen, so wird das Netzwerk nicht mehr in der Lage sein, eine auch nur annähernd optimale Verteilung der Referenzvektoren (im Sinne von Gleichung (2.2)) oder eine adäquate Beschreibung der topologischen Struktur von S (im Sinne von Definition 2.3) zu finden.

Zu beachten ist, daß die aufgrund der vom *Kohonen*-Algorithmus gefundenen Positionen der Referenzvektoren gegebene Kodierungsfunktion ϕ:S→I i.a. keine **nachbarschafts-erhaltende** und somit insbesondere keine **topologieerhaltende Abbildung** darstellt, da im Sinne von Definition 2.1b benachbarte Referenzvektoren durchaus nicht auf benachbarte *units* abgebildet werden müssen. Betrachtet man Abbildung 2.10, so sieht man, daß der eindimensional angeordnete *competing layer* die Topologie einer zweidimensionalen, homogenen Verteilung natürlich nicht vollständig auf sich abbilden kann; das Verfahren versucht jedoch, die Signalmenge durch eine „flächenfüllende" Kurve so gut wie nur möglich abzudecken.

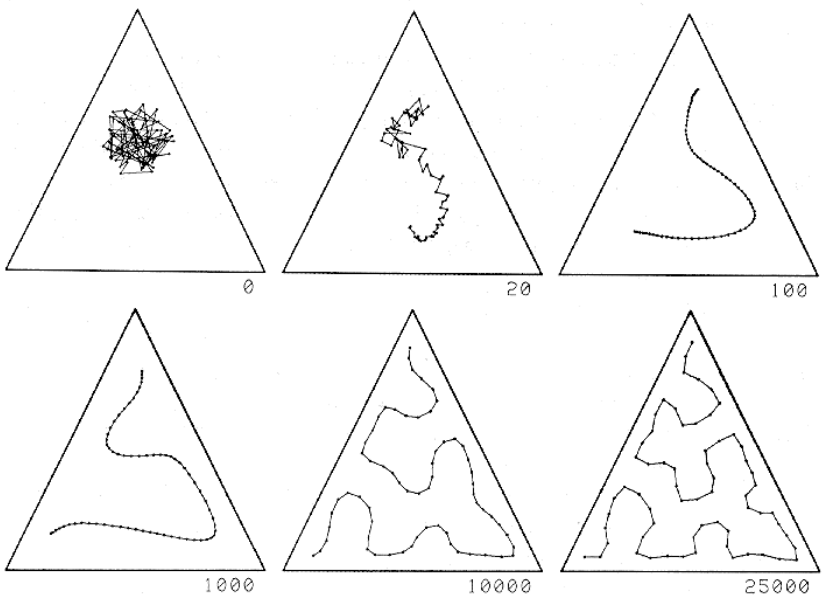

Abb. 2.10: Ein eindimensional angeordnetes *Kohonen*-Netzwerk adaptiert einen dreieckförmigen Ausschnitt des $\Re^2$. Aufgrund des topologischen *mismatch* zwischen der eindimensionalen Struktur des *competing layer* und der zweidimensionalen Struktur der Eingabesignalmenge ist es dem Netzwerk nicht möglich, eine **topologieerhaltende Abbildung** ϕ:S→I zu finden. Die in diesem Beispiel vom *Kohonen*-Algorithmus gefundene inverse Funktion ϕ^{-1}:I→W ist allerdings **nachbarschaftserhaltend**, d.h. benachbarte *units* werden auf benachbarte Referenzvektoren abgebildet.

Allgemein versucht der *Kohonen*-Algorithmus, die Verteilung der Eingangssignalmenge entlang der lokal „interessantesten" Richtungen (das sind jene Richtungen, entlang derer sich die Signalmerkmale am stärksten ändern) zu approximieren.

Es sei an dieser Stelle noch auf ein bemerkenswertes theoretisches Resultat hingewiesen, welches die Fähigkeit von auf dem *Kohonen*-Modell basierenden selbstorganisierenden Systemen illustriert, sich der topologischen Struktur der Eingabesignalmenge anzupassen und in dieser vorhandene Ordnungsrelationen unter der Abbildung auf die Menge der *competing units* zu erhalten: trainiert man ein eindimensionales *Kohonen*-Netzwerk mit Gewichten $\{\mu_1, ..., \mu_m\}$ auf einer Teilmenge des $\Re^1$, so wird, eine geeignete Wahl der Funktion h(.,.) vorausgesetzt, die Gewichtssequenz $<\mu_1, ..., \mu_m>$ in einer endlichen Anzahl von Schritten entweder auf- oder absteigend sortiert. Der Beweis dieses „Ordnungstheorems“ findet sich z.B. in [Kohonen, 1989], S 143 ff.

Obwohl *Kohonens* Verfahren in vielen Fällen gute Ergebnisse liefert, weist es doch einige Schwächen auf:

- Um eine optimale Verteilung der Referenzvektoren im Sinne von Gleichung (2.2) erzielen zu können, muß die topologische Struktur des Eingabesignalraums (bzw. der interessierenden Teilmenge) von vornherein bekannt sein, um die Nachbarschaftstopologie des Netzwerks auf diese abstimmen zu können; weist die Menge der Eingabesignale eine komplexe, inhomogene topologische Struktur auf, ist es auch mit diesem *a priori*-Wissen nicht immer möglich, eine geeignete Netzwerktopologie zu finden.[13]
- Da während des Trainings nicht nur die *winning unit*, sondern auch deren Nachbarn auf das Eingabesignal hinbewegt werden, weist das Verfahren eine höhere Konvergenzgeschwindigkeit auf und ist weit weniger anfällig für *dead units* als das klassische *competetive learning*. Allerdings führen Löcher oder das Vorhandensein unzusammenhängender Teilmengen in der Eingabesignalmenge S im allgemeinen dazu, daß etliche Referenzvektoren in einen Bereich mit nicht positiver Signalwahrscheinlichkeit gezogen werden und diesen nicht wieder verlassen können; Abbildung 2.11 illustriert einen solchen Fall.

[13] Dies muß nicht unbedingt ein Nachteil sein; sucht man eine möglichst einfache Beschreibung der Struktur der Eingabesignalmenge $S \subset \Re^n$, indem man sich auf die Beschreibung der m Richtungen mit dem größten Informationsgehalt beschränkt (wobei m typischerweise einen Wert von 2 oder 3 hat und sehr viel kleiner als n ist), man das Kohonen-Netzwerk also vor allem zur Bildung einer dimensionsreduzierenden Karte verwenden will, kann ein größerer Beschreibungsfehler gemäß Gleichung (2.2) unter Umständen durchaus in Kauf genommen werden.

- Das Netzwerk benötigt zumeist eine relativ lange Entfaltungsphase, bis es die Eingabesignalverteilung zumindest grob abgedeckt hat. Oftmals „verdreht“ sich das Netz in dieser frühen Phase in sich selbst (siehe Abbildung 2.9), so daß es sich auch im weiteren Trainingsverlauf nicht mehr richtig entfalten kann und nur eine suboptimale Verteilung der Referenzvektoren liefert; die Ursache dafür liegt in der statisch vorgegeben Nachbarschaftstopologie des *competing layer*.

Der Kohonenalgorithmus sowie seine biologischen und theoretischen Grundlagen sind ausführlich in [Kohonen, 1989] beschrieben; eine gute Einführung und Beispiele für praktische Anwendungen, insbesondere in der Robotik, bieten [Martinetz, 1992] sowie [Ritter, Martinetz, Schulten, 1991].

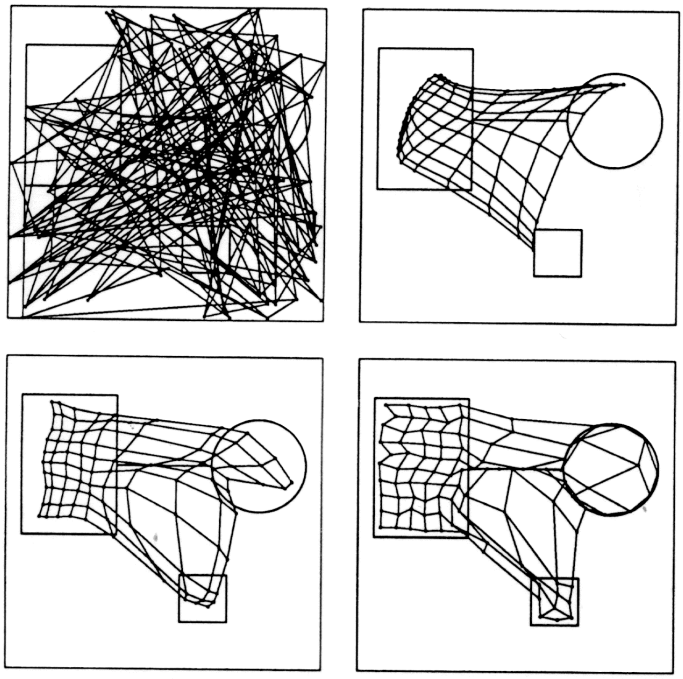

Abb. 2.11: Dargestellt ist die zeitliche Entwicklung eines zweidimensional angeordneten 10x10-*Kohonen*-Netzwerks. Die Signalmenge S zerfällt in zwei unzusammenhängende Bereiche: ein Quadrat und ein Rechteck, welches über eine Linie mit einem Kreisbogen verbunden ist. Darüberhinaus ist die topologische Struktur von S inhomogen: sie enthält sowohl ein- (Linie und Kreisbogen) als auch zweidimensionale (Quadrat und Rechteck) Komponenten. Man bemerkt die große Zahl von Referenzvektoren, die sich auch nach Abschluß des Trainings (unten rechts) in einem Bereich mit nicht positiver Signalwahrscheinlichkeit befinden.

2.2. Neural Gas Basisalgorithmus

Einen neueren Ansatz, der sowohl Elemente des *Kohonen*-Algorithmus als auch des *Simulated Annealing*[14] beinhaltet, aber die mit diesen Verfahren verbundenen Probleme vermeidet, stellt das *Neural Gas*-Verfahren dar. Der *Neural Gas*-Algorithmus kann sowohl ausschließlich zur Vektorquantisierung als auch zur Bildung **topologieerhaltender Karten** eingesetzt werden.

2.2.1. Neural Gas Vektorquantisierungsverfahren

Zunächst wird - wie auch beim *Kohonen-Algorithmus* und beim *competetive learning* - für das aktuelle Eingabesignal $\mathbf{x} \in S$ dessen Abstand zu jedem der m Referenzvektoren $\mathbf{w}_i \in W$, $1 \leq i \leq m$, berechnet, anschließend jedoch nicht nur der Referenzvektor mit maximaler Ähnlichkeit, sondern die Ähnlichkeitsrangfolge aller Referenzvektoren bezüglich $\mathbf{x}$ bestimmt. Die Aktivierungen der *competing units* c_i berechnen sich als monoton fallende Funktion des Ähnlichkeitsrangs ihres zugeordneten Referenzvektors $\mathbf{w}_i$; je ähnlicher also ein Referenzvektor dem Eingabesignal ist, desto größer ist die Aktivierung der korrespondierenden *competing unit*. Während bei den bisher vorgestellten Verfahren nur die Aktivierung der *winning unit* ungleich 0 war, stehen nun die - positiven und voneinander verschiedenen - Aktivierungen aller *units* zur Verfügung; damit ist natürlich auch eine wesentlich feinere Diskretisierung von S - nämlich in m! Diskretisierungspunkte bei m Referenzvektoren - möglich.

In *Kohonens* Modell hängt die Größe eines Adaptionsschritts für jeden Referenzvektor $\mathbf{w}_i$ neben der Lernrate auch von der „Nähe“ der zugeordneten *unit* c_i zur *winning unit* ab, welche aufgrund einer statisch auf dem *competing layer* vorgegebenen Nachbarschaftstopologie und zugehöriger Metrik $d_T(.,.)$ bestimmt wird. Beim *Neural Gas*-Verfahren hingegen werden die für die Größe der Gewichtsupdates maßgeblichen Nachbarschaften zwischen den *competing units* bei jeder Musterpräsentation anhand des Ähnlichkeitsrangs der ihnen zugeordneten Referenzvektoren bezüglich des Eingabevektors $\mathbf{x}$ neu berechnet:

---

[14] *Simulated Annealing* [Kirkpatrick & Vecchi, 1983] wird gerne zur Lösung von Optimierungsproblemen mit vielen lokalen Minima eingesetzt; ein Nachteil ist allerdings die große Zahl benötigter Lernschritte. Die Motivition für diese Kategorie von Verfahren stammt nicht aus der Biologie, sondern aus der statistischen Mechanik (einen Überblick findet man z.B. in [Freeman & Skapura, 1991], S 169 ff.).

Algorithmus 2.3: Neural Gas

1. Berechnung der Netto-Inputs
   Die Distanzen der m Referenzvektoren $\mathbf{w}_j$ zum Eingabevektor $\mathbf{x}$ werden ermittelt und aufsteigend sortiert; der Netto-Input $net_j \in \{0, .., m-1\}$der *unit* c_j errechnet sich als Ähnlichkeitsrang ihres zugeordneten Referenzvektors $\mathbf{w}_j$:

$$W_j = \{ \mathbf{w}_k \in W \setminus \{\mathbf{w}_j\} \mid d_S(\mathbf{w}_k, \mathbf{x}) < d_S(\mathbf{w}_j, \mathbf{x}) \text{ [15]}, 1 \le j \le m \qquad (2.6)$$

$$net_j = rang_{\mathbf{x}}(\mathbf{w}_j) = | W_j |, 1 \le j \le m \qquad (2.7)$$

2. Berechnung der Aktivierungen
   Die Aktivierung z_j der *unit* c_j berechnet sich zu

$$z_j = e^{-net_j / \lambda(t)}, \qquad (2.8)$$

   ist also eine stetige, streng monoton fallende Funktion des Ähnlichkeitsrangs mit Wertebereich [0..1]. Der zeitabhängige Parameter $\lambda(t)$ entspricht der in Algorithmus 2.2 beschriebenen Nachbarschaftsreichweite $\sigma(t)$.

3. Lernen
   Verschiebe die Gewichte (den Referenzvektor) aller *units* auf das Eingabesignal hin:

$$\mathbf{w}_j^{neu} = \mathbf{w}_j^{alt} + \varepsilon(t) * z_j * (\mathbf{x} - \mathbf{w}_j^{alt}), \qquad (2.9)$$

   wobei $\varepsilon(t)$ wieder ein zeitabhängiger Lernparameter ist. Die Lernschrittweite ist außerdem von der Aktivierung der *unit*, z_j, abhängig, welche hier die Rolle der in Gleichung (2.5) eingeführten Skalierungsfunktion $h(.,.)$ übernimmt.

Durch Skalierung der Lernschrittweite mit z_i als streng monoton fallender Funktion des Ähnlichkeitsrangs $rang_{\mathbf{x}}(\mathbf{w}_i)$ fließt in jeden Lernschritt Information über die topologische Struktur von S bzw. W ein; insbesondere gilt für die beiden units c_i, c_j, welche die

[15] Es ist natürlich möglich, daß zwei Referenzvektoren w_i, w_j, $i \neq j$, den gleichen Abstand zum Eingabevektor $\mathbf{x}$ aufweisen. Dieser Fall kann in der Praxis allerdings vernachlässigt werden.

höchsten Aktivierungswerte z_i, z_j aufweisen, daß ihre zugeordneten Referenzvektoren im Sinne von Definiton 2.1b benachbart sind (der Beweis wird weiter unter gegeben).

Der *Neural Gas*-Algorithmus hat gegenüber den bisher vorgestellten Vektorquantisierungsverfahren den Vorteil, daß er auch für unzusammenhängende und topologisch inhomogene Verteilungen der Eingabesignale eine nahezu optimale Verteilung der Referenzvektoren findet (siehe Abbildung 2.12); auch wird keinerlei Initialwissen über die topologische Struktur der Eingabesignalmenge benötigt, was dem Verfahren zusätzliche Attraktivität verleiht.

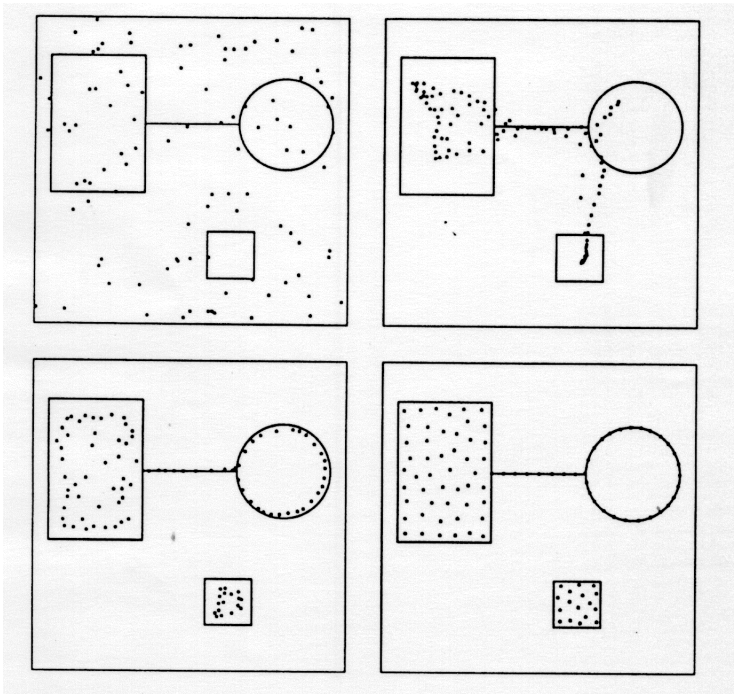

Abb. 2.12: Die bereits aus Abbildung 2.11 bekannte Signalverteilung wird durch ein *Neural Gas*-Netzwerk adaptiert. Im Gegensatz zum *Kohonen*-Netzwerk befinden sich nach Abschluß des Trainings alle Referenzvektoren in einem Bereich mit positiver Signalwahrscheinlichkeit.

2.2.2. Competitive Hebbian Learning

Soll das *Neural Gas*-Netzwerk um die Fähigkeit zur Bildung **topologieerhaltender Karten** erweitert werden, benötigen wir ein Verfahren, um Verbindungen (i.e. Kanten) zwischen genau jenen *units* aufzubauen, deren zugeordnete Referenzvektoren in der Eingabesignalmenge S benachbart sind. Der hier beschriebene Algorithmus beruht auf der bekannten *Hebb*'schen Lernregel

$$\Delta w_{ij} \propto z_i * z_j, \tag{2.10}$$

welche postuliert, daß die Änderung der Stärke einer synaptischen Verbindung zwischen den *units* c_i, c_j proportional zum Produkt ihrer Aktivierungen ist. Allerdings wird in jedem Lernschritt nur das Gewicht zwischen jenen beiden *units* i, j modifiziert, für welche das Produkt $z_i * z_j$ den größten Wert aufweist, d.h. das Verfahren weist auch eine „kompetitive" bzw. WTA-Komponente auf (siehe Abschnitt **2.1.1**). Die Gewichte zwischen den *competing units* repräsentieren wir im folgenden wieder durch die Verbindungsmatrix $\mathbf{A} = (a_{ij})$, $a_{ij} \in \{0, 1\}$. Da die Gewichte (Kanten) zwischen den *units* als ungerichtet angenommen werden, ist A symmetrisch.

Algorithmus 2.4: Competitive Hebbian Learning (CHL)

1. Setze alle $a_{ij} = 0$
2. Wähle mit Wahrscheinlichkeit $p(\mathbf{x})$ einen Vektor $\mathbf{x} \in S$ aus.
3. Bestimme die beiden *units* c_{i_0}, c_{i_1}, deren zugeordnete Refererenzvektoren dem Eingabesignal **x** am ähnlichsten sind:

$$d_S(\mathbf{w}_{i_0}, \mathbf{x}) \leq d_S(\mathbf{w}_j, \mathbf{x}) \; \forall j \in I \setminus \{i_0\}, \tag{2.11}$$

$$d_S(\mathbf{w}_{i_1}, x) \leq d_S(\mathbf{w}_j, \mathbf{x}) \; \forall j \in I \setminus \{i_0, i_1\}. \tag{2.12}$$

   Es gilt also $rang_{\mathbf{x}}(\mathbf{w}_{i_0}) = 0$ und $rang_{\mathbf{x}}(\mathbf{w}_{i_1}) = 1$. Berechnet man die Aktivierungen z_i der *units* - wie z.B. in Gleichung (2.8) - als streng monoton fallende Funktion des Ähnlichkeitsrangs, so gilt klarerweise $z_{i_0} * z_{i_1} \geq zi * z_j \; \forall i, j \in I \setminus \{i_0, i_1\}$.
4. Sind c_{i_0}, c_{i_1},bereits verbunden ($a_{i_0 i_1} \neq 0$), so lasse die Verbindung unverändert, ansonsten setze $a_{i_0 i_1} = 1$.

5. Gehe zurück zu 2.

Wir wissen aus Abschnitt **2.1.3** bereits, daß der *competing layer* I genau dann eine **topologieerhaltende Karte** von S darstellt, wenn seine Verbindungsmatrix **A** jene der induzierten Delaunay-Triangulation ist. Um zu zeigen, daß die mittels CHL erzeugte Verbindungsmatrix für t gegen unendlich (t bezeichne wieder die Anzahl der Lernschritte respektive Musterpräsentationen) diese Bedingung erfüllt, benötigen wir zunächst den Begriff des **Voronoi-Polyhedrons zweiter Ordnung** V_{ij}:

$$V_{ij} = \{ \mathbf{x} \in \Re^n \mid d_S(\mathbf{x}, \mathbf{w}_i) \leq d_S(\mathbf{x}, \mathbf{w}_k) \wedge d_S(\mathbf{x}, \mathbf{w}_j) \leq d_S(\mathbf{x}, \mathbf{w}_k), \forall k \in I \setminus \{i, j\}\}. \quad (2.13)$$

Analog zu den maskierten Voronoi-Polyhedra $V_i^{(S)}$ erhält man **die maskierten Voronoi-Polyhedra zweiter Ordnung** $V_{ij}^{(S)}$ durch Durchschnittsbildung mit S: $V_{ij}^{(S)} = V_{ij} \cap S$. Wie man leicht sieht, verbindet CHL die beiden units c_i, c_j nur dann, wenn $V_{ij}^{(S)} \neq \emptyset$; wird andererseits mit positiver Wahrscheinlichkeit ein Element aus V_{ij} präsentiert, d.h. $\int_{V_{ij}} p(\mathbf{x})d\mathbf{x} > 0$, so wird die Verbindung zwischen c_i und c_j auch aufgebaut.

CHL baut also genau dann eine Verbindung zwischen zwei *units* c_i, c_j auf, wenn das maskierte Voronoi-Polyhedron zweiter Ordnung $V_{ij}^{(S)}$ nicht leer ist. Um zu beweisen, daß der mittels dieses Verfahrens erzeugte Graph tatsächlich die induzierte Delaunay-Triangulation ist, genügt es somit zu zeigen, daß $V_{ij}^{(S)} \neq \emptyset$ g.d.w. $V_i^{(S)} \cap V_j^{(S)} \neq \emptyset$. Dies ist allerdings nur unter der Voraussetzung möglich, daß die Verteilung der Referenzvektoren in S - verglichen mit den strukturellen Details von S - **dicht** ist: betrachtet man Abbildung 2.13, so sieht man, daß $V_{41}^{(S)}$ und $V_{43}^{(S)}$ nicht leer sind und CHL somit auch die strichliert dargestellten Verbindungen, welche nicht zur induzierten Delaunay-Triangulation gehören, aufbauen wird. Plaziert man jedoch einen weiteren Referenzvektor $\mathbf{w}_5$ in der rechten Teilmenge von S (Abbildung 2.14), so gilt $V_{41}^{(S)} = V_{43}^{(S)} = \emptyset$, und es werden keine nicht zur induzierten Delaunay-Triangulation gehörigen Verbindungen mehr aufgebaut; CHL liefert in diesem Fall ein korrektes Ergebnis. Wir vereinbaren, die Verteilung der Referenzvektoren dann als **dicht in S** zu bezeichnen, wenn für alle Punkte $\mathbf{x} \in S$ auch das Dreieck $\Delta(\mathbf{x}, \mathbf{w}_{i_0}, \mathbf{w}_{i_1})$, welches vom Vektor $\mathbf{x}$, dem $\mathbf{x}$ ähnlichsten Refererenzvektor $\mathbf{w}_{i_0}$ und dem $\mathbf{x}$ zweitähnlichsten Referenzvektor $\mathbf{w}_{i_1}$ gebildet wird, in S liegt, also $\Delta(\mathbf{x}, \mathbf{w}_{i_0}, \mathbf{w}_{i_1}) \subseteq S$ gilt.

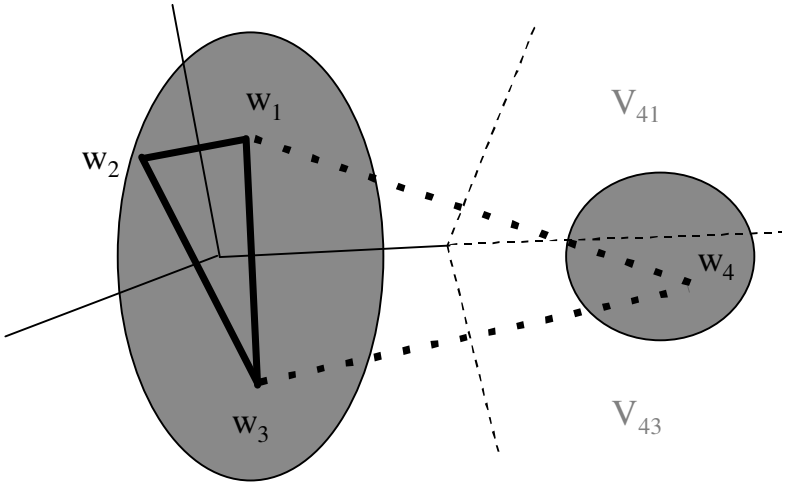

Abb. 2.13: Die bereits aus Abbildung 5 bekannte unzusammenhängende Signalverteilung. Die dünnen, strichliert ausgeführten Linien beschreiben die Grenzen der beiden Voronoi-Poloyhedra zweiter Ordnung V_{41} und V_{43}.

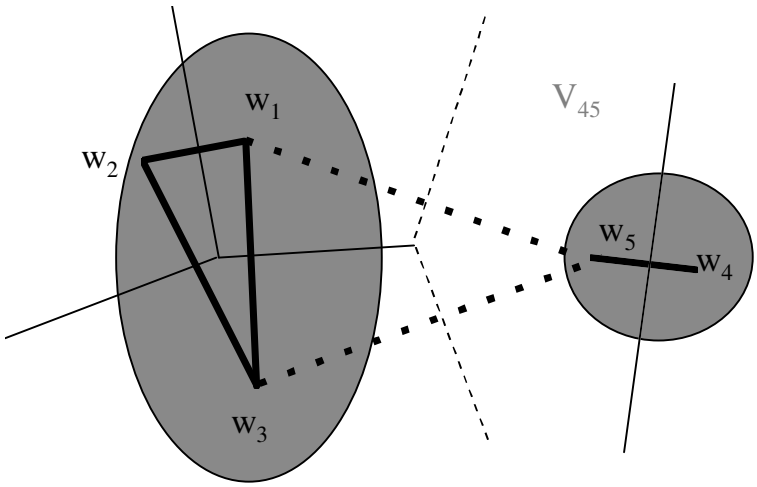

Abb. 2.14: Duch das Hinzufügen von $\mathbf{w}_5$ zur Menge der Referenzvektoren wird deren Verteilung in S **dicht** (siehe Text).

Theorem 2.2

Sei W = {$\mathbf{w}_1$, .., $\mathbf{w}_m$} eine Menge von Referenzvektoren, deren Verteilung auf der Menge S dicht ist. Der Graph, welcher durch CHL auf der Menge der zugeordneten *competing units* I = {c_1, ..c_m} gebildet wird, ist die induzierte Delaunay-Triangulation, und somit eine **topologieerhaltende Karte** von S.

Beweis

Der hier wiedergegebene Beweis wurde [Martinetz & Schulten, 1994] entnommen; wie bereits weiter oben ausgeführt, genügt es zu zeigen, daß $V_i^{(S)} \cap V_j^{(S)} \neq \emptyset$ g.d.w. $V_{ij}^{(S)} \neq \emptyset$.

Ist $V_i^{(S)} \cap V_i^{(S)} \neq \emptyset$, so gibt es ein $\mathbf{v} \in S$ mit $v \in V_i$ und $v \in V_j$. Es muß also $d_S(\mathbf{w}_i, \mathbf{v}) = d_S(\mathbf{w}_j, v) \leq d_S(\mathbf{w}_i, v)$ für $k \in I \setminus \{i, j\}$ und somit auch $\mathbf{v} \in V_{ij}^{(S)}$ gelten.

Ist $V_{ij}^{(S)} \neq \emptyset$, gibt es ein $\mathbf{v} \in S$ für welches - da die Verteilung der Referenzverktoren als dicht angenommen wurde - $\Delta(\mathbf{v}, \mathbf{w}_i, \mathbf{w}_j) \subseteq S$ gilt. Sei nun o.B.d.A. $\mathbf{w}_i$ der $\mathbf{v}$ ähnlichste Referenzvektor. Für jedes $\mathbf{u} \in \overline{\mathbf{w}_j\mathbf{v}} \subseteq S$ ist $\mathbf{w}_j$ der ähnlichste oder zweitähnlichste Referenzvektor; insbesondere ist für $\mathbf{u} = \mathbf{w}_j$ $\mathbf{w}_j$ und für $\mathbf{u} = \mathbf{w}_i$ $\mathbf{w}_i$ der ähnlichste Referenzvektor. Es muß also einen Punkt $\mathbf{u}' \in \overline{\mathbf{w}_j\mathbf{v}}$ geben mit $d_S(\mathbf{w}_i, \mathbf{u}') = d_S(\mathbf{w}_j, \mathbf{u}')$; damit haben wir $\mathbf{u}' \in V_i$, $\mathbf{u}' \in V_j$ und - da aufgrund unserer Voraussetzungen $\mathbf{u}' \in \Delta(\mathbf{v}, \mathbf{w}_i, \mathbf{w}_j) \subseteq S$ gilt - $(V_i \cap V_j) \cap S = V_i^{(S)} \cap V_j^{(S)} \neq \emptyset$.

Es gibt zwei Möglichkeiten, um mittels *Neural Gas*-Algorithmus und CHL **topologieerhaltende Karten** zu erzeugen: entweder man findet zuerst mit Hilfe des *Neural Gas*-Algorithmus eine geeignete Verteilung der Referenzvektoren und bestimmt anschließend mittels CHL die Verbindungen zwischen den *competing units*, oder man führt beide Verfahren parallel aus. Im letzteren Fall müssen jedoch Vorkehrungen getroffen werden, um in einer frühen Phase des Trainings aufgebaute Verbindungen später wieder abtrennen zu können, da ja anfänglich benachbarte Referenzvektoren im Verlauf des Trainings durchaus nicht benachbart bleiben müssen. Dies läßt sich erreichen, indem man die Verbindungen zwischen den *competing units* kontinuierlich altern läßt (*aging*) und die Verbindung zwischen zwei units c_i und c_j wieder auffrischt, falls deren zugeordnete Referenzvekoren die größte Ähnlichkeit zum Eingabesignal $\mathbf{x}$ aufweisen (also $\mathbf{x} \in V_{ij}^{(S)}$ gilt). Überschreitet eine Verbindung ein gewisses Alter - da sie nie aufgefrischt wurde - wird sie schließlich abgetrennt. Gegen Ende des Trainings, wenn sich die Positionen der Referenzvektoren kaum mehr verändern, bleiben schließlich auch die Verbindungen auf dem *competing layer* konstant, es stellt sich ein dynamisches Gleichgewicht ein.

Auf diesem Lernverfahren basierende Netzwerke werden als *topology representing networks* bezeichnet; in der nachfolgenden algorithmischen Darstellung des Verfahrens wird das Alter der Verbindung zwischen den *units* c_i, c_j in der Variablen t_{ij} gehalten, die Verbindungen selbst wieder in der Matrix $A = (a_{ij})$.

Algorithmus 2.5: Topology Representing Network (TRN)

1. Setze alle a_{ij}, $t_{ij} = 0$
2. Wähle mit Wahrscheinlichkeit $p(\mathbf{x})$ einen Vektor $\mathbf{x} \in S$ aus.
3. Führe einen Lernschritt des *Neural Gas*-Verfahrens - wie unter Algorithmus 2.3 beschrieben - durch.
4. Bestimme die beiden *units* c_{i_0}, c_{i_1}, deren zugeordnete Referenzvektoren dem Eingabesignal $\mathbf{x}$ am ähnlichsten sind.
5. Sind c_{i_0}, c_{i_1},bereits verbunden ($a_{i_0 i_1} = 1$), so lasse die Verbindung unverändert, ansonsten setze $a_{i_0 i_1} = 1$.
6. Setze $t_{i_0 i_1} = 0$.
7. Lasse alle Verbindungen altern: setze für alle i,j für welche $a_{ij} = 1$ gilt, $t_{ij} = t_{ij} + 1$.[16]
8. Hat eine Verbindung ihr maximales Alter überschritten, so trenne sie ab: setze für alle c_i, c_j mit $t_{ij} > \text{MAXAGE}$ $a_{ij} = 0$.
9. Zurück zu Schritt 2.

Zum Abschluß soll die zeitliche Entwicklung eines TRN anhand eines Beispiels demonstriert werden. Die in Abbildung 2.15 dargestellte Signalmenge S ist zwar zusammenhängend, aber topologisch inhomogen. Während die Verteilung von S im Verlauf des Trainings in gewohnt bewährter Weise durch den *Neural Gas*-Algorithmus adaptiert wird, bildet CHL gleichzeitig deren topologische Struktur auf den *competing layer* ab. Man sieht deutlich, daß in einer frühen Phase des Verfahrens noch etliche Verbindungen zwischen nicht benachbarten *units* bestehen (links Mitte und links unten, besonders innerhalb des Kreisbogens), diese aber mit zunehmender „Stabilität“ der Referenzvektoren abgetrennt werden, so daß nach Abschluß des Trainings die topologische Struktur vollständig und korrekt auf den *competing layer* abgebildet wird, dieser also eine **topologieerhaltende Karte** von S darstellt.

[16] Unter der Voraussetzung, daß alle *units* im Durchschnitt gleich oft gewinnen, reicht es aus, nur die von der *winning unit* c_{i_0} wegführenden Verbindungen $a_{i_0 j}$ altern zu lassen.

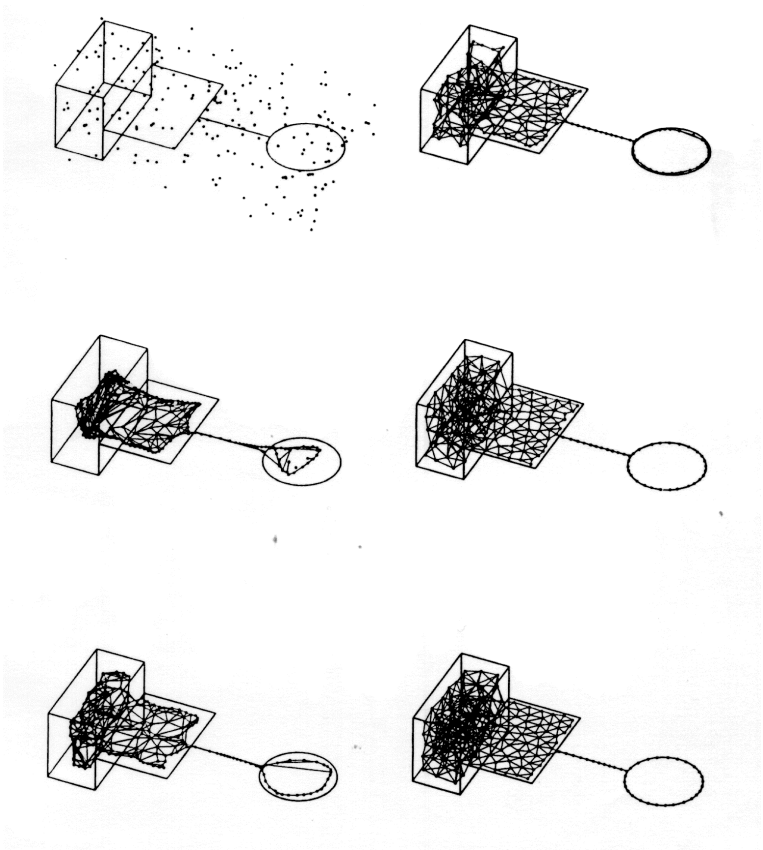

Abb. 2.15: Zeitliche Entwicklung eines TRN. Die topologisch inhomogene Signalmenge S, welche durch das Verfahren adaptiert werden soll, setzt sich aus einem Quader, einem Rechteck, einem Streckensegment und einem Kreisbogen zusammen. Dargestellt sind der Anfangszustand (links oben), der Zustand nach 5000, 1000, 15000, 25000 und schließlich 40000 Schritten (rechts unten).

2.2.3. Verwandte Verfahren

Neural Gas besitzt, wie bereits erwähnt, gegenüber dem *Kohonen*-Modell den Vorteil, daß die topologische Struktur der Eingabesignalmenge S nicht *a priori* bekannt zu sein braucht, und das Verfahren auch bei unzusammenhängenden oder topologisch inhomogenen Verteilungen eine nahezu optimale Verteilung der Referenzvektoren und - bei Verwendung von TRN - eine **topologieerhaltende Karte** der Eingabesignalmenge liefert. Das Verfahren ist in dieser Hinsicht also sehr flexibel, unterliegt aber dennoch einer Beschränkung: die Anzahl der *competing units* muß vor dem Beginn des Trainings statisch festgelegt werden. So führt zum Beispiel bei TRN eine zu kleine Menge von *competing units* (und somit Referenzvektoren) dazu, daß die Verteilung der Referenzvektoren auf der Eingabesignalmenge nicht mehr **dicht** werden kann und somit zwischen einzelnen *units* nicht zur induzierten Delaunay-Triangulation gehörige aufgebaut werden; eine zu große Anzahl von *units* führt hingegen zu unnötig langen Trainingszeiten.

Ähnliche Probleme ergeben sich bei überwachten Lernverfahren (wie bei dem im nächsten Abschnitt behandelten erweiterten *Neural Gas*-Modell): eine zu große Anzahl von *units* veranlaßt das Netzwerk zum „Auswendiglernen" der Trainingsdaten und resultiert in schlechter Generalisierungsfähigkeit; eine zu geringe Anzahl von *units* macht es dem Netz unmöglich, die Struktur der Eingabesignalmenge zu erfassen und führt unter Umständen dazu, daß nicht einmal das Trainingsset erlernt werden kann.

Aus diesem Grund wurden Netzwerkmodelle entwickelt, welche zur Laufzeit bei Bedarf selbständig *units* hinzufügen[17] oder auch entfernen. Im Bereich der selbstorganisierenden Netzwerke spricht man in diesem Zusammenhang von *Growing Self-Organizing Networks* (siehe z.B. [Fritzke, 1997], [Fritzke, 1994]); als Beispiel sei hier das *Growing Neural Gas*-Verfahren erwähnt: dieses ist dem bereits vorgestellten TRN ähnlich, beginnt jedoch lediglich mit zwei *competing units* und fügt in festen Zeitintervallen ein neue *unit* zwischen jener *unit*, welche bisher den größten Diskretisierungsfehler akkumuliert hat, und jenem ihrer direkten topologischen Nachbarn mit dem größten akkumulierten Diskretisierungsfehler ein. *Units*, zu denen keine Verbindung mehr hinführt, werden entfernt. Abbildung 2.16 illustriert das zeitliche Verhalten dieses Netzwerktyps.

[17] Ein bekannter „klassischer" Vertreter dieses Typs von Verfahren ist z.B. der von Fahlman entwickelte *Cascade-Correlation*-Algorithmus [Fahlman & Lebiere, 1990].

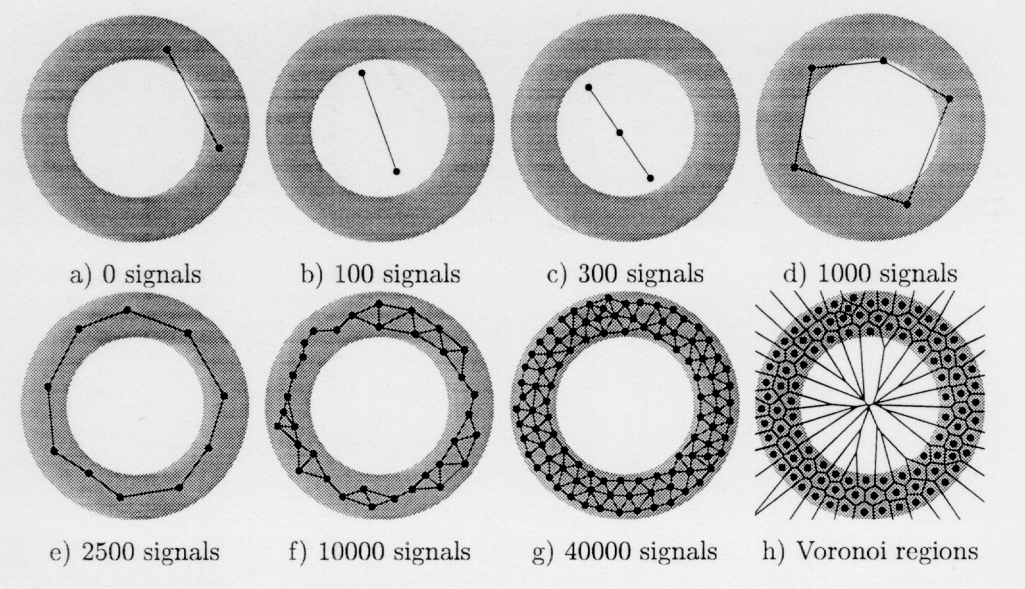

Abb. 2.16: Ein *Growing Neural Gas*-Netzwerk adaptiert eine ringförmige Verteilung; die Eingabesignalmenge S ist innerhalb des dunklen Bereichs gleichverteilt. Das Netzwerk beginnt mit nur 2 *units* (a) und hat nach Durchlaufen der Zwischenzustände (b) bis (f) im Endzustand (g) seine maximale Größe von 100 *units* erreicht. In (h) ist die durch die aufgrund der vom Netzwerk gefundenen Verteilung der Referenzvektoren gegebene Voronoi-Tessellation des $\Re^2$ dargestellt.

Eine genaue Beschreibung des *Growing Neural Gas*-Verfahrens sowie eine Herleitung der Lernregeln findet man in [Fritzke, 1995].

2.3. Erweiterter Neural Gas Algorithmus

Das *Neural Gas*-Verfahren kann um die Fähigkeit zum Erlernen vektorwertiger Abbildungen φ: $\Re^p \rightarrow \Re^q$, $\varphi(\mathbf{u}) = \mathbf{y}$ erweitert werden; es handelt sich dabei um das **überwachte** (*supervised*) Gegenstück zum *Neural Gas*-Vektorquantisierungsverfahren bzw. TRN. Der im nachfolgenden vorgestellte Algorithmus funktioniert auch im Fall allgemeiner Ein- Ausgaberelationen aus $\Re^p \times \Re^q$, wobei lediglich φ^{-1} als bekannt und eindeutig vorausgesetzt wird.

2.3.1. Architektur des erweiterten Neural Gas Modells

Die Durchführung des Vektorquantisierungsschritts im *competing layer* bildet die Grundlage für die Berechnung des Ausgabevektors **y**, wobei wir im folgenden davon ausgehen, daß der *competing layer* n *units* enthält. Die einzelnen Koordinaten y_j von **y** ergeben sich dabei als gewichtete Summe von Abbildungen $\mathbf{A}_{ji}$: $\Re^p \rightarrow \Re$, $1 \leq j \leq q$, $1 \leq i \leq n$

$$y_j = \sum_{i=1}^{n} y_{ji} = z_i * \mathbf{A}_{ji}(\mathbf{u}),\ 1 \leq j \leq q \tag{2.14}$$

Jeder der Abbildungen $\mathbf{A}_{ji}$ ist eine der n *competing units* c_i zugeordnet, und ihr Ergebnis wird mit dem Aktivierungswert z_i dieser *unit* moduliert bzw. gewichtet: je ähnlicher also ein Referenzvektor $\mathbf{w}_i$ dem Eingabevektor **u** ist, desto größer ist gemäß Gleichung (2.8) z_i und somit der Beitrag von $\mathbf{A}_{ji}$ zur Gesamtsumme y_j. Im erweiterten *Neural Gas*-Modell erfolgt die Berechnung der Werte: $z_i * \mathbf{A}_{ji}(\mathbf{u})$ in den sogenannten *modulation units* m_{ij}: dies geschieht durch Bildung des Inneren Produkts des vom *input layer* zu m_{ij} hinführenden Gewichtsvektors $\mathbf{a}_{ji}$ mit **u**, der Addition einer Konstanten α_{ji} (*bias*) und anschließender Multiplikation mit dem Ausgabewert der korrespondierenden *competing unit* z_i[18]. Der *bias* kann dabei als das Gewicht einer mit m_{ij} eingangsseitig verbundenen sogenannten *dummy unit*, deren Aktivierung konstant gleich 1 ist, interpretiert werden.

[18] Werden bei der Berechnung der Ausgangsaktivierung einer *unit* ihre gewichteten Eingangssignale miteinander multipliziert (und nicht, wie üblich, aufaddiert), so spricht man von Sigma-Pi *units*.

Gleichung (2.14) läßt sich somit als

$$y_j = \sum_{i=1}^{n} y_{ji} = z_i * (\mathbf{a}_{ji} . \mathbf{u} + \alpha_{ji}), 1 \leq j \leq q \qquad (2.15)$$

schreiben. Alle *modulation units*, die zur Berechnung von y_j beitragen, werden im j-ten *modulation layer* zusammengefaßt; y_j als Aktivierung der j-ten *output unit* ergibt sich dann einfach als Summe der Aktivierungen der *units* im zugeordneten *modulation layer*. Abbildung 2.17 soll diese Zusammenhänge veranschaulichen.

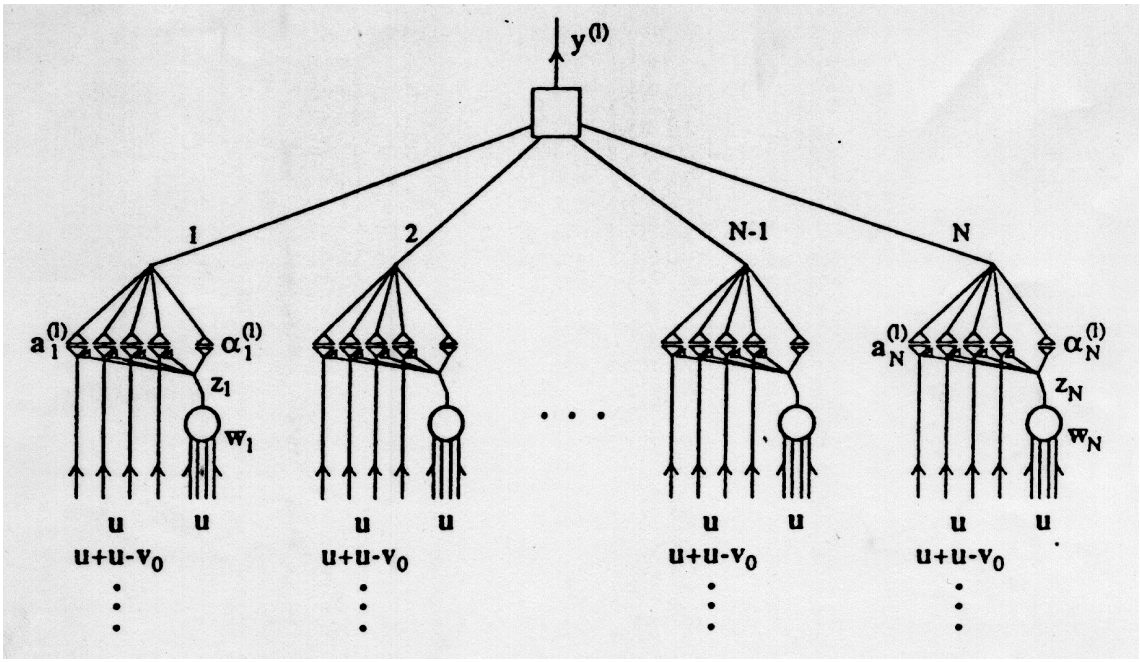

Abb. 2.17: Architektur des erweiterten *Neural Gas*-Modells. Dargestellt ist hier der l-te *modulation layer*. Der *layer*-Index ist im Unterschied zum Text hochgestellt und eingeklammert, d.h. $\mathbf{a}_{li}$ im Text entspricht $\mathbf{a}_i^{(l)}$ in der Graphik. Die *modulation units* befinden sich an den mit 1 bis N gekennzeichneten Punkten in der Mitte der Graphik, die *competing units* sind als Kreise dargestellt. Jede der N *modulation units* m_{li} erhält zunächst das Eingabesignal **u** und berechnet ihre Aktivierung zu $y_{li} = z_i*(\mathbf{a}_{li} . \mathbf{u} + \alpha_{li})$. Die Tatsache, daß die Beiträge der Eingangsleitungen $\mathbf{a}_{li}$ und α_{li} mit der Aktivierung z_i der zugeordneten *competings unit* gewichtet (moduliert) werden, wird durch zwei voneinander wegweisende Pfeilspitzen symbolisiert. Die Aktivierung y_l der l-ten *output unit* ergibt sich als Summe der y_{li}.
Nachdem das Netz den Ausgabewert **y**(0) berechnet hat, wird der Ausgabefehler $\mathbf{u} - \varphi^{-1}(\mathbf{y}(0)) = \mathbf{u} - \mathbf{v}(0)$ berechnet und ins Netz rückgekoppelt; das *Neural Gas*-Netzwerk berechnet nun auf Grundlage des Ausgabefehlers einen neuen Ausgabevektor **y**(1) usw.

2.3.2. Lernen durch Fehlerrückkopplung

Das Lernverfahren für die Ausgabeelemente basiert auf Fehlerrückkopplung und iterativer Verbesserung der Ausgabewerte; bemerkenswert ist, daß - obwohl es sich beim erweiterten *Neural Gas*-Algorithmus um ein überwachtes (*supervised*) Lernverfahren handelt - die zu lernende Funktion φ selbst nicht bekannt oder gar eindeutig sein muß; die Inverse Funktion φ^{-1} hingegen muß als bekannt und eindeutig vorausgesetzt werden.

Nachdem das Netzwerk einen Ausgabevektor $\mathbf{y}(0)$ berechnet hat, wird auf diesen die Inverse der zu lernenden Abbildung angewendet: $\varphi^{-1}(\mathbf{y}(0)) = \mathbf{v}(0)$. Da wir gefordert haben, daß φ^{-1} eindeutig ist, ist $\mathbf{v}(0)$ wohldefiniert, und wir können den Ausgabefehler des Netzwerks mit $\mathbf{u} - \mathbf{v}(0)$ berechnen. Der Ausgabefehler wird nun in das Netzwerk zurückgekoppelt, was konkret durch die Präsentation eines modifizierten Eingabevektors $\mathbf{u} + (\mathbf{u} - \mathbf{v}(0))$ erfolgt, für welchen das Netzwerk einen weiteren Ausgabevektor $\mathbf{y}(1)$ berechnet. Auf Grundlage des neuen Ausgabefehlers $(\mathbf{u} - \mathbf{v}(1))$ (wobei man $\mathbf{v}(1)$ durch Anwendung von φ^{-1} auf den Ausgabevektor $\mathbf{y}(1)$ erhält) wird der Eingabevektor $\mathbf{u} + (\mathbf{u} - \mathbf{v}(0)) + (\mathbf{u} - \mathbf{v}(1))$ berechnet und dieser wiederum dem Netzwerk präsentiert. Diese Rückkopplungsschleife wird mehrere Male durchlaufen, wobei die genaue Anzahl der Iterationen applikationsabhängig ist.

Der Sinn der Rückkopplung liegt darin, die Ausgabewerte für den Eingabevektor $\mathbf{u}$ möglichst rasch - und nicht erst nach längerem Training mit typischerweise mehreren hundert Musterpräsentationen - in die Nähe eines Lösungsvektors $\mathbf{y}$ mit $\varphi^{-1}(\mathbf{y}) = \mathbf{u}$ zu bringen. Um zu einer intuitiven Vorstellung von der Leistungsfähigkeit des oben beschriebenen Rückkopplungsmechanismus zu gelangen, nehmen wir an, daß der Ausgabefehler unabhängig vom Wert des Eingabevektors immer $\mathbf{u} - \mathbf{v}(0)$ beträgt. In diesem Fall gilt für $\mathbf{v}(1) = \varphi^{-1}(\mathbf{y}(1))$:

$$\mathbf{v}(1) + (\mathbf{u} - \mathbf{v}(0)) = \mathbf{u} + (\mathbf{u} - \mathbf{v}(0)),$$

woraus aber sofort $\mathbf{u} = \mathbf{v}(1)$ folgt und das Netzwerk somit bereits nach der ersten Iteration den gesuchten Ausgabewert $\mathbf{y} = \mathbf{y}(1)$ gefunden hätte. Auch wenn die idealisierte Annahme eines konstanten Ausgabefehlers in der Praxis wohl kaum aufrechtzuerhalten sein dürfte, so führt doch die Präsentation des modifizierten Eingabevektors unter den im folgenden gegebenen Voraussetzungen zumindest zu einer Verringerung des Ausgabefehlers.

Lemma 2.1:

Sei $\mathbf{J}\varphi$ die Jakobimatrix von φ,

$$\mathbf{J}_{NG} = \left(\sum_{i=1}^{n} z_i * \mathbf{a}_{1i} , ..., \sum_{i=1}^{n} z_i * \mathbf{a}_{qi} \right)^T$$

die Jakobimatrix der vom *Neural Gas*-Netzwerk berechneten Abbildung NG und $\Delta \mathbf{J}_{NG} = \mathbf{J}_{NG} - \mathbf{J}\varphi$, so ist die positive Definitheit der Matrix $(\mathbf{J}\varphi - \Delta \mathbf{J}_{NG})^T(\mathbf{J}\varphi + \Delta \mathbf{J}_{NG})$ eine hinreichende Bedingung dafür, daß sich der Ausgabefehler $\mathbf{u} - \mathbf{v}(k)$ des Netzwerks mit jedem Iterationsschritt k verringert.

Anschaulich formuliert, fordert Lemma 2.1, daß der Approximationsfehler $\Delta \mathbf{J}_{NG}$ an der Stelle $\mathbf{u}$ kleiner ist als die Matrix $\mathbf{J}\varphi$ selbst. Der Beweis des Lemmas findet sich in [Martinetz, 1992].

Es wurde weiter oben festgestellt, daß die zu lernende Abbildung φ nicht eindeutig zu sein braucht. Werden dem Netzwerk jedoch zwei in S benachbarte Eingabevektoren $\mathbf{u}_1$, $\mathbf{u}_2$ präsentiert, so werden jene *units* c_i, die bei der Präsentation von $\mathbf{u}_1$ hohe Aktivierungswerte z_i aufweisen, dies auch bei der Präsentation von $\mathbf{u}_2$ tun, das heißt die Abbildung von S auf die Menge der *competing units* verhält sich im weiteren Sinne **nachbarschaftserhaltend**.[19] Diese Tatsache ermöglicht es, die Lernregeln der - für die Generierung der Ausgabewerte y_j zuständigen - Gewichte $\mathbf{a}_{ji}$, α_{ji} so zu wählen, daß, ausreichendes Training vorausgesetzt, in S benachbarte Eingabevektoren auf in $\Re^q$ benachbarte Ausgabevektoren abgebildet werden, die vom Netzwerk gefundene Abbildung NG also quasi **stetig** ist.

[19] Die Umkehrung ist i.a. ebenfalls richtig (d.h. ähnliche Aktivierungsmuster kodieren benachbarte Signalvektoren); da die Topologie der Eingabesignalmenge vom erweiterten *Neural Gas*-Verfahren jedoch nicht explizit berücksichtigt wird und in jedem Lernschritt alle Ausgabegewichte $\mathbf{a}_{ji}$, αji modifiziert werden (siehe nächste Seite), können im Fall unzusammenhängender oder „löchriger" Eingabesignalverteilungen Probleme auftreten (siehe Abschnitt **4.3.3**).

Algorithmus 2.6: Erweitertes Neural Gas Verfahren

1. Präsentiere dem Netzwerk den Eingabevektor $\mathbf{u} \in \Re^p$
2. Führe den Lernschritt für den *competing layer* wie in **2.1.3** beschrieben durch.
3. Berechne den Ausgabevektor $\mathbf{y}(0) \in \Re^q$ gemäß Gleichung (2.15).
4. Berechne $\varphi^{-1}(\mathbf{y}(0)) = \mathbf{v}(0) \in \Re^p$.
5. Setze Zähler k = 1.
6. Präsentiere dem Netzwerk den Differenzvektor $\mathbf{u}$ - $\mathbf{v}$(k - 1).
7. Berechne einen neuen Ausgabevektor $\mathbf{y}(k) = (y_1(k), y_2(k), ..., y_q(k))^T$ zu:

$$y_j(k) = y_j(k-1) + \sum_{i=1}^{n} z_i * (\mathbf{a}_{ij} . (\mathbf{u} - \mathbf{v}(k-1)), \quad 1 \leq j \leq q \text{ [20]} \qquad (2.16)$$

8. Berechne $\varphi^{-1}(\mathbf{y}(k)) = \mathbf{v}(k) \in \Re^p$.
9. Verwende die beiden Tupel ($\mathbf{y}$(k), $\mathbf{v}$(k)) und ($\mathbf{y}$(k-1), $\mathbf{v}$(k-1)), um die Inkremente für die zu den *modulation units* hinführenden Gewichte zu ermitteln.

$$\Delta \mathbf{a}_{ji} = z_i * [(\mathbf{y}_j(k) - \mathbf{y}_j(k-1) - \sum_{i=1}^{n} z_i * (\mathbf{a}_{ji} . \mathbf{u})] . (\mathbf{v}(k) - \mathbf{v}(k-1)), \qquad (2.17)$$

$$\Delta \alpha_{ji} = z_i * [(\mathbf{y}_j(k) - \mathbf{y}_j(k-1)]. \qquad (2.18)$$

10. Adaptiere die Gewichte:

$$\mathbf{a}_{ji}^{neu} = \mathbf{a}_{ji}^{alt} + \varepsilon'(t) * \Delta \mathbf{a}_{ji}, \qquad (2.19)$$

$$\alpha_{ji}^{neu} = \alpha_{ji}^{alt} + \varepsilon'(t) * \Delta \alpha_{ji}. \qquad (2.20)$$

11. Wenn die maximale Anzahl von Iterationen erreicht ist: breche ab; sonst erhöhe k um 1 und gehe zurück zu 6.

[20] Dies entspricht dem Anlegen eines „neuen“ Eingabevektors $\mathbf{u} + \mathbf{u} - \mathbf{v}(0) + ... + \mathbf{u} - \mathbf{v}(k-1)$. Zu beachten ist jedoch, daß die Ausgabewerte z_i der *competing units* nur zu Beginn - also bei der Präsentation des Originalvektors $\mathbf{u}$ - berechnet und im weiteren Verlauf der Rückkopplung konstant gehalten werden.

2.3.3. Herleitung der Lernregeln

Die Gewichtsdeltas für die $\mathbf{a}_{ji}$ erhält man durch Gradientenabstieg auf der quadratischen Kostenfunktion

$$E_a = 0.5 * \sum_{j=1}^{q} [(\mathbf{y}_j(k) - \mathbf{y}_j(k-1)) - \sum_{i=1}^{n} z_i * (\mathbf{a}_{ji} . (\mathbf{v}(k) - \mathbf{v}(k-1)))]^2. \quad (2.21)$$

zu

$$\Delta \mathbf{a}_{ji} = -\partial E_a / \partial \mathbf{a}_{ji}. \quad (2.22)$$

Man sieht leicht, daß die rechten Seiten der beiden Bestimmungsgleichungen (2.17) und (2.22) identisch sind. Sei im folgenden wiederum $\mathbf{J}_\varphi$ die Jakobimatrix von φ und $\mathbf{J}_{NG}$ die Jakobimatrix der vom *Neural Gas*-Netzwerk berechneten Abbildung. Man beachte, daß in Gleichung (2.21) $\mathbf{y}_j(k) = \varphi_j(\mathbf{v}(k)$ und $\mathbf{y}_j(k-1) = \varphi_j(\mathbf{v}(k-1))$ gilt; sind nun die Abweichungen $\mathbf{u} - \mathbf{v}(k)$, $\mathbf{u} - \mathbf{v}(k-1)$ gering, so gilt in guter Näherung $(\mathbf{y}_j(k) - \mathbf{y}_j(k-1)) = \mathbf{J}_\varphi|\mathbf{u} * (\mathbf{v}(k) - \mathbf{v}(k-1))$. Durch die Wahl von E_a als Kostenfunktion wird also gefordert, daß die Jakobimatrix der vom *Neural Gas*-Netzwerk gelernten Abbildung die Jakobimatrix von φ an der Stelle $\mathbf{u}$ adaptiert: $\mathbf{J}_\varphi|\mathbf{u} = \mathbf{J}_{NG}|\mathbf{u}$.

Analog ergibt sich die Änderung der additiven Konstanten α_{ji} durch Gradientenabstieg auf der quadratischen Kostenfunktion

$$E_\alpha = 0.5 * \sum_{j=1}^{q} [(\mathbf{y}_j(k) - \mathbf{y}_j(k-1)]^2$$

$$= 0.5 * \sum_{j=1}^{q} [(\mathbf{y}_j(k) - \sum_{i=1}^{n} z_i * \alpha_{ji} + z_i * \mathbf{a}_{ji} . (\mathbf{u} + \mathbf{u} - \mathbf{v}(0) + ... + \mathbf{u} - \mathbf{v}(k-2))]^2 \quad (2.23)$$

zu

$$\Delta \alpha_{ji} = -\partial E_\alpha / \partial \alpha_{ji} = z_i * [(\mathbf{y}_j(k) - \mathbf{y}_j(k-1)] \quad (2.24)$$

E_α fordert, daß die Abweichungen zwischen aufeinanderfolgenden Ausgabewerten minimal sein sollen und - insbesondere - bereits der erste Ausgabewert $\mathbf{y}(0)$ eine gute Annäherung an $\varphi(\mathbf{u})$ darstellt.

3. Aufbau des Systems

Der grundlegende Aufbau des Systems wurde bereits in Kapitel 1 skizziert und soll hier ausführlich beschrieben werden. Der Arbeitsbereich des Roboters wird durch zwei festmontierte CCD-Kameras beobachtet, welche ausgangsseitig mit einem PC, der über eine mit einem *frame grabber* ausgestattete Bildverarbeitungskarte verfügt, verbunden sind. Die Ansteuerung des Roboters erfolgt ebenfalls vom PC aus. Das erweiterte *Neural Gas*-Netzwerk selbst wurde mit Hilfe des Simulationstools XERION unter UNIX entwickelt und läuft auf einer DEC-Workstation. Abbildung 3.1 gibt eine Übersicht über die einzelnen Systemkomponenten.

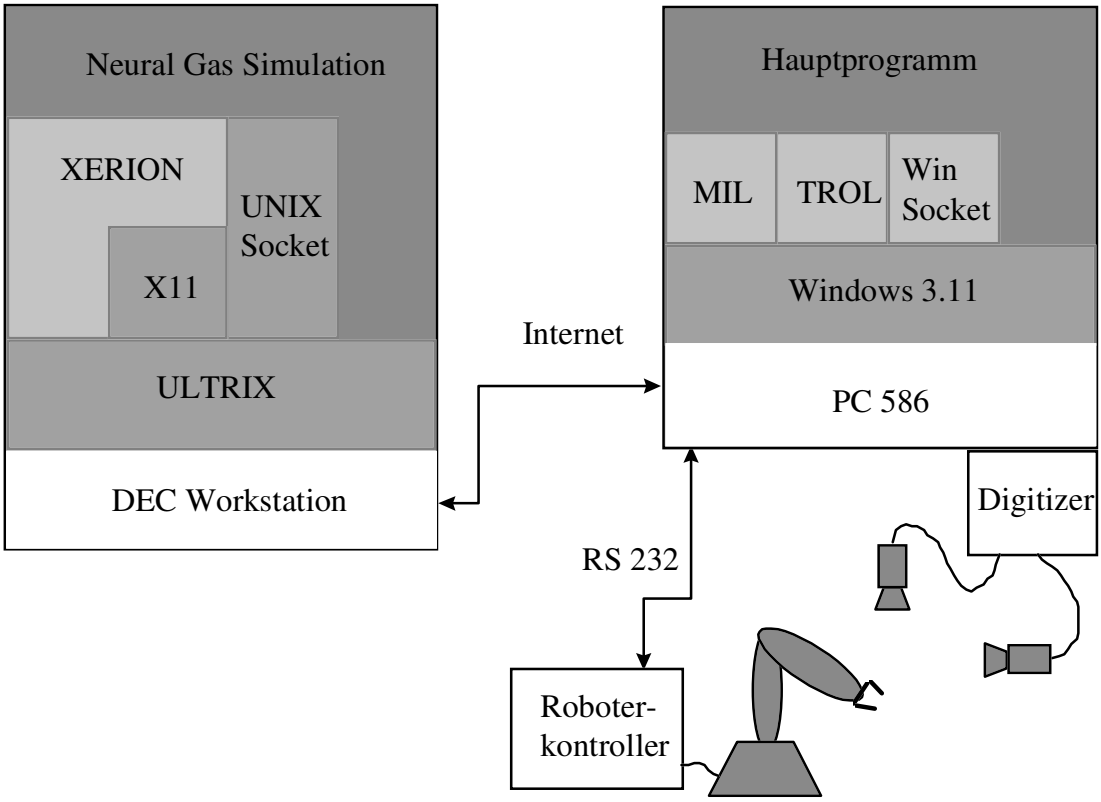

Abb. 3.1: Darstellung der verwendeten Hard- und Softwarekomponenten. Der Hauptmodul, welcher die gesamte Ablauflogik steuert, läuft auf einem Windows-PC. Die Ansteuerung des Roboters erfolgt über die serielle Schnittstelle mittels der am PRIP entwickelten TROL-Library; die Schnittstelle zur Bildverarbeitungshardware bildet die MIL-Library, welche auch einige grundlegende Bildverarbeitungsoperationen zur Verfügung stellt. Zur Kommunikation mit der DEC-Unix-Workstation werden Sockets verwendet. Das *Neural Gas*-Netzwerk wurde mit dem für X11 verfügbaren Simulationstool XERION implementiert; der Simulator wartet, bis der Hauptmodul die Stereo-Bildkoordinaten des anzufahrenden Objekts übermittelt und übernimmt anschließend für die Dauer der Effektorpositionierung die Ablaufkontrolle.

Um das erweiterte *Neural Gas*-Netzwerk mittels des in Abschnitt **2.3** vorgestellten Verfahrens die **inverse Kinematik** des Robotersystems lernen zu lassen, müssen diesem einige hundert Stereobildkoordinaten von im Arbeitsbereich gelegenen Zielpunkten als Trainingsvektoren präsentiert werden. Nach der Präsentation des Zielkoordinatenvektors $\mathbf{u} \in \mathfrak{R}^4$ generiert das Netzwerk zunächst einen vorläufigen Gelenkwinkelvektor $\mathbf{y}(0) \in \mathfrak{R}^3$, mit welchem der Roboter angesteuert wird. In regelungstechnischer Terminologie handelt es sich bei diesem ersten Positioniervorgang um eine **open loop**-Positionierung, da noch keine Information über die Regelabweichung (Ausgabefehler) in die Berechnung der Ausgabewerte einfließt. Anschließend wird die Position des Endeffektors $\mathbf{v}(0) \in \mathfrak{R}^4$ aus beiden Kamerabildern extrahiert und der Ausgabefehler $\mathbf{u} - \mathbf{v}(0)$ ins Netzwerk zurückgekoppelt, woraufhin dieses einen neuen Ausgabewert $\mathbf{y}(1)$ generiert und eine erste Korrekturbewegung des Roboterarms veranlaßt usw; diese Feedback-Schleife wird wiederholt durchlaufen, bis ein vorgegebenes Abbruchkriterium (z.B. minimale Positioniergenauigkeit) erreicht wurde.

Abbildung 3.2 soll die regelkreisähnliche Arbeitsweise des Systems nochmals illustrieren. Man bemerkt, daß die Zielpostion **u** nur einmal, und zwar vor der Durchführung der **open loop**-Positionierung, ermittelt wird; es muß also gewährleistet sein, daß sich die Position des anzufahrenden Objekts während des gesamten Positioniervorgangs nicht verändert; insbesondere ist das Verfahren nicht geeignet, um den Effektor sich bewegenden Objekten nachzuführen.[21] Wichtig ist auch, sich klarzumachen, daß es sich bei der hier vorgestellten adaptiven Robotersteuerung um kein Echtzeitsystem handelt, d.h. zwischen den einzelnen Arbeitsschritten können theoretisch beliebig große Zeitspannen liegen. Dies betrifft insbesondere die Bildverarbeitung: die Greiferposition wird nicht getrackt, sondern erst **nach** Abschluß der Armbewegung in beiden Kamerabildern neu ermittelt, d.h. es muß keine minimale Framerate gefordert werden.

Um die Position des Greifers in beiden Kamerabildern bestimmen zu können, wurde die Greiferspitze mit einer LED markiert. Dies macht es auch möglich, durch konventionelle Ansteuerung eines Punkts im Arbeitsbereich und anschließende Ermittlung der LED-Position auf einfache Weise Trainingsdaten für das Netzwerk zu generieren, welche mit Sicherheit im Arbeitsbereich des Roboters liegen.

[21] Diese Einschränkung ergibt sich im wesentlich daraus, daß die Gelenkwinkel für die Korrekturbewegungen anhand einer Näherung ersten Grades an der Stelle **u** berechnet werden (Gleichung 2.16, Algorithmus 2.6).

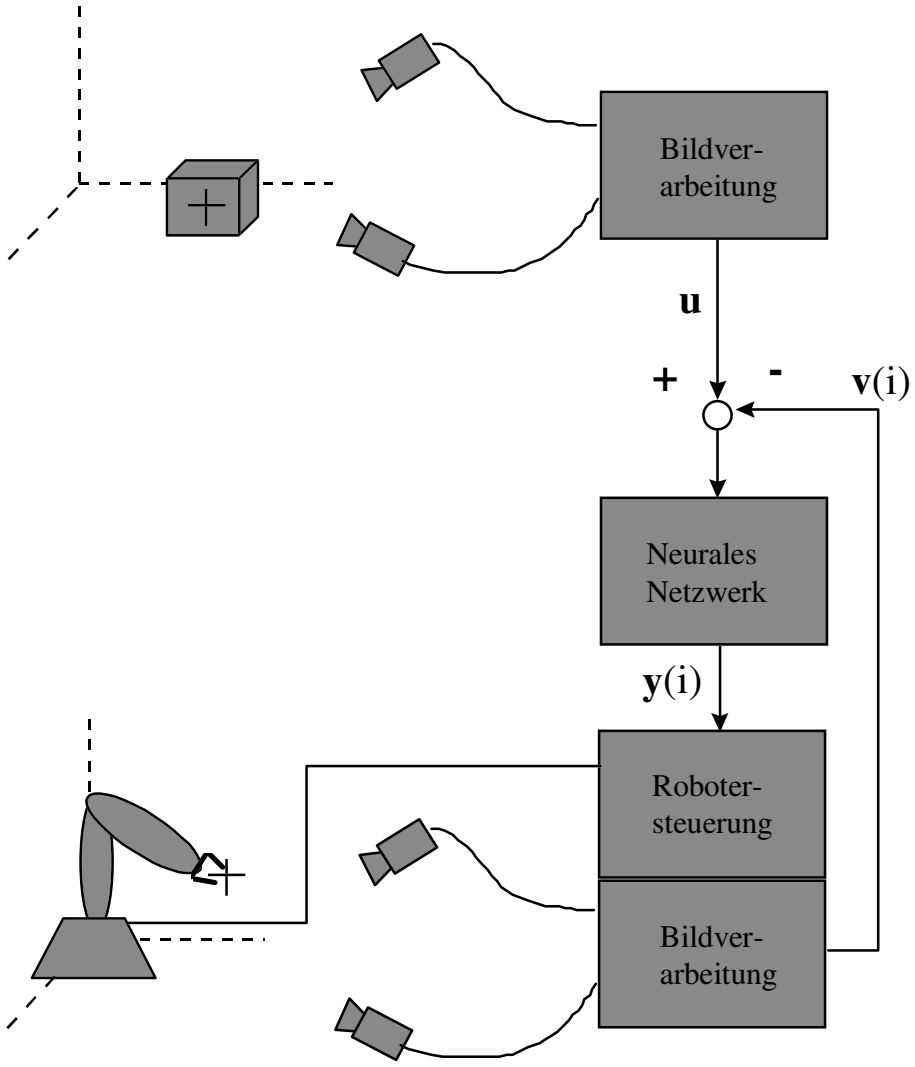

Abb. 3.2: Schematische Darstellung des Positioniervorgangs. Zunächst wird die Zielposition $\mathbf{u} = ((x_{cam1}, y_{cam1}), (x_{cam2}, y_{cam2})) \in \Re^4$ in Stereobildkoordinaten, also als Projektion des anzufahrenden Punktes auf die beiden Kamerabildebenen, ermittelt (im Rahmen der Diplomarbeit wurde dazu die Greiferspitze selbst verwendet). Das neurale Netzwerk generiert einen vorläufigen Gelenkwinkelvektor $\mathbf{y}(0) \in \Re^3$, welcher zu einer ersten **open loop**-Positionierung des Roboterarms führt. Anschließend wird durch die Bildverarbeitungseinheit die aktuelle Position $\mathbf{v}(0) \in \Re^4$ des mit einer LED markierten Effektors ermittelt und der Ausgabefehler $\mathbf{u} - \mathbf{v}(0)$ ins Netzwerk zurückgekoppelt, welcher von diesem zur Berechnung eines verbesserten Ausgabevektors $\mathbf{y}(1)$ verwendet wird. Die Robotersteuerung veranlaßt nun eine erste Korrekturbewegung der Arms, welche unter den in Abschnitt **2.3.2** gegebenen Voraussetzungen zu einer Verringerung des Ausgabefehlers $\mathbf{u} - \mathbf{v}(1)$ führt. Diese Rückkopplungsschleife wird - je nach Abbruchkriterium - mehrere Male durchlaufen.
Die Korrekturbewegungen dienen nicht nur der unmittelbaren Verbesserung der Positioniergenauigkeit, sondern werden vom Netzwerk auch zum Training der Ausgabegewichtsvektoren benötigt.

Ein weiterer positiver Effekt dieser Vorgangsweise ist darin zu sehen, daß zu jedem Trainingsvektor auch dessen kartesische Koordinaten bekannt sind (die aktuelle Effektorpositon kann jederzeit vom Roboterkontroller abgefragt werden) und somit der Positionierfehler nicht nur in Stereobild-, sondern auch in dreidimensionalen kartesischen Koordinaten gemessen werden kann.

Es ist jedoch nicht möglich, das erweiterte *Neural Gas*-Netzwerk ausschließlich aufgrund des oben beschriebenen Systemaufbaus die **inverse Kinematik** des Roboters erlernen zu lassen. Das Netzwerk generiert zu Beginn des Trainings Gelenkwinkelwerte, welche zu sehr großen Positionierfehlern führen, was unter anderem folgende Probleme mit sich bringt:

- Der Effektor verläßt den Sichtbereich einer oder beider Kameras, wodurch die Effektorposition nicht mehr feststellbar ist.
- Der Effektor verläßt den sicheren Arbeitsbereich und kollidiert z.B. mit der Arbeitsplatte.
- Der Roboterarm nimmt häufig Stellungen ein, in welchen sich eine oder mehrere Gelenkachsen *(joints)* in Extrempositionen (maximale Auslenkung der betreffenden Achse) befinden, was zu starker Belastung und unnötiger Abnutzung der Hardware führt.

Unter diesen Voraussetzungen ist es also nicht zielführend, dem Netzwerk sofort die Kontrolle über die Roboterhardware zu übergeben.[22] Die ihm Rahmen der Diplomarbeit implementierte Alternative besteht darin, zunächst eine Softwareemulation des aus Roboter und Kameras bestehenden Systems zu erstellen und das neurale Netz auf dieser zu trainieren, bis der mittlere Ausgabefehler der **open loop**-Positionierung hinreichend gering[23] ist, um sicherzustellen, daß sich der Effektor in der überwiegenden Mehrzahl aller Fälle auch nach Durchführung der **open loop**-Positionierung im gültigen Arbeitsbereich befindet. Sobald dies gewährleistet ist, wird das Netz *online* geschaltet und auf dem „realen“ System weitertrainiert. Es ist natürlich trotzdem notwendig, geeignete Mechanismen vorzusehen, um erkennen zu können, falls der Effektor den Sichtbereich der Kameras oder den sicheren Arbeitsbereich verläßt, und in diesem Fall den Positioniervorgang abzubrechen.

---

[22] Ein Aspekt, auf den Martinetz in seiner Originalarbeit interessanterweise nicht eingeht.

[23] Wir orientieren uns am mittleren Ausgabefehler der **open loop**-Positionierung, da die mittleren Ausgabefehler der Korrekturbewegungen bereits nach einigen hundert Trainingsschritten wesentlich kleiner ausfallen.

3.1. Roboter

3.1.1. Hardware

Im Rahmen der Diplomarbeit wurde ein sechsachsiger Industrieroboter vom Typ A465 des kanadischen Herstellers CRS eingesetzt. Die Tatsache, daß dieses Robotersystem auch in einer *High Level*-Sprache wie z.B. „C“ programmiert werden kann, läßt es für den Einsatz im technisch-wissenschaftlichen Bereich besonders geeignet erscheinen (siehe Abschnitt **3.1.2**). Abbildung 3.3 zeigt den A465 in seinem Weltkoordinatensystem (dem sogenannten *base frame*), in welchem üblicherweise die Position (x, y, z) und Orientierung (*yaw*, *pitch*, *roll*) des Effektors angegeben wird. Es ist immer möglich, die Effektorposition in kartesischen Koordinaten vom Roboterkontroller abzufragen, auch wenn der Arm mit Gelenkwinkelkoordinaten (wie sie das *Neural Gas*-Netzwerk liefert) angesteuert wurde.

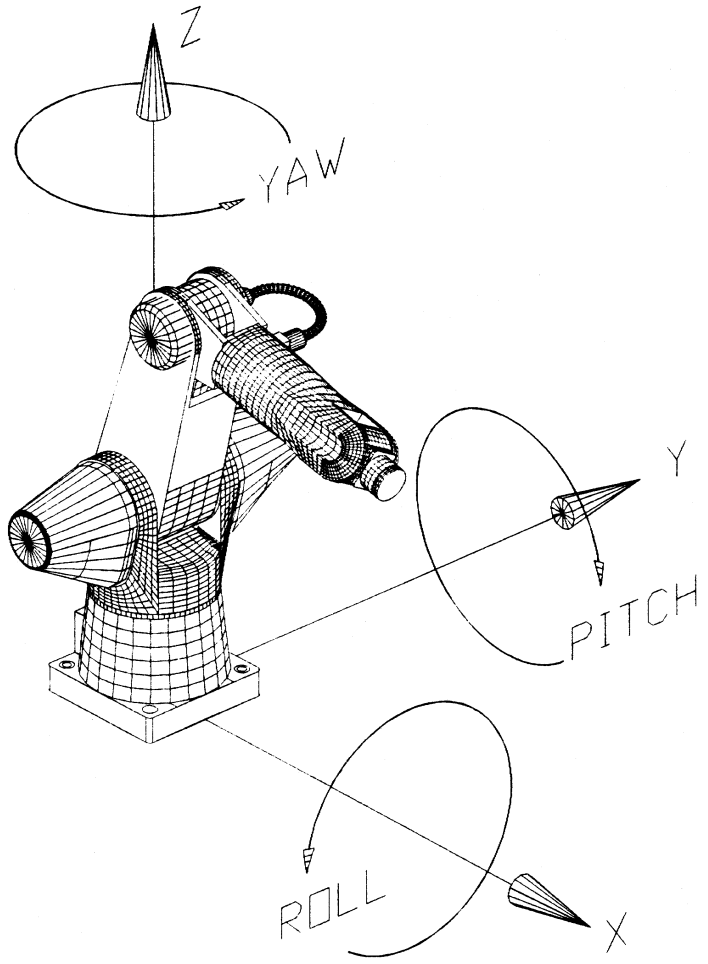

Abb. 3.3: Das Weltkoordinatensystem des A465.

Der A465 verfügt über sechs Achsen (siehe Abbildung 3.4), von denen ihm Rahmen der Diplomarbeit allerdings nur die ersten drei - *base*, *shoulder* und *elbow* - verwendet werden. Dies stellt keine wesentliche Einschränkung dar, da die Achsen 4 (*wrist rotate*) und 6 (*wrist roll*) ausschließlich und die Achse 5 (*wrist bend*) vornehmlich der Orientierung des Effektors dienen, wir aber nur an der Positionierung des Effektors, d.h. am Erreichen eines vorgegebenen Zielpunkts im Arbeitsbereich, interessiert sind. Aus unserer Sicht sind diese 3 Achsen also redundant. Würden wir sie dennoch berücksichtigen, müßten wir das Netzwerk entsprechend größer auslegen (die Topologie der Eingabesignalmenge wäre 6-dimensional) und somit auch längere Rechenzeiten in Kauf nehmen; das größere Problem besteht jedoch darin, daß es mit zunehmender Zahl der Ausgabewerte immer schwieriger wird, ein konvergentes Verhalten des Netzwerks zu gewährleisten.[24]

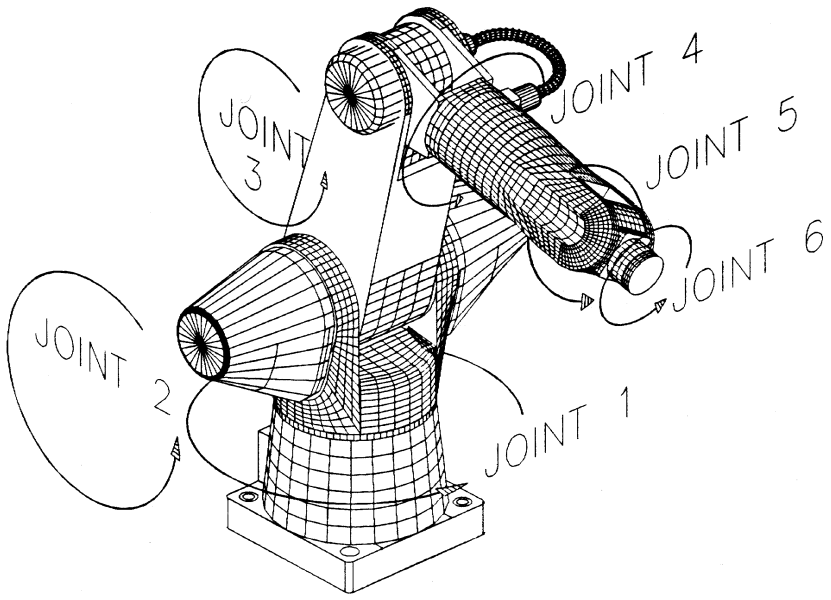

Abb. 3.4: Die Achsen des A465: 1. *Base rotation*, 2. *Shoulder rotation*, 3. *Elbow rotation*, 4. *Euler wrist (wrist rotate*), 5. *Wrist bend*, 6. *Wrist roll*.

[24] Diese Probleme lassen sich z.B. durch den Einsatz hierarchischer Netzwerkmodelle vermeiden. Martinetz, Ritter und Schulter beschreiben in [Martinetz & Ritter & Schulter, 1991] ein hierarchisches *Kohonen*-Netzwerk, welches zusätzlich zur Position des Zielobjekts auch dessen Orientierung im Arbeitsbereich berücksichtigt und sogar in der Lage ist, eine einfache Greifstrategie zu erlernen.

Die Achsen 4, 5 und 6 werden aus diesem Grund also konstant in ihrer Ruhelage gehalten**,** wodurch das System effektiv nur mehr über 3 Freiheitsgrade verfügt. Diese kinematisch eingeschränkte Version des A465 enthält die 3 Armsegmente *base* (Aufsatzpunkt für Achse 1), *upper arm* (zwischen Achse 2 und 3) und *forearm* (zwischen Achse 3 und 5). Da sich Achse 5 stets in der Ruhelage befindet, fungiert das letzte Armsegment *wrist* (ab Achse 5) lediglich als Verlängerung des Segments *forearm*. Nicht eingezeichnet ist der Effektor (Greifer), welcher an der Flansch am Ende des *wrist*-Segments montiert ist; die effektive Länge des letzten Armsegments ergibt sich somit als Summe: Länge(*forearm*) + Länge(*wrist*) + Länge(Effektor).

Die vollständige kinematische Beschreibung eines Roboterarms umfaßt für jedes Armsegment (*link*) die relative Orientierung der durch es verbundenen Achsen (*link twist*), den Abstand der beiden Achsen (*link length*), den Versatz zum nächsten Armsegment entlang der verbindenden Achse (*link offset*) sowie den mit dem nächsten Armsegment eingeschlossen Winkel entlang der verbindenden Achse (*joint angle*). Werden diese vier Parameter für jedes Armsegment angegeben, so erhält man die kinematische Beschreibung des Systems in Form der sogenannten **Denavit-Hartenberg-Notation**.[25] Verfügt der Roboter - wie im vorliegenden Fall - ausschließlich über Drehgelenke, sind die ersten drei Parameter für jedes Armsegment konstant, lediglich der mit dem nächsten Armsegment eingeschlossene Winkel (*joint angle*) kann sich ändern. Eine ausführliche Diskussion der Denavit-Hartenberg-Parameter sowie deren Verwendung zur kinematischen Simulation findet man z.B. in [Craig, 1989].

Die wichtigsten kinematischen Parameter des eingeschränkten Systems sind in den beiden nachfolgenden Tabellen zusammengestellt (der Drehsinn der einzelnen Achsen kann Abbildung 3.4 entnommen werden):

Achse	**Bereich**
1 (*base*)	+/- 175 °
2 (*shoulder*)	+/- 90 °
3 (*elbow*)	+/- 105 °

Tabelle 3.1: Schwenkbereich der ersten 3 Achsen des A465.

[25] Entsprechend werden die Größen *link length*, *link twist*, *link offset* und *joint angle* als **Denavit-Hartenberg-Parameter** bezeichnet.

Segment	Länge
base	330,2 mm
upper arm	304,8 mm
forearm	330,2 mm
wrist	76,2 mm
Effektor	135,0 mm
forearm (effektiv)	541,4 mm

Tabelle 3.2: Länge der Armsegmente des A465.

Anhand dieser Daten wurde von mir eine kinematische Simulation des A465 entwickelt, welche - wie bereits ausgeführt - in der Frühphase des Trainings zur Emulation des Robotersystems benötigt wird; sobald im weiteren Verlauf das neurale Netzwerk die Kontrolle über das Robotersystem erhält, wird die kinematische Simulation außerdem vor jeder tatsächlichen Armpositionierung aufgerufen, um sicherzustellen, daß sich die anzufahrende Position innerhalb des sicheren Arbeitsbereichs befindet, um gegebenenfalls den Positioniervorgang abbrechen zu können. Eine weitere Aufgabe der kinematischen Simulation besteht darin, ungültige - also außerhalb des in Tabelle 3.1 angegebenen Bereichs liegende - Gelenkwinkelwerte zu klippen, d.h. auf den ähnlichsten noch erlaubten Wert zu setzen.

3.1.2. Steuerung und Kommunikation

Die meisten verfügbaren Industrieroboter können ausschließlich in der proprietären Kommandosprache des Herstellers programmiert werden. Dies ist im Falle typischer industrieller *stand alone*-Applikationen i.a. nicht weiter problematisch, konfrontiert den Entwickler aber mit größeren Schwierigkeiten, sobald die Robotersteuerung - wie im vorliegenden Fall - nur eine Komponente eines komplexen Gesamtsystems darstellt. Speziell auf die Programmierung von Robotern zugeschnittene Sprachen verfügen nicht über die Flexibilität und die Vielfalt an Datentypen einer Allzwecksprache wie „C“ oder Pascal, welche z.B. für die Bewältigung von Bildverarbeitungsaufgaben erforderlich ist.

Roboterapplikationen werden tpyischerweise in den Speicher des Roboterkontrollers geladen und dort ausgeführt. Dies bringt - neben den eingangs erwähnten Schwierigkeiten - folgende Probleme mit sich:

- Die Knappheit an verfügbarem Arbeitsspeicher.
- Die geringe Ausführungsgeschwindigkeit der Programme.
- Fehlende Möglichkeiten, vom Kontroller aus Spezialhardware (Netzwerkkarte, *frame grabber* etc.) anzusprechen.

Viele Robotersysteme akzeptieren jedoch zusätzlich über die serielle Schnittstelle empfangene Kommandos (der mit dem Roboter verbundene Rechner fungiert hierbei als Terminal), welche allerdings in Form eines Textstrings übermittelt und im Roboterkontroller erst geparst werden müssen, bevor eine entsprechende Aktion des Roboterarms veranlaßt werden kann. Es wäre aber durchaus möglich, die Ansteuerung des Roboters auf diese Weise z.B. von einem PC aus vorzunehmen, wodurch die oben erwähnten Restriktionen wegfielen. Allerdings wäre die Kommunikation zwischen Roboterkontroller und PC in diesem Falle langsam (für jedes Kommando muß eine Textstring generiert und geparst werden; außerdem bringt die Stringrepräsentation im Vergleich mit einer binären Kodierung erheblich größeren Platzbedarf und somit eine längere Übertragungsdauer mit sich) und - was das größere Problem darstellt - unzuverlässig (falsch übertragene Zeichen werden nicht erkannt).

Das im Rahmen der Diplomarbeit verwendete Industrierobotersystem vom Typ A465 der kanadischen Firma CRS ist hier weitaus flexibler, indem es erlaubt, in 8086-Intel Maschinencode vorliegende Programme - welche z.B. mit einem Assembler oder einem „C"-Compiler erstellt wurden - in den Roboterkontroller zu laden und dort auszuführen; diese in CRS-Terminologie als **PCP**s (***P**rocess **C**ontrol **P**rogram*) bezeichneten Programme können über eine Softwareschnittstelle zur Roboterfirmware im Prinzip jede gewünschte Aktion des Roboters veranlassen. Dies gestattet es, kontrollerseitig einen „Kommunikationsserver" zu installieren, welcher vom PC über die serielle Schnittstelle übermittelte Anfragen dekodiert, in einen Aufruf der Roboterfirmware umsetzt und schließlich - wiederum über die serielle Schnittstelle - eine Erfolgsmeldung zurückgibt. Diesen Ansatz verfolgt auch die am PRIP entwickelte Bibliothek **TROL** *(**T**U Vienna **R**obot **Co**ntrol **L**ibrary*), welche über die serielle Schnittstelle eine zuverlässige Verbindung zwischen PC und Roboterkontroller aufbaut und auf dem PC zusätzlich ein komfortables „C"-**API**[26] zur Verfügung stellt, welches es dem Benutzer ermöglicht, den Roboter aus seiner auf dem PC laufenden Applikation heraus anzusteuern, ohne selbst auf **PCP**-Ebene programmieren zu müssen.

[26] ***A**pplication **P**rogram **I**nterface*: der Begriff stammt ursprünglich aus dem Bereich der Programmierung graphischer Benutzerschnittstellen und wird mittlerweile allgemein zur Bezeichnung von Programmierschnittstellen verwendet.

TROL stellt ausschließlich ein Kommunikationsprotokoll plus Programmierschnittstelle zur Verfügung und kann nicht dazu verwendet werden, die für die Bewegung der Achsen zuständigen Servomotoren direkt anzusprechen. Dies ist auch nicht notwendig, da die Steuerung durch das erweiterte *Neural Gas*-Netzwerk rein kinematischer Natur ist und somit lediglich die Geometrie des Arms, nicht aber seine Masse berücksichtigt. Die Motoren selbst werden jedoch nicht direkt über Gelenkwinkel, sondern über Drehmomente angesteuert, was die Kenntnis der dynamischen Parameter des Arms (z.B. seiner Masseverteilung) voraussetzt; zusätzlich müssen - vor allem bei höheren Geschwindigkeiten auftretende - Trägheitseffekte und die wechselseitige Beeinflussung der einzelnen Armsegmente berücksichtigt werden, was die wiederholte Lösung eines nichtlinearen Systems von Bewegungsgleichungen erforderlich macht. Die Aufgabe, anhand der vom *Neural Gas*-Netzwerk vorgegebenen Gelenkwinkel diese Berechnungen in Echtzeit vorzunehmen und die einzelnen Motoren anzusteuern, übernimmt daher weiterhin die Roboterfirmware.[27]

[27] Ein um Ausgabewerte erweitertes *Kohonen*-Netzwerk, welches in der Lage ist, die dynamischen Parameter einer Roboterarms zu erlernen, ist in [Martinetz & Ritter & Schulten, 1991], Kapitel 13, beschrieben.

3.2. Bildverarbeitung

Der Bildverarbeitungseinheit kommt die Aufgabe zu, die aktuelle Effektorposition in beiden Kamerabildern zu ermitteln. Diese Information wird sowohl für das *online*-Training des Netzwerks als auch zur Kalibrierung der Kameras benötigt. Obwohl der Einsatz des erweiterten *Neural Gas*-Netzwerks die Kamerakalibrierung theoretisch überflüssig macht, muß dennoch vor Beginn des Trainings zumindest die externe Orientierung der beiden Kameras ermittelt werden, da diese ja in der ersten Phase des Trainings durch Software emuliert werden sollen.

3.2.1. Hardware

Bei der verwendeten Bildverarbeitungshardware handelt es sich um eine ISA[28]-Steckkarte der Firma Matrox, welche über einen programmierbaren Digitizer mit drei Eingängen verfügt und imstande ist, einige grundlegende Bildverarbeitungsoperationen in Echtzeit auszuführen. Die Programmierung der Karte im *native*-Modus, in welchem die volle Leistungsfähigkeit der Hardware ausgeschöpft werden kann, ist äußerst komplex und zeitaufwendig, weshalb die Programmbibliothek **MIL** (**M**atrox **I**mage **L**ibrary) als Softwareschnittstelle verwendet wurde. **MIL** stellt unabhängig von der tatsächlich vorhandenen Bildverarbeitungshardware eine breite Palette an Bildverarbeitungsoperationen zur Verfügung, ist jedoch in der Lage, gewisse Steckkarten (unter anderem auch die im Rahmen der Diplomarbeit verwendete Matrox-Karte) zu erkennen und für eine beschleunigte Durchführung der von der Hardware unterstützen Operationen zu nutzen; dies geschieht in für den Benutzer transparenter Weise.

Die Bilder selbst wurden mit zwei in einem Winkel von ca. 90 Grad montierten CCD-Kameras mit einer Chipgröße von 8.8×6.6 mm respektive $7{,}95 \times 6{,}45$ mm gewonnen und mit einer Auflösung von 764 × 574 Pixeln in 256 Graustufen gesampelt; beide Kameras waren mit einem 16mm-Objektiv ausgestattet. Da die Bildverarbeitungsaufgabe im Auffinden einer aktiven Lichtquelle besteht (siehe unten), wurde, um störende Licht- und Reflexionseffekte soweit wie möglich auszuschließen, jeweils die maximale Blendeneinstellung von 16 gewählt.

[28] ***I**ndustrie **S**tandard **A**rchitecture*: veraltetes (aber immer noch weit verbreitetes) 16 bit-PC-Bussystem, das aufgrund seines geringen Durchsatzes einen Flaschenhals für jede datenintensive Anwendung darstellt.

3.2.2. Ermittlung der Effektorposition

Die Effektorspitze wurde zwecks leichterer Erkennbarkeit mit einer LED markiert, sodaß sich das eigentliche Bildverarbeitungsproblem auf die Ermittlung der LED-Position reduziert (siehe Abbildung 3.5).

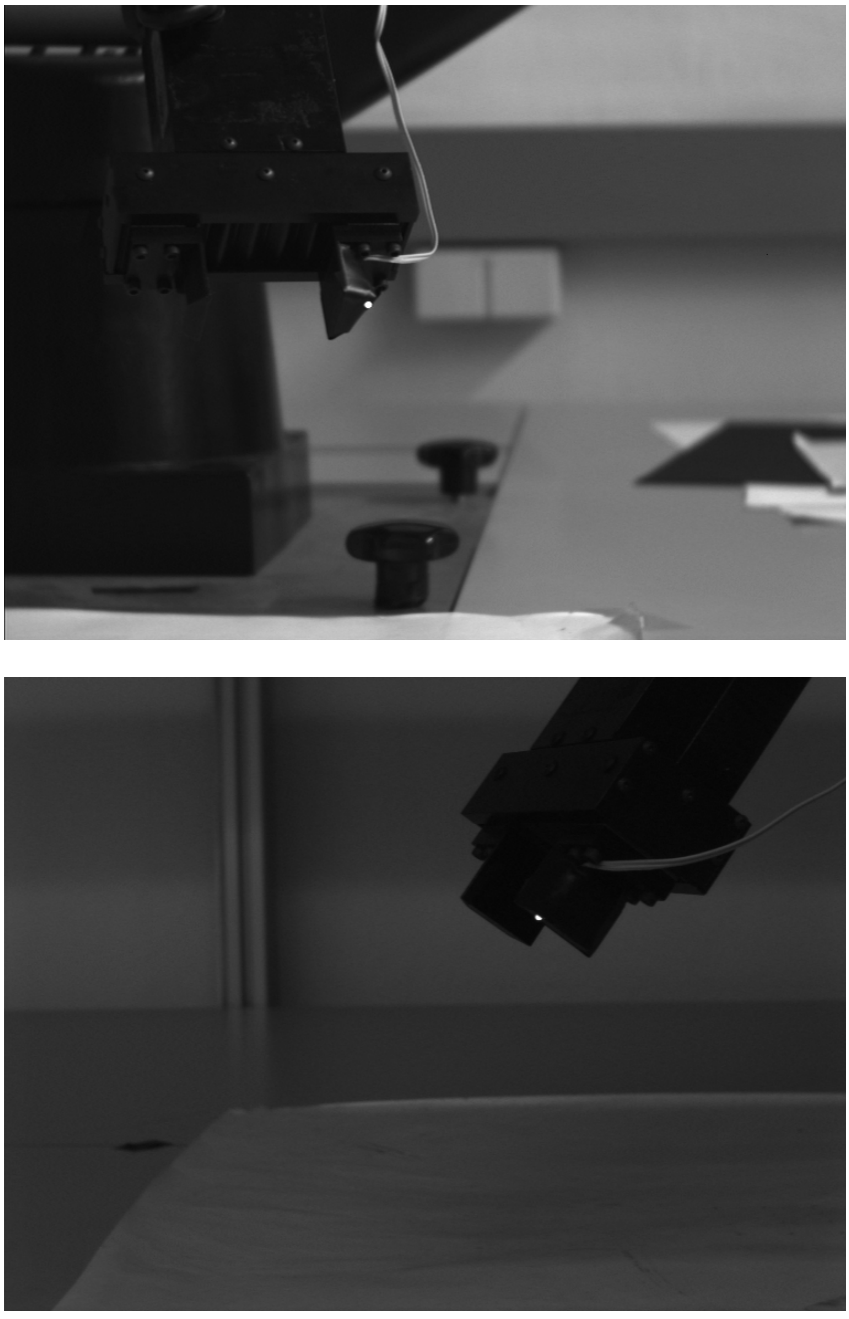

Abb 3.5: Blick durch die beiden Kameras auf die mit einer LED markierte Effektorspitze.

Dies wird mittels eines einfachen Schwellwertverfahrens bewerkstelligt: der Schwellwert T_{LED} wird dabei anhand des Grauwert-Histogramms als höchster auftretender Grauwert ermittelt. Enthält das Bild nach der Segmentierung genau einen Blob[29], so wird angenommen, daß dieser mit der LED korrespondiert.

Um sicherzustellen, daß, falls sich der Greifer nicht im Sichtbereich der Kamera befindet, nicht einfach der hellste Punkt im Bild fälschlicherweise als LED interpretiert wird, wurde zunächst empirisch die minimale Helligkeit H_{min} der LED im Kamerabild ermittelt (diese hängt außer von der Umgebungshelligkeit und der Blendeneinstellung auch von der Position und Orientierung der LED relativ zur Kamera ab). Sind nun die beiden Bedingungen:

a) der gefundene Schwellwert ist nicht kleiner als der geforderte Minimalwert: $T_{LED} \geq H_{min}$ und
b) das segmentierte Bild enthält genau einen Blob[30]

erfüllt, so berechnet sich die Position der LED im Bild als Schwerpunkt bzw. erster Moment des gefundenen Blobs (in der Praxis betrug die Größe des gefundenen Blobs jedoch selten mehr als 1 Pixel). Andernfalls wird dem Benutzer die Auswahl überlassen, ob er den Vorgang abbrechen oder mit einem neu aufgenommenen Bild wiederholen möchte. Eine Wiederholung führt erfahrungsgemäß in Fällen, in denen ursprünglich Bedingung b) verletzt war, meist zum Erfolg.

Die so ermittelte LED-Position muß anschließend noch von Pixel- bzw. *frame buffer*- in Bildkoordinaten umgerechnet werden. Der *frame buffer* (b_{ij}) ist jener Speicherbereich, in welchem der Digitizer das digitalisierte Bild ablegt, und entspricht in unserem Fall einer 764×574 Matrix, wobei der Zeilenindex i und der Spaltenindex j im Bereich 0 .. 763 respektive 0 .. 573 liegen. Hat der verwendete CCD-Chip eine Größe von 8.8×6.6 mm, so lassen sich die dem Pixel b_{ij} entsprechenden Bildkoordinaten x', y' in mm mittels

---

[29] Blob: zusammenhängende Menge von Pixeln p_i, die ein vorgegebenes Prädikat erfüllen (im konkreten Fall: Helligkeit(p_i) = T_{LED}).

[30] Als Alternative zu diesen beiden Forderungen könnte man zunächst die „erwartete" LED-Position berechnen und mit den tatsächlich gefundenen Positionen vergleichen; überschreitet die Distanz zwischen erwarteter Position und der ähnlichsten gefundenen Position einen vorgegebenen Maximalwert, wird dies als Fehler betrachtet, ansonsten wird der ähnlichste Wert ausgewählt. Diese Vorgangsweise setzt allerdings voraus, daß die Kamera bereits kalibriert wurde und die Weltkoordinaten der LED bekannt sind.

$$x' = (j - 763 / 2) * 8.8 / 764 \tag{3.1}$$

$$y' = -(i - 573 / 2) * 6.6 / 574. \tag{3.2}$$

schätzen. Die gemäß obiger Gleichungen errechneten Werte entsprechen aufgrund kleiner Fehler in der Kamera- und Digitizerhardware allerdings nicht den wirklichen Bildkoordinaten; bei Bedarf können diese Ungenauigkeiten jedoch durch geeignete Kalibrierungstechniken ausgeglichen werden.

3.2.3. Kamerakalibrierung

Die Kalibrierung der beiden Kameras ist, wie bereits weiter oben ausgeführt, erforderlich, um diese in der Frühphase des Trainings durch Software emulieren zu können. Die hinsichtlich der erzielten Genauigkeit an das Kalibrierungsverfahren gestellten Anforderungen sind eher gering, da die Emulation nur benötigt wird, um das neurale Netzwerk *offline* vorzutrainieren. Sobald das *Neural Gas*-Netzwerk die tatsächliche Kontrolle über das System erhält, ist es aufgrund seiner adaptiven Natur in der Lage, die Unterschiede zwischen den emulierten und realen Kameras relativ schnell auszugleichen. Weiters ist keine Stereokalibrierung erforderlich, da lediglich Welt- in Bildkoordinaten (und nicht umgekehrt) umgerechnet werden müssen, sodaß die beiden Kameras unabhängig voneinander kalibriert werden können. Die Kalibrierungspunkte selbst wurden in naheliegender Weise mit Hilfe des Roboters gewonnen, indem zunächst mittels konventioneller Ansteuerung ein Punkt im Arbeitsbereich angefahren und anschließend die Position der LED bestimmt wurde.

Bei den 12 zu kalibrierenden Parametern handelt es sich um die Koeffizienten der 3×3 Rotationsmatrix $\mathbf{R}$ und des dreidimensionalen Vektors $\mathbf{T} = (t_1, t_2, t_3)^T$, welche Position und Orientierung des Weltkoordinatensystems relativ zum Kamerakoordinatensystem festlegen (externe Orientierung):

$$\mathbf{p}_{cam} = \mathbf{R} * \mathbf{p}_{world} + \mathbf{T}. \tag{3.3}$$

Der mit $\mathbf{p}_{cam} = (x, y, z)^T$ korrespondierende Bildpunkt $\mathbf{p}_{img} = (x', y')^T$ errechnet sich dann mittels perspektivischer Projektion zu:

$$x' = x * (f/z), \quad y' = y * (f/z), \tag{3.4}$$

wobei wir mit f die fokale Länge der Kamera bezeichnen.

Die Beziehungen (3.3) und (3.4) liefern uns für jeden Kalibrierungspunkt **p**, dessen Welt- und Bildkoordinaten bereits bekannt sind, zwei homogene lineare Gleichungen mit den 12 zu bestimmenden Parametern als Unbekannten (die Werte $\mathbf{r}_i$ bezeichnen die Zeilenvektoren von **R**):

$$x' * (\mathbf{r}_3 * \mathbf{p}_{world} + t_3) - f * (\mathbf{r}_1 * \mathbf{p}_{world} + t_1) = 0, \tag{3.5}$$

$$y' * (\mathbf{r}_3 * \mathbf{p}_{world} + t_3) - f * (\mathbf{r}_2 * \mathbf{p}_{world} + t_2) = 0. \tag{3.6}$$

Im vorliegenden Fall wurden 27 auf 3 Ebenen verteilte Kalibrierungspunkte ermittelt, wodurch sich insgesamt 54 Gleichungen ergeben. Um den Nullvektor als triviale Lösung des Systems zu vermeiden, wird ein Parameter (im konreten Fall t_1) festgehalten und auf die rechte Seite gebracht. Das so erhaltene überbestimmte lineare Gleichungssystem kann nun mit Hilfe eines Standardverfahrens wie z.B. **SVD**[31] gelöst werden. Diese Vorgangsweise bezeichnet man als **affine Kalibrierung** oder **DLT** (***D**irect **L**inear **T**ransform*).[32] Die in den Gleichungen (3.5) und (3.6) vorkommende fokale Länge f muß lediglich als Schätzwert vorliegen und wird - ebenso wie die im Zusammenhang mit Gleichung (3.1) und (3.2) erwähnten Hardwarefehler - durch das Verfahren implizit mitkalibriert.

Zu beachten ist, daß die mittels affiner Kalibrierung gefundene Rotationsmatrix **R** i.a. nicht orthonormal ist. Bei meinen ersten Kalibrierungsversuchen mußte ich feststellen, daß die Qualität der gefundenen Lösung sehr stark von der Wahl des festzulegenden Parameters t_1 beeinflußt wird; wird t_1 insbesondere zu klein angesetzt, so werden einige

---

[31] ***S**ingular **V**alue **D**ecomposition*: numerisch äußerst stabiles Verfahren zur Lösung linearer Gleichungssysteme bzw. Optimierungsprobleme (siehe z.B. [Press & Teukolsky & Vetterling & Flannery, 1992], Kapitel 2.6). Dieses Verfahren konstruiert explizit die Basen für Bild und Kern der Koeffizientenmatrix und liefert im Falle überbestimmter Gleichungssysteme den im *least square* Sinne besten Lösungsvektor.

[32] Eine genaue Beschreibung des Verfahrens findet man z.B. in [Jain & Kasturi & Schunck, 1995], S 352 ff.

Koeffizienten von **R** mitunter sehr groß, was zu numerischen Problemen führen kann. In einem zweiten Anlauf verwendete ich daher das von **Tsai** [Tsai, 1987] vorgeschlagene zweistufige Kalibrierungsverfahren. Dieses liefert im Unterschied zur affinen Kalibrierung eine orthonomale Rotationsmatrix **R** und ermittelt zusätzlich die fokale Länge f und den Koeffizienten κ, welcher die radiale Linsenverzerrung[33] beschreibt. Im ersten Schritt werden sowohl die Matrix **R** als auch die beiden Werte t_1, t_2 bestimmt, was die Lösung eines aufgrund des *radial alignment constraint*[34] gewonnenen überbestimmten linearen Systems erforderlich macht. Im zweiten Schritt werden die Parameter f, t_3 und κ durch nichtlineare Suche ermittelt.

Ihm Rahmen der Diplomarbeit wurde eine Kombination aus beiden Verfahren verwendet, welche für den gegebenen Systemaufbau zumeist bessere Ergebnisse erzielte als eines der Verfahren alleine: zunächst werden mittels des *Tsai*-Algorithmus die Parameter κ, f, und t_1 ermittelt, anschließend die Bildkoordinaten der Kalibrierungspunkte radial entzerrt und die affine Kalibrierung mit den vorher ermittelten Werten für f und t_1 durchgeführt.

[33]Die radiale Linsenverzerrung wird mathematisch durch $x' = x'_d (1 + \kappa_1 * r^2 + \kappa_2 * r^4 ...)$, $y' = y'_d (1 + \kappa_1 * r^2 + \kappa_2 * r^4 ...)$ modelliert, wobei x', y' die ungestörten Bildkoordinaten, x'_d, y'_d die gestörten Bildkoordinaten und r den Abstand des Vektors (x'_d, y'_d) vom Ursprung des Bildkoordinatensystems bezeichnen. In der Praxis werden die Terme höherer Ordnung als 2 jedoch meist nicht berücksichtigt. Der vom *Tsai*-Algorithmus gefundene Parameter κ entspricht somit κ_1.

[34] Der *radial alignment constraint* (RAC) postuliert, daß die Verbindungsstrecke zwischen $\mathbf{p}_{cam}$ und der optischen Achse $\overline{(x,y,z),(0,0,z)}$ sowie die Verbindungsstrecke zwischen dem korrespondierenden Bildpunkt $\mathbf{p}_{img}$ und dem Ursprung des Bildkoordinatensystems $\overline{(x',y',f),(0,0,f)}$ auch im Falle radialer Linsenverzerrung immer parallel sind.

3.3. Neurales Netzwerk

Das erweiterte *Neural Gas*-Netzwerk wurde mit Hilfe des unter X11 verfügbaren Simulationstools XERION auf einer DEC Workstation implementiert. Die Simulation kann entweder als eigenständige Applikation im *stand alone*-Modus betrieben oder mittels eines einfachen Protokolls über eine *socket*-basierte[35] Verbindung vom PC aus „ferngesteuert“ werden. In der zweiten Betriebsart wartet das neurale Netzwerk zunächst, bis die Stereokoordinaten des anzufahrenden Zielobjekts übermittelt werden und übernimmt anschließend für die Dauer des Positioniervorgangs die Ablaufkontrolle, wobei es für jede Korrekturbewegung (inklusive der **open-loop** Positionierung) i, $0 \leq i \leq n$, die folgenden Schritte ausführt:

1. Generierung der Gelenkwinkel **y**(i).
2. Übertragung der Gelenkwinkel zum PC.
3. Warten auf das Quittungspaket, welches die Stereobildkoordinaten der korrespondierenden Effektorposition **v**(i) sowie den evt. modifizierten Wert von **y**(i) (siehe Abschnitt **3.3.2**) enthält.

Nach Abschluß des Positioniervorgangs oder bei Auftreten eines Fehlers geht die Ablaufkontrolle wieder an das auf dem PC laufende Hauptprogramm zurück.

Der „Fernsteuerungs“-Modus war ursprünglich ausschließlich für die *online*-Kontrolle des realen Robotersystems durch das neurale Netzwerk gedacht, kann aber durchaus auch zum Training des Netzwerks in der simulierten Umgebung eingesetzt werden. Dies ist (trotz der geringeren Ausführungsgeschwindigkeit im Vergleich zum *stand alone*-Modus) vorteilhaft, da die Kamerakalibrierung und die Erzeugung der Trainingsdaten auf dem PC vorgenommen werden und somit die Notwendigkeit, diese Daten mittels ftp auf die DEC-Workstation zu übertragen und ins XERION-Format zu konvertieren (ein etwas umständliches und fehleranfälliges Unterfangen), entfällt.

[35] *Sockets* bieten eine komfortable Möglichkeit zur Kommunikation zwischen zwei verteilten - d.h. auf verschiedenen Knoten eines Rechnernetzwerks laufenden - Prozessen; aus der Sicht des Benutzers wird ein *socket* wie eine Datei oder eine *pipe* behandelt. Der Vorteil gegenüber der Verwendung eines verteilten Dateisystems (z.B. NFS) liegt darin, daß *sockets* auch zur Synchronisation der beiden verbundenen Prozesse verwendet werden können.

3.3.1. Modifikationen gegenüber dem Originalverfahren

Aufbau und Funktionsweise des um Ausgabewerte erweiterten *Neural Gas*-Netzwerks wurden bereits in Abschnitt **2.3** beschrieben. Allerdings mußten gegenüber dem Originalverfahren noch einige Ergänzungen und Veränderungen vorgenommen werden, um die Konvergenz des Verfahrens speziell für die vorliegende Aufgabenstellung zu gewährleisten; diese beinhalten außer dem im nächsten Abschnitt **3.3.2** behandelten *clipping* für zu große Gelenkwinkelwerte die Skalierung der Ausgabewerte und Gewichtsdeltas. Wir präsentieren zunächst den modifizierten Algorithmus für ein Netzwerk mit n *competing units*:

Algorithmus 3.1: Erweitertes Neural Gas Verfahren zur Robotersteuerung mittels visueller Rückkopplung

1. Präsentiere dem Netzwerk den Eingabevektor $\mathbf{u} \in \Re^4$
2. Führe den Lernschritt für den *competing layer* wie in Abschnitt **2.1.3** beschrieben durch.
3. Setze für $1 \leq i \leq n$
$$z_i^{neu} = z_i^{alt} \; / \; \sum_{i=1}^{n} \; z_i^{alt} \tag{3.7}$$
4. Berechne den Ausgabevektor $\mathbf{y}(0) \in \Re^3$ zu
$$y_j \; = \sum_{i=1}^{n} \; y_{ji} = \; z_i \; * \; (\mathbf{a}_{ji} \; . \; \mathbf{u} + \; \alpha_{ji} \;), \; 1 \leq j \leq 3. \tag{3.8}$$
5. Steuere den Roboter mit den Gelenkwinkeln $\mathbf{y}(0)$ an und ermittle die korrespondierenden Stereobildkoordinaten des Effektors $\mathbf{v}(0) \in \Re^4$.
6. Setze Zähler k = 1.
7. Präsentiere dem Netzwerk den Differenzvektor $\mathbf{u}$ - $\mathbf{v}$(k-1).
8. Berechne einen neuen Ausgabevektor $\mathbf{y}(k) = (y_1(k), y_2(k), y_3(k))^T$ zu:
$$y_j(k) = y_j(k\text{-}1) \; + \; \sum_{i=1}^{n} \; z_i \; * \; (\mathbf{a}_{ij} \; . \; (\mathbf{u} \; \text{-} \; \mathbf{v}(k\text{-}1) \;), \; 1 \leq j \leq 3 \tag{3.9}$$
9. Führe die durch $\mathbf{y}(k)$ festgelegte Korrekturbewegung durch und ermittle die resultierende Effektorposition $\mathbf{v}(k) \in \Re^4$.

10. Verwende die beiden Tupel (**y**(k), **v**(k)) und (**y**(k-1), **v**(k-1)), um die Inkremente für die zu den *modulation units* hinführenden Gewichte zu ermitteln.

$$\Delta \mathbf{a}_{ji} = z_i * [(\mathbf{y}_j(k) - \mathbf{y}_j(k-1) - \sum_{i=1}^{n} z_i * (\mathbf{a}_{ji} \,.\, \mathbf{u})] \,.\, (\mathbf{v}(k) - \mathbf{v}(k-1)), \qquad (3.10)$$

$$\Delta \alpha_{ji} = z_i * [(\mathbf{y}_j(k) - \mathbf{y}_j(k-1)] \qquad (3.11)$$

11. Adaptiere die Gewichte:

$$\mathbf{a}_{ji}^{neu} = \mathbf{a}_{ji}^{alt} + \varepsilon'(t) * (1 / \|\mathbf{v}(k) - \mathbf{v}(k-1)\|^2) * \Delta \mathbf{a}_{ji}\,, \qquad (3.12)$$

$$\alpha_{ji}^{neu} = \alpha_{ji}^{alt} + \varepsilon'(t) * \Delta \alpha_{ji} \qquad (3.13)$$

12. Wenn die maximale Anzahl von Iterationen erreicht ist: breche ab; sonst erhöhe k um 1 und gehe zurück zu 7.

Die Skalierung der Gewichtsdeltas $\Delta \mathbf{a}_{ji}$ mit $1 / \|\mathbf{v}(k) - \mathbf{v}(k-1)\|^2$ in Gleichung (3.12) erweist sich vor allem in der Frühphase der Trainings als wichtig, um ein unbeschränktes Anwachsen der $\mathbf{a}_{ji}$ zu verhindern. Eine ähnliche Skalierung wurde von Martinetz für ein um Ausgabewerte erweiteres *Kohonen*-Netzwerk, welches ebenfalls imstande ist, die **inverse Kinematik** eines Robotersystems zu erlernen, verwendet ([Martinetz & Ritter & Schulten, 1991], S184), in seiner Beschreibung der erweiterten *Neural Gas*-Verfahrens wird jedoch keine entsprechende Skalierung erwähnt.

Da der Wert $\sum_{i=1}^{n} z_i^{alt}$ gemäß Gleichung (2.8) von der Nachbarschaftsreichweite $\lambda(t)$ abhängig ist, benötigen wir zusätzlich Gleichung (3.7), um sicherzustellen, daß die z_i nur den relativen Beitrag der einzelnen *modulation units* zur Berechnung der Ausgabewerte **y** und der Gewichtsdeltas $\Delta \mathbf{a}_{ji}$ und $\Delta \alpha_{ji}$ festlegen und diese Werte nicht zusätzlich nach oben skalieren ($\sum_{i=1}^{n} z_i^{neu} \equiv 1$). Wird diese Normierung der z_i nicht vorgenommen, so kann dies zu einer Verschlechterung der Positioniergenauigkeit nach anfänglicher Konvergenz des Netzwerks führen bzw. diesen „Inversionseffekt“ verstärken. (siehe Abschnitt **4.3.4**). In früheren Experimenten wurden die z_i sofort (d.h. noch in den *competing units*) normiert; es zeigte sich jedoch, daß dies zu einer dramatischen Verschlechterung des Diskretisierungsfehlers führt, sodaß die Normierung nun erst nach dem Training des *competing layer* vorgenommen wird.

3.3.2. Fehlerbehandlung

Während des Positioniervorgangs können verschiedene Fehler auftreten, welche jedoch - mit einer Ausnahme - nur bei der Ansteuerung des realen Systems berücksichtigt werden müssen:

a) Die vom Netzwerk generierten Gelenkwinkel liegen außerhalb des von der Architektur des Roboterarms unterstützen Bereichs.
b) Die Position des Effektors kann von der Bildverarbeitungseinheit nicht festgestellt werden.
c) Die mit den vom Netzwerk generierten Gelenkwinkeln korrespondierende Effektorposition liegt außerhalb des sicheren Arbeitsbereichs (z.B. unterhalb der Arbeitsplatte).
d) Von der Bildverarbeitungseinheit wird eine falsche Effektorposition ermittelt.

Fall a) muß sowohl beim Training des Netzwerks auf dem simulierten als auch auf dem realen System behandelt werden. Die Fehlerbehandlung besteht einfach darin, ungültige Werte zu *clippen*, d.h. auf den ähnlichsten noch erlaubten Wert zu setzen; diese Aufgabe wird von der kinematischen Simulation (siehe Abschnitt **3.1.1**) übernommen. Dem neuralen Netzwerk müssen in diesem Fall außer der Effektorposition auch die geänderten Gelenkwinkel mitgeteilt werden, da sonst die Berechnung des nächsten Gelenkwinkelvektors (Gleichung (3.9)) sowie der Gewichtsdeltas (3.10) und (3.11) nicht korrekt durchgeführt wird. Diese Vorgangsweise hat sich in der Praxis gut bewährt und ist insbesondere geeignet, bereits während der Simulationsphase die vom *Neural Gas*-Netzwerk generierten Ausgabewerte zu beschränken und in den kinematisch gültigen Bereich zu bringen.

Tritt während des Trainings des Netzwerks auf der realen Umgebung Fall (b) oder (c) ein, so wird der aktuelle Positioniervorgang abgebrochen und mit der Präsentation des nächsten Trainingsvektors fortgefahren. Fall (b) wird von der Bildverarbeitungseinheit, Fall (c) von der kinematischen Simulation (und zwar vor Durchführung der Armbewegung!) erkannt. Wird das neurale Netzwerk hingegen auf der simulierten Umgebung trainiert, kann Fall (b) gar nicht erst auftreten und Fall (c) ignoriert werden.

Die Detektion einer falschen Effektorposition durch die Bildverarbeitungseinheit (Fall (d)) kann, obwohl entsprechende Sicherheitsmaßnahmen vorgesehen wurden (siehe Abschnitt **3.2.2**) nicht ausgeschlossen werden. Sollte dieser Fall tatsächlich eintreten

(was bisher allerdings noch nicht vorgekommen ist), so kann dies schlimmstenfalls darin resultieren, daß das neurale Netz einen Lernschritt mit falschen Werten durchführt; Kollisionen oder Beschädigungen sind jedoch ausgeschlossen, da dies rechtzeitig von der kinematischen Simulation erkannt und der Positioniervorgang abgebrochen werden würde. Fall (d) kann klarerweise nur während des *online*-Trainings auftreten.

Weitere Fehlerursachen sind im Auftreten von Kommunikationsfehlern (PC - Roboterkontroller, PC - Workstation) oder im versehentlichen Betätigen der Not-Stop-Taste des Roboters (geschieht gar nicht so selten) zu sehen. Diese Fehler führen zwar im allgemeinen zu Datenverlust, können aber keine unvorhergesehenen oder ungewollten Bewegungen des Roboterarms verursachen; das System verhält sich also *fail-safe*.

4. Beschreibung der Experimente

In diesem Kapitel werden die Ergebnisse von insgesamt fünf verschiedenen Experimenten präsentiert, welche mit dem erweiterten *Neural Gas*-Verfahren durchgeführt wurden.

4.1. Parameterauswahl

Bei den für das erweiterte *Neural Gas*-Modell festzulegenden Parametern handelt es sich um die Größe des *competing layer*, die Anzahl der durchzuführenden Korrekturbewegungen, die Anzahl der Trainingsschritte sowie jene Parameter, welche den zeitlichen Verlauf der Lernrate für den *competing layer* $\varepsilon(t)$, der Lernrate für den *modulation layer* $\varepsilon'(t)$ und der Nachbarschaftsreichweite $\lambda(t)$, welche für die Berechnung der Ausgabewerte der *competing units* benötigt wird, bestimmen. Da für alle weiter unten präsentierten Versuchsanordnungen mit Ausnahme von Softwareexperiment 4 die gleichen Parameter verwendet wurden, sollen diese hier besprochen werden, um sie nicht für jedes Experiment gesondert anführen zu müssen.

- Der *competing layer* umfaßte 40 Neuronen.
- Die Anzahl der durchzuführenden Korrekturbewegungen wurde mit 4 festgesetzt.
- Es wurden jeweils 4000 Trainingsschritte durchgeführt.

Der zeitliche Verlauf von $\varepsilon(t)$, $\varepsilon'(t)$ und $\lambda(t)$ war von folgender Gestalt[36]

$$v_{start} * (v_{end} / v_{start})^{t / t_{max}} \text{ für } t \leq t_{max} \qquad (4.1)$$
$$v_{end} \text{ für } t > tmax,$$

wobei folgende Werte gewählt wurden:

- $\varepsilon(t)$: $\varepsilon_{start} = 0.3$, $\varepsilon_{end} = 0.02$, $\varepsilon_{max} = 5000$
- $\varepsilon'(t)$: $\varepsilon'_{start} = 0.9$, $\varepsilon'_{end} = 0.1$, $\varepsilon'_{max} = 5000$
- $\lambda(t)$: $\lambda_{start} = 10$, $\lambda_{end} = 1$, $\lambda_{max} = 1000$

[36] Dieser Funktionstyp wurde auch von Martinetz in seiner Originalarbeit verwendet.

Aufgrund des mit jeder Armpositionierung verbundenen Aufwands ist eine der Anforderungen an das erweiterte *Neural Gas*-Netzwerk schnelle Konvergenz, d.h. eine zufriedenstellende Positioniergenauigkeit muß bereits nach einigen Tausend Musterpräsentationen erreicht werden; die Anzahl der Musterpräsentationen wurde daher mit 4000 festgesetzt. Es sei jedoch darauf hingewiesen, daß die Anzahl der Lernschritte, die der *competing layer* für die Adaption der Topologie des Eingabesignalraums benötigt, typischerweise um einen Faktor 10 größer ist.

Während die Anzahl der Korrekturbewegungen willkürlich mit 4 festgelegt wurde, wurden die restlichen Parameter empirisch ermittelt. Wie ich auch in früheren Versuchsreihen [Melzer, 1995] bereits beobachtet habe, wird das Konvergenzverhalten des Netzwerks besonders stark durch die Wahl der Nachbarschaftsreichweite $\lambda(t)$ beeinflußt. Man bemerkt, daß das durch die obigen Parameter gegebene $\lambda(t)$ - im Unterschied zu $\varepsilon(t)$ und $\varepsilon'(t)$ - bereits nach 1000 Lernschritten auf seinen Minimalwert abgefallen ist und anschließend konstant bleibt; dies ist - wie Softwareexperiment 4 zeigen wird - notwendig, um eine nach anfänglicher Konvergenz einsetzende Verschlechterung der Positioniergenauigkeit mit jeder weiteren Musterpräsentation zu vermeiden.

4.2. Berechnung der Ausgabefehler

Wir unterscheiden 3 Arten von Fehlern:

- Den Diskretisierungsfehler err_{comp} des *competing layer.*
- Die i + 1 Positionierfehler des *output layer* (open loop + i Iterationen), $0 \le i \le 4$,
  - in Stereo-Bildkoordinaten: $err_{cam}(i)$
  - in Weltkoordinaten: $err_{world}(i)$.

Der Diskretisierungsfehler wird vom Verfahren nicht explizit benötigt und wurde nur zu Referenzzwecken ermittelt; er berechnet sich als gewichtetes Mittel der euklidischen Distanzen der Referenzvektoren zum Eingabevektor **u**:

$$err_{comp} = \sum_{i=1}^{n} \|\mathbf{w}_i - \mathbf{u}\| * z_i \; / \; \sum_{i=1}^{n} z_i, \qquad (4.2)$$

wobei sich die z_i gemäß Gleichung (2.8) berechnen.

Der i-te Positionierfehler des *output layer* in Stereo-Bildkoordinaten entspricht der euklidischen Norm des nach der i-ten Iteration ins Netzwerk zurückgekoppelten Differenzvektors **u** - **v**(i) (Algorithmus 3.1, Schritt 7); $err_{cam}(0)$ gibt dabei den Fehler der open loop-Positionierung an.

Um die erreichte Positioniergenauigkeit besser beurteilen zu können, wurden in einigen Experimenten auch die Positionierfehler in dreidimensionalen Weltkoordinaten $err_{world}(i)$ ermittelt; dies setzt voraus, daß stets sowohl die Weltkoordinaten des anzufahrenden Zielobjekts als auch des Effektors bekannt sind, was aufgrund des Versuchsaufbaus jedoch gewährleistet ist. Man wird intuitiv erwarten, daß sich in jenen Fällen, in welchen eine Korrekturbewegung zu einer Verringerung des Positionierfehlers in Stereobildkoordinaten führt ($err_{cam}(i) < err_{cam}(i-1)$), auch der Positionierfehler in Weltkoordinaten verringern wird; daß dies jedoch nicht zwingend so sein muß, soll Abbildung 4.1 illustrieren. Jedoch tritt der Fall, daß eine Verringerung von err_{cam} zu einer Vergrößerung von err_{world} führt, erfahrungsgemäß sehr selten und meist nur zu Beginn des Trainings ein.

Die im nachfolgenden angegebenen Fehlerwerte ergaben sich durch Mittelwertbildung über jeweils hundert Musterpräsentationen und wurden während des Trainings ermittelt. Es wurde kein Testset verwendet, da dies beim Hardwareexperiment aufgrund des damit verbundenen Aufwands nicht möglich war; um die Ergebnisse mit jenen der Softwaresimulationen vergleichbar zu machen, wurde auch bei diesen auf die Verwendung eines Testsets verzichtet.

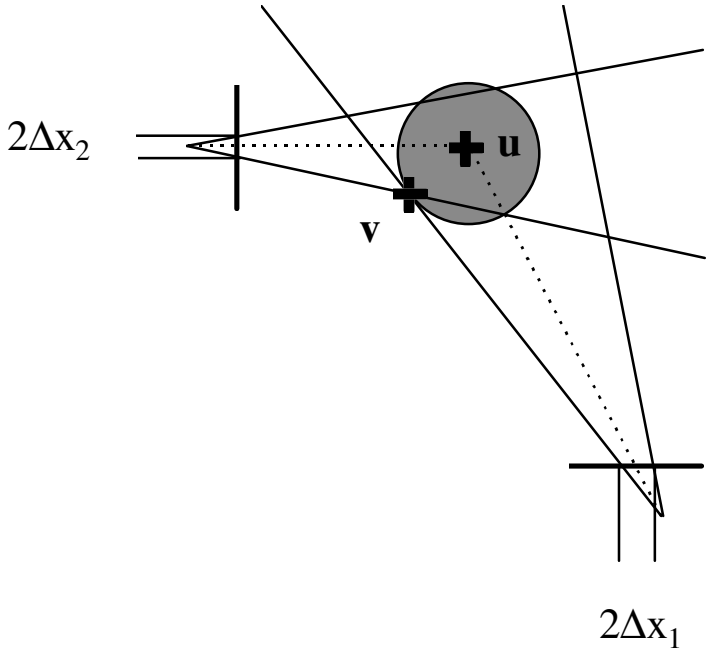

Abb. 4.1: Vereinfachte Darstellung der Projektion der Ziel- und Effektorposition auf die beiden Kamerabildebenen in einem zweidimensionalen Arbeitsbereich. Die Kamerabildebenen werden durch die beiden fetten Striche symbolisiert. Die mit einem Kreuz markierte Zielposition **u** befindet sich im Zentrum des Kreises und wird auf den Punkt x_1 (x_2) der Bildebene von Kamera 1 (Kamera 2) projiziert (Schnittpunkt der Bildebene mit der gestrichelten Linie zwischen **u** und dem Projektionszentrum); das Kreuz am Rand des Kreises gibt die aktuelle Effektorposition **v** an. Alle Punkte im Inneren des Kreis haben in zweidimensionalen Weltkoordinaten eine kleinere euklidische Distanz vom Zielvektor als **u**, wohingegen jene Punkte, deren Projektion auf die Kamerabildebenen sich um weniger als Δx_1 (Kamera 1) und Δx_2 (Kamera 2) von der Projektion von **u** unterscheidet, sicherlich einen kleineren Fehler in Stereobildkoordinaten liefern; dieser Bereich wird durch die insgesamt 4 von den beiden Projektionszentren ausgehenden Strahlen begrenzt. Wie man sieht, liegen jedoch auch Punkte in diesem Bereich, die nicht zur Kreisfläche gehören, also einen größeren Positionierfehler in Weltkoordinaten liefern.

.

4.3. Softwareexperimente

4.3.1. Softwareexperiment 1

Dieses Experiment dient als Referenz für alle weiteren Softwareexperimente sowie die Hardwareversuchsanordnung. Der Ausschnitt des Arbeitsbereichs, auf dem die inverse Kinematik erlernt werden soll, ist eine Untermenge eines Quaders mit Abmessungen 140 x 60 x 100 mit der linken unteren Ecke an der Position (420, 190, 60) (alle Angaben in mm). Das in Abbildung 4.2 dargestellte Trainingsset umfaßt 1000 zufällig ausgewählte Punkte aus diesem Bereich; die Teilmenge des Quaders, in der sich keine Trainingsvektoren befinden, gehört nicht zum *reachable workspace* des auf 3 Gelenkachsen eingeschränkten A465, ist also nicht mit der Effektorspitze erreichbar.

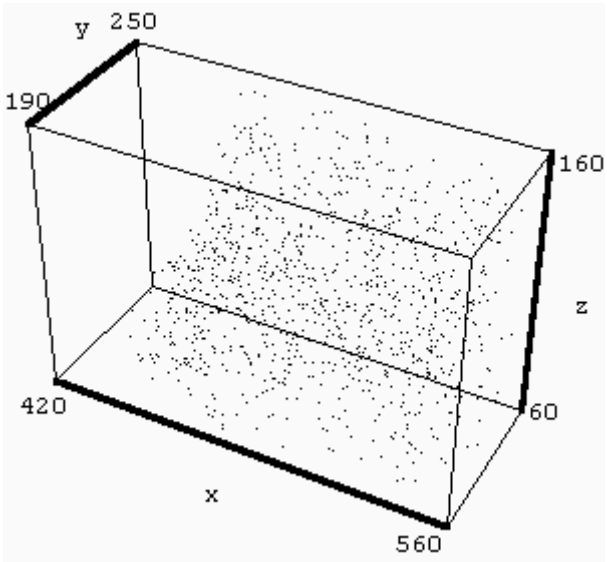

Abb. 4.2: Das Trainingsset für Experiment 1.

Kamera 1 befand sich an Position (1037, 261, 182), Kamera 2 an Position (478, 704, 167). Der Schnittpunkt der optischen Achsen mit der x-y-Ebene des Weltkoordinatensystems befand sich an den Koordinaten (-257, 208, 0) respektive (537, -882, 0) [37]. Abbildung 4.3 zeigt das Trainingsset aus der Sicht der beiden Kameras.

[37] Diese Daten wurde durch Kalibrierung der „echten“ Kameras gewonnen.

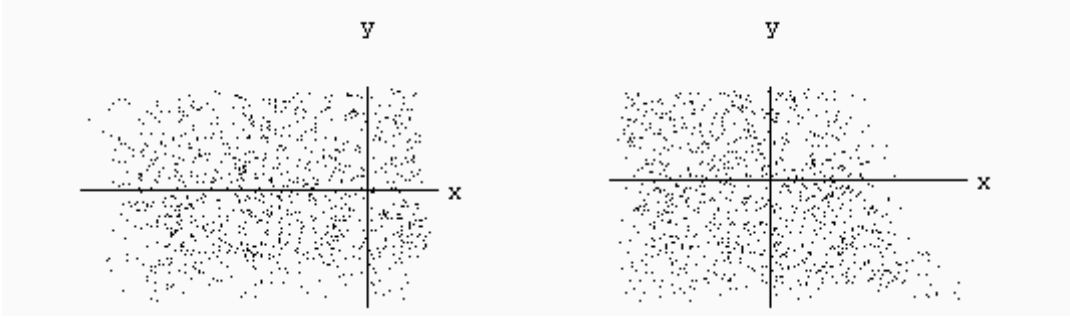

Abb. 4.3: Das Trainingsset aus der Sicht von Kamera 1 (links) und Kamera 2 (rechts).

In Abbildung 4.4 ist der zeitliche Verlauf der mittleren Positionierfehler in 4D-Stereobildkoordinaten dargestellt, wobei Fehler größer als 2mm geklippt wurden. Die 5 aneinandergrenzenden Balken geben jeweils den Fehler der open loop-Positionierung und der 4 Korrekturbewegungen an; der Diskretisierungsfehler wurde aus Gründen der Übersichtlichkeit nicht graphisch dargestellt. Die genauen Werte können der Tabelle 4.1 entnommen werden. Um einen Vergleich mit dem in Abschnitt **4.4** beschriebenen Hardwareexperiment zu ermöglichen, wurden in Abbildung 4.5 und Tabelle 4.2 außerdem die Ausgabefehler in 3D-Weltkoordinaten wiedergegeben.

Wie man sieht, konvergiert das Netzwerk relativ rasch; so ist der Ausgabefehler in Weltkoordinaten nach der vierten Korrekturbewegung $err_{world}(4)$ bereits nach 500 Musterpräsentationen auf unter 1 mm und nach 2000 Lernschritten auf einen Wert deutlich unter 0.1 gefallen. Nach ungefähr 2000 Musterpräsentationen flacht die Lernkurve jedoch deutlich ab, und die Präsentation weiterer 2000 Trainingsvektoren führt nur mehr zu einer geringen Verbesserung der Positioniergenauigkeit. Man bemerkt, daß die Lernkurve periodisch lokale Extrema aufweist (z.B. die lokalen Maxima nach 1700, 2700 und 3700 Lernschritten); dies ist darauf zurückzuführen, daß das aus 1000 Vektoren bestehende Trainingsset viermal hintereinander präsentiert wurde und die Ausgabefehler während des Trainings (also ohne Testset) ermittelt wurden.

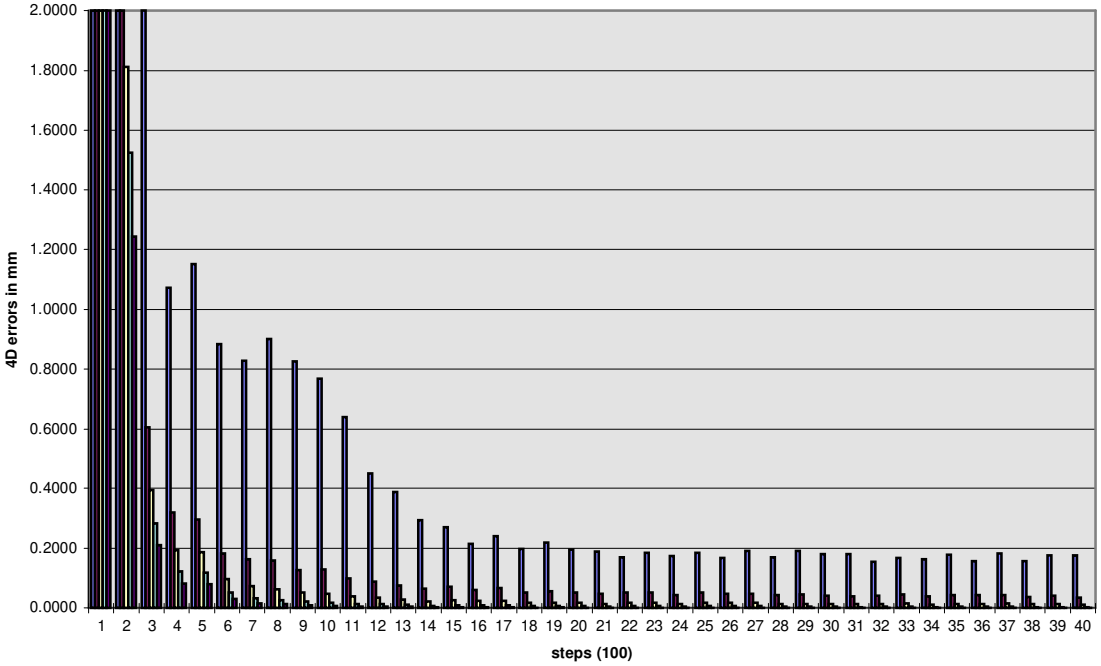

Abb 4.4: Ausgabefehler in 4D-Stereobildkoordinaten, Experiment 1. Werte größer als 2 mm wurden geklippt.

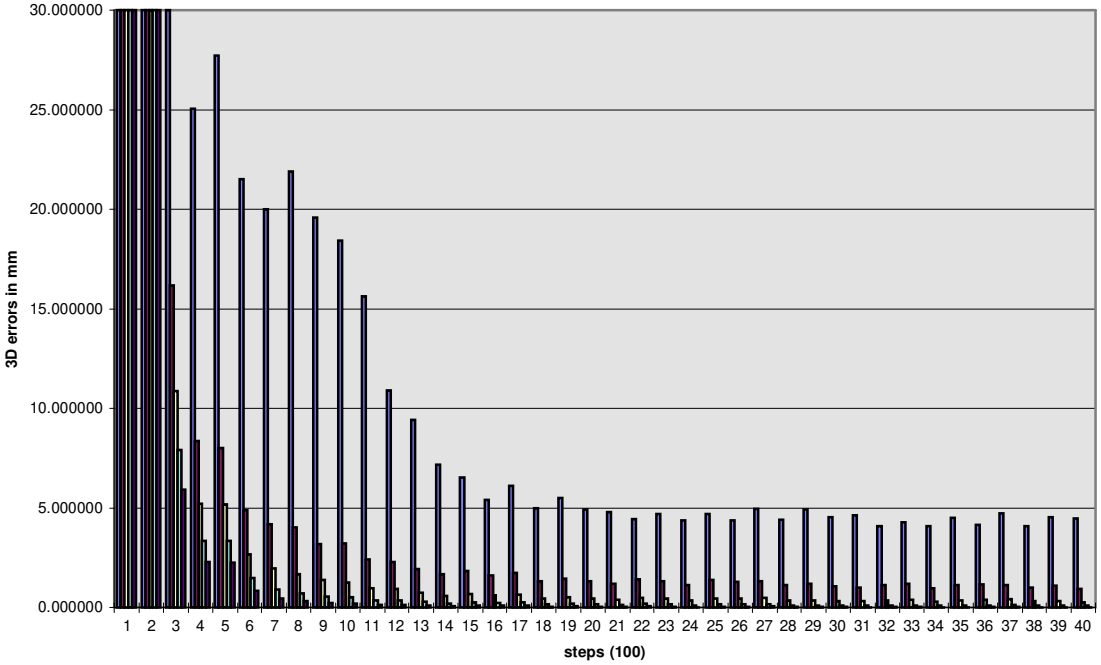

Abb. 4.5: Ausgabefehler in 3D-Weltkoordinaten, Experiment 1. Werte größer als 30 mm wurden geklippt.

steps	competing l.	open-loop	iteration 1	iteration 2	iteration 3	iteration 4
100	3.0568	40.1656	23.6909	22.3903	23.1578	23.0714
200	2.4264	12.0913	2.2727	1.8129	1.5247	1.2436
300	2.0987	2.2873	0.6050	0.3940	0.2834	0.2093
400	1.4710	1.0720	0.3207	0.1933	0.1213	0.0823
500	1.5133	1.1524	0.2966	0.1884	0.1197	0.0805
600	1.1471	0.8843	0.1835	0.0969	0.0528	0.0299
700	1.0375	0.8281	0.1629	0.0740	0.0335	0.0164
800	0.8720	0.9003	0.1604	0.0627	0.0270	0.0120
900	0.7405	0.8269	0.1274	0.0539	0.0215	0.0087
1000	0.6555	0.7686	0.1286	0.0492	0.0195	0.0079
1100	0.5771	0.6386	0.0984	0.0388	0.0135	0.0054
1200	0.5971	0.4508	0.0881	0.0353	0.0135	0.0054
1300	0.5743	0.3902	0.0758	0.0287	0.0111	0.0045
1400	0.5421	0.2936	0.0658	0.0225	0.0080	0.0029
1500	0.5953	0.2700	0.0704	0.0262	0.0098	0.0037
1600	0.5655	0.2165	0.0613	0.0234	0.0089	0.0035
1700	0.5760	0.2406	0.0669	0.0250	0.0096	0.0037
1800	0.5726	0.1983	0.0511	0.0173	0.0062	0.0022
1900	0.5482	0.2189	0.0562	0.0194	0.0068	0.0025
2000	0.5563	0.1962	0.0509	0.0166	0.0059	0.0021
2100	0.5376	0.1895	0.0464	0.0148	0.0049	0.0017
2200	0.5703	0.1685	0.0532	0.0185	0.0066	0.0023
2300	0.5534	0.1849	0.0511	0.0176	0.0060	0.0021
2400	0.5285	0.1736	0.0446	0.0135	0.0043	0.0013
2500	0.5753	0.1852	0.0524	0.0173	0.0059	0.0019
2600	0.5530	0.1676	0.0476	0.0171	0.0060	0.0022
2700	0.5590	0.1915	0.0492	0.0179	0.0062	0.0022
2800	0.5490	0.1690	0.0432	0.0143	0.0047	0.0015
2900	0.5346	0.1932	0.0457	0.0146	0.0046	0.0015
3000	0.5438	0.1798	0.0409	0.0129	0.0042	0.0014
3100	0.5253	0.1814	0.0386	0.0124	0.0038	0.0012
3200	0.5588	0.1553	0.0428	0.0139	0.0045	0.0015
3300	0.5411	0.1672	0.0452	0.0150	0.0048	0.0016
3400	0.5225	0.1627	0.0385	0.0113	0.0033	0.0010
3500	0.5626	0.1781	0.0442	0.0137	0.0043	0.0013
3600	0.5410	0.1586	0.0432	0.0146	0.0048	0.0016
3700	0.5512	0.1825	0.0432	0.0153	0.0050	0.0017
3800	0.5391	0.1566	0.0378	0.0122	0.0038	0.0012
3900	0.5272	0.1774	0.0420	0.0127	0.0037	0.0012
4000	0.5375	0.1775	0.0361	0.0109	0.0033	0.0010

Tabelle 4.1: Diskretisierungsfehler und Ausgabefehler in 4D-Stereobildkoordinaten, Experiment 1.

steps	open-loop	iteration 1	iteration 2	iteration 3	iteration 4
100	834.8649	480.7444	493.3622	522.7953	532.6116
200	235.0307	56.6116	46.1537	39.1564	32.6047
300	51.4057	16.1745	10.8825	7.9016	5.9038
400	25.0575	8.3704	5.2119	3.3502	2.3108
500	27.7320	8.0263	5.1975	3.3515	2.2704
600	21.5367	4.9173	2.6644	1.4713	0.8429
700	20.0261	4.2124	1.9653	0.9138	0.4535
800	21.9050	4.0373	1.6576	0.7245	0.3231
900	19.5786	3.2001	1.3995	0.5582	0.2291
1000	18.4180	3.2134	1.2711	0.5076	0.2086
1100	15.6242	2.4245	0.9939	0.3521	0.1428
1200	10.9165	2.3112	0.9330	0.3580	0.1449
1300	9.4279	1.9512	0.7551	0.2951	0.1195
1400	7.1775	1.6705	0.5830	0.2068	0.0765
1500	6.5291	1.8252	0.6834	0.2553	0.0971
1600	5.4329	1.6162	0.6138	0.2336	0.0909
1700	6.0966	1.7480	0.6632	0.2552	0.1006
1800	4.9973	1.3318	0.4563	0.1628	0.0567
1900	5.4967	1.4604	0.5066	0.1790	0.0665
2000	4.9375	1.3231	0.4352	0.1550	0.0560
2100	4.7921	1.2015	0.3875	0.1292	0.0450
2200	4.4332	1.4252	0.5009	0.1775	0.0630
2300	4.7121	1.3465	0.4668	0.1611	0.0570
2400	4.3880	1.1512	0.3526	0.1117	0.0352
2500	4.7193	1.3745	0.4560	0.1545	0.0511
2600	4.3786	1.2800	0.4576	0.1588	0.0574
2700	4.9614	1.3064	0.4820	0.1682	0.0617
2800	4.4104	1.1478	0.3811	0.1242	0.0407
2900	4.9324	1.1909	0.3801	0.1199	0.0400
3000	4.5370	1.0680	0.3392	0.1095	0.0359
3100	4.6189	1.0107	0.3290	0.1011	0.0331
3200	4.0914	1.1541	0.3773	0.1220	0.0401
3300	4.3069	1.1949	0.3986	0.1273	0.0418
3400	4.1186	0.9970	0.2967	0.0880	0.0262
3500	4.5176	1.1500	0.3581	0.1115	0.0333
3600	4.1552	1.1636	0.3893	0.1274	0.0431
3700	4.7365	1.1569	0.4151	0.1368	0.0480
3800	4.0839	1.0094	0.3253	0.0999	0.0310
3900	4.5437	1.0927	0.3309	0.0977	0.0305
4000	4.4736	0.9415	0.2878	0.0862	0.0265

Tabelle 4.2: Ausgabefehler in 3D-Weltkoordinaten, Experiment 1.

4.3.2. Softwareexperiment 2

Wir verwenden hier denselben Versuchsaufbau wie in Experiment 1, schränken den Eingabesignalbereich jedoch auf ungefähr ein Drittel des ursprünglichen Volumens ein; die tausend Trainingsvektoren wurden zufällig dem quaderförmigen Bereich [490 - 560] x [190 - 250] x [60 - 110] (welcher eine Teilmenge des *reachable workspace* ist) entnommen. Die zeitliche Entwicklung der Ausgabefehler kann Abbildung 4.6 bzw. Tabelle 4.3 entnommen werden.

Der - wie ein Vergleich der ersten Spalte von Tabelle 3.1 und Tabelle 3.3 zeigt - gegenüber Experiment 1 deutlich kleinere Diskretisierungsfehler resultiert daraus, daß ein kleinerer Eingabesignalbereich durch eine gleichbleibende Anzahl von Referenzvektoren natürlich wesentlich besser abgedeckt werden kann. Der Zugewinn an lokaler Beschreibungsgenauigkeit wirkt sich erwartungsgemäß positiv auf den mittleren Fehler der open loop-Positionierung aus: dieser nimmt zwar zunächst etwas langsamer ab, fällt nach 1700 Musterpräsentationen aber dann geringer aus als in Experiment 1. Betrachten wir nochmals die Kostenfunktion für die Gewichte α_{ji} (2.23): diese fordert, daß bereits mit der open loop-Positionierung eine möglichst hohe Positioniergenauigkeit erreicht werden soll. Die Gewichte α_{ji}, welche nur die Gelenkwinkel für die open loop-Positionierung unmittelbar beeinflussen, übernehmen somit in Gleichung (2.15) (Berechnung der Gelenkwinkelwerte der open loop-Bewegung) - unter Vernachlässigung der Terme $\mathbf{a}_{ji}$.**u** - die Rolle von Stützstellen im Raum der Ausgabewerte, wohingegen sich die Korrekturbewegungen als Näherung 1.Grades berechnen. Je feiner die Diskretisierung der Eingabesignalmenge, desto eher ist es dem Netzwerk also möglich, die den Trainingsdaten zugrundeliegende Ein- Ausgaberelation unter Verwendung dieser Ausgabestützstellen „auswendigzulernen".

Man bemerkt nun, daß die mittleren Ausgabefehler der vier Korrekturbewegungen bei Experiment 2 durchgehend größere Werte als bei Experiment 1 aufweisen (bei $err_{cam}(4)$ nach 4000 Lernschritten sogar um einen Faktor 10!). Dies läßt sich zwar einerseits aus dem aus der Einschränkung des Trainingssets resultierenden Informationsverlust bezüglich der globalen Struktur der Arbeitsbereichs erklären; folgen wir jedoch der obigen Argumentation, so sehen wir, daß eine Verringerung des open loop-Positionierfehlers aufgrund der feineren Diskretisierung der Eingabesignalmenge zu kleineren Korrekturbewegungen (Gleichung 2.16) und in der Folge zu kleineren Gewichtsänderungen für die Ausgabevektoren $\mathbf{a}_{ji}$ (Gleichung 2.17), welche vornehmlich für die Berechnung der Korrekturbewegungen von Bedeutung sind, führt.

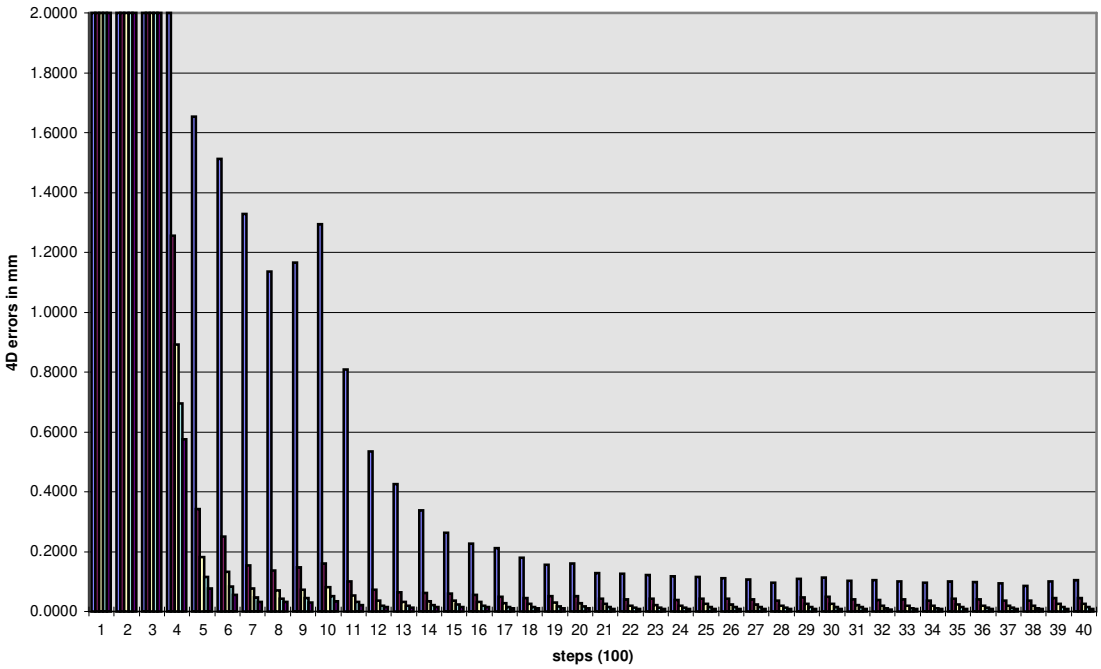

Abb. 4.6: Ausgabefehler in 4D-Stereobildkoordinaten, Experiment 2.

steps	competing l.	open loop	iteration 1	iteration 2	iteration 3	iteration 4
100	1.9197	37.6704	20.9063	18.4273	17.9945	17.9058
200	1.1662	13.4430	3.6176	2.6447	2.6001	2.5946
300	0.9864	6.2043	2.9375	2.8345	2.7806	2.7316
400	0.7972	3.7117	1.2570	0.8928	0.6960	0.5743
500	0.6475	1.6545	0.3428	0.1810	0.1160	0.0769
600	0.5450	1.5136	0.2522	0.1334	0.0824	0.0561
700	0.4486	1.3293	0.1553	0.0783	0.0468	0.0313
800	0.3856	1.1382	0.1373	0.0696	0.0440	0.0306
900	0.3622	1.1674	0.1484	0.0714	0.0450	0.0303
1000	0.3212	1.2940	0.1603	0.0807	0.0510	0.0351
1100	0.2833	0.8097	0.1018	0.0529	0.0323	0.0223
1200	0.3043	0.5347	0.0712	0.0358	0.0206	0.0134
1300	0.2761	0.4261	0.0642	0.0332	0.0191	0.0126
1400	0.2755	0.3389	0.0624	0.0351	0.0212	0.0142
1500	0.2625	0.2623	0.0612	0.0358	0.0228	0.0154
1600	0.2640	0.2261	0.0570	0.0331	0.0210	0.0141
1700	0.2610	0.2107	0.0503	0.0275	0.0159	0.0103
1800	0.2535	0.1801	0.0453	0.0257	0.0158	0.0104
1900	0.2737	0.1570	0.0522	0.0291	0.0177	0.0114
2000	0.2759	0.1614	0.0521	0.0287	0.0174	0.0112
2100	0.2676	0.1280	0.0431	0.0246	0.0153	0.0101
2200	0.2863	0.1272	0.0407	0.0210	0.0120	0.0074
2300	0.2612	0.1219	0.0430	0.0221	0.0129	0.0083
2400	0.2664	0.1182	0.0397	0.0210	0.0123	0.0078
2500	0.2509	0.1152	0.0443	0.0248	0.0151	0.0096
2600	0.2589	0.1108	0.0433	0.0232	0.0140	0.0090
2700	0.2559	0.1081	0.0412	0.0230	0.0141	0.0092
2800	0.2478	0.0974	0.0379	0.0209	0.0128	0.0082
2900	0.2672	0.1087	0.0464	0.0254	0.0153	0.0097
3000	0.2682	0.1138	0.0477	0.0259	0.0155	0.0098
3100	0.2639	0.1026	0.0399	0.0219	0.0133	0.0086
3200	0.2789	0.1049	0.0392	0.0197	0.0112	0.0068
3300	0.2563	0.1010	0.0399	0.0201	0.0114	0.0071
3400	0.2599	0.0970	0.0379	0.0196	0.0114	0.0071
3500	0.2457	0.1005	0.0418	0.0228	0.0137	0.0086
3600	0.2560	0.0992	0.0403	0.0210	0.0125	0.0080
3700	0.2524	0.0944	0.0380	0.0208	0.0126	0.0082
3800	0.2455	0.0867	0.0367	0.0195	0.0117	0.0074
3900	0.2620	0.1016	0.0450	0.0244	0.0145	0.0090
4000	0.2636	0.1050	0.0459	0.0248	0.0148	0.0093

Tabelle 4.3: Diskretisierungsfehler und Ausgabefehler in 4D-Stereobildkoordinaten, Experiment 2.

4.3.3. Softwareexperiment 3

Wir trainieren das Netzwerk wiederum auf einer Untermenge des aus Experiment 1 bekannten Arbeitsbereichs, welche diesmal jedoch aus zwei unzusammenhängenden Teilmengen (siehe Abbildung 4.7) besteht. Die 1000 Trainingsvektoren wurden zufällig dem Bereich ([420 - 480] x [190 - 250] x [60 - 160]) U ([520 - 560] x [190 - 250] x [60 - 160]) entnommen. Der zeitliche Verlauf der Positionierfehler ist in Abbildung 4.8 und Tabelle 4.4 dargestellt.

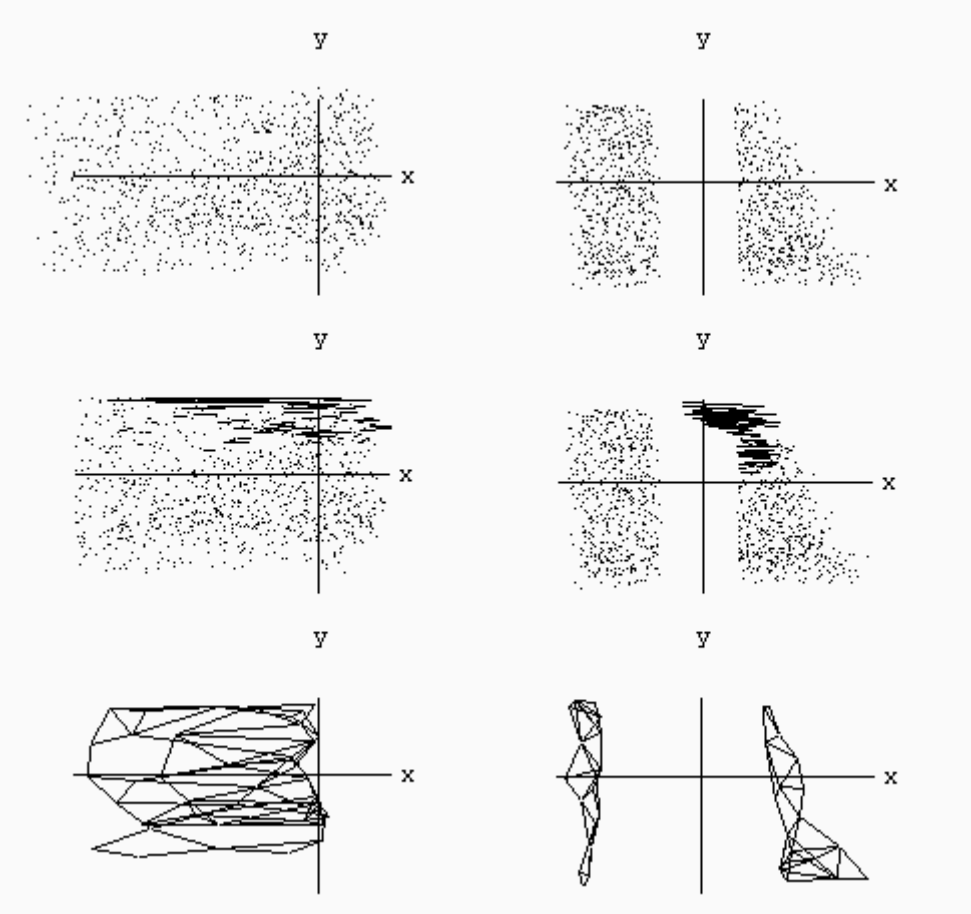

Abb. 4.7: Dargestellt sind die Verteilung der Trainingsvektoren (erste Zeile), die Positionierfehler nach der vierten Korrekturbewegung (mittlere Zeile) sowie die mittels TRN gelernten Verbindungen zwischen den *competing units* (letzte Zeile) nach 4000 Musterpräsentationen, jeweils als Projektion auf die beiden Kamerabildebenen. Im mittleren Bildpaar wurden jeweils die Zielposition **u** und die Effektorposition nach der vierten Iteration **v**(4) durch eine Gerade verbunden. Wie man in der letzten Zeile rechts sieht, teilt TRN die *competing units* korrekt auf zwei separate Teilnetze auf; allerdings wird diese Information über die Topologie der Eingabesignalmenge vom erweiterten *Neural Gas*-Verfahren nicht verwendet.

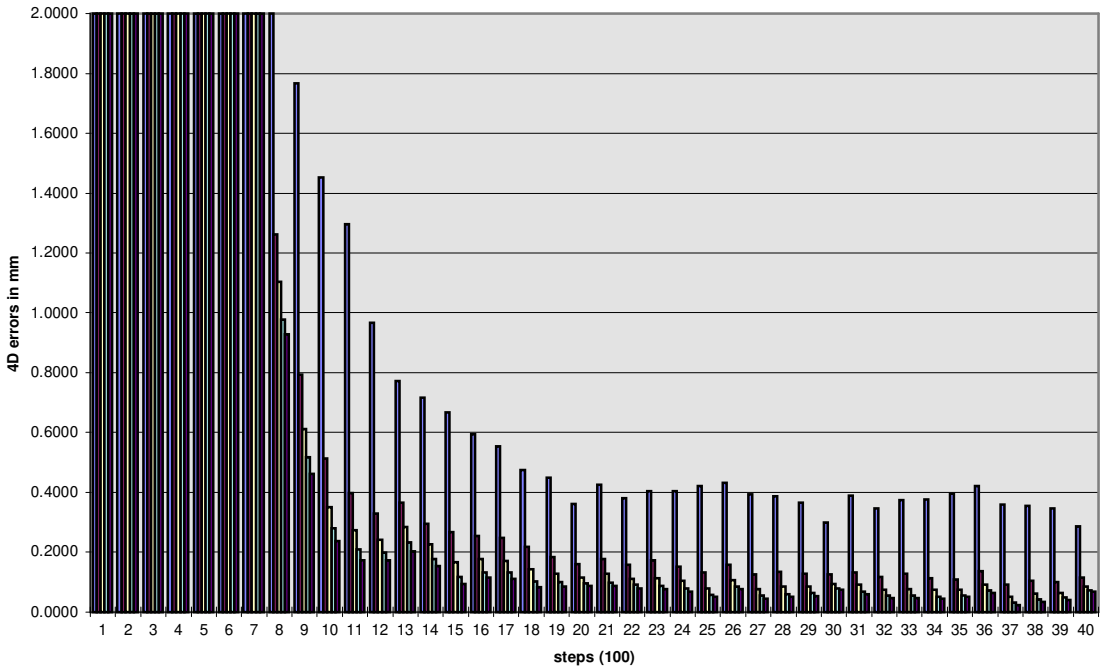

Abb. 4.8: Ausgabefehler in 4D-Stereobildkoordinaten, Experiment 3.

Die Ergebnisse fallen im Vergleich zu den beiden vorhergehenden Experimenten sehr schlecht aus. Betrachtet man Abbildung 4.7, so bemerkt man, daß die Menge jener Trainingsvektoren, welche für die großen mittleren Positionierfehler verantwortlich sind, in einem relativ kleinen Bereich lokalisiert ist. Dieser befindet sich (von Kamera 2 aus gesehen) am linken oberen Rand der rechten Teilmenge der Signalverteilung, entlang des Bereichs mit nicht positiver Signalwahrscheinlichtkeit, welcher die Signalverteilung in zwei Hälften trennt (im folgenden kurz „Spalt“ genannt). Ich habe dieses Phänomen in der Vergangenheit schon häufiger bei Simulationen mit unzusammenhängenden Verteilungen beobachtet [Melzer, 1995]; seine Ursache ist darin zu sehen, daß die topologische Struktur der Eingabesignalmenge beim Training der Ausgabewerte nicht explizit berücksichtigt wird. Deshalb können *units*, deren zugeordneter Referenzvektor in der Nähe des Spalts liegt und die eine hohe Aktivierung aufweisen, die Ausgabegewichte anderer *units* über den Spalt hinweg in unstetiger Weise beeinflussen. Ist die Verteilung der Trainingsvektoren, wie im vorliegenden Beispiel, auf einer Seite des Spalts dichter als auf der anderen Seite, so wird dieses Phänomen noch ausgeprägter (man sieht, daß die vom Netz berechnete Effektorposition für jene Trainingsvektoren, die einen großen Positionierfehler verursachen, tatsächlich in Richtung jener Hälfte der Verteilung mit lokal größerer Signaldichte liegt).

steps	competing l.	open loop	iteration 1	iteration 2	iteration 3	iteration 4
100	3.7984	31.7167	23.8501	22.6511	22.2460	22.0245
200	2.5521	18.5221	17.0056	16.8085	16.7236	16.6755
300	2.0263	15.9874	15.2543	15.1922	15.4379	15.9184
400	1.6379	13.8086	12.2912	12.1797	12.9177	14.2344
500	1.4079	11.0817	10.0856	10.0531	10.0872	10.1664
600	1.1211	10.0681	8.3617	8.0421	8.0940	7.9813
700	0.8739	5.0858	3.3426	3.0207	3.0620	2.9770
800	0.6820	2.1362	1.2632	1.1036	0.9786	0.9279
900	0.6266	1.7670	0.7935	0.6126	0.5178	0.4603
1000	0.4784	1.4524	0.5135	0.3511	0.2803	0.2378
1100	0.4828	1.2976	0.3966	0.2732	0.2091	0.1740
1200	0.4548	0.9675	0.3297	0.2411	0.1989	0.1733
1300	0.4423	0.7712	0.3653	0.2843	0.2331	0.2030
1400	0.4697	0.7167	0.2953	0.2259	0.1789	0.1535
1500	0.4629	0.6665	0.2675	0.1675	0.1179	0.0930
1600	0.4684	0.5931	0.2545	0.1760	0.1319	0.1141
1700	0.4581	0.5530	0.2481	0.1703	0.1311	0.1108
1800	0.4197	0.4755	0.2182	0.1416	0.1014	0.0833
1900	0.4606	0.4502	0.1851	0.1285	0.0998	0.0862
2000	0.4202	0.3602	0.1610	0.1142	0.0956	0.0877
2100	0.4636	0.4252	0.1788	0.1280	0.0991	0.0870
2200	0.4385	0.3815	0.1578	0.1105	0.0901	0.0791
2300	0.4268	0.4043	0.1728	0.1128	0.0871	0.0761
2400	0.4496	0.4050	0.1519	0.1051	0.0798	0.0690
2500	0.4537	0.4201	0.1313	0.0791	0.0572	0.0509
2600	0.4504	0.4329	0.1578	0.1070	0.0842	0.0759
2700	0.4486	0.3940	0.1272	0.0773	0.0544	0.0435
2800	0.4120	0.3853	0.1340	0.0841	0.0598	0.0509
2900	0.4503	0.3662	0.1286	0.0850	0.0634	0.0531
3000	0.4101	0.2981	0.1270	0.0932	0.0785	0.0731
3100	0.4555	0.3884	0.1309	0.0912	0.0672	0.0593
3200	0.4312	0.3452	0.1171	0.0749	0.0551	0.0470
3300	0.4198	0.3746	0.1277	0.0759	0.0562	0.0476
3400	0.4387	0.3773	0.1130	0.0729	0.0523	0.0450
3500	0.4475	0.3957	0.1073	0.0737	0.0547	0.0503
3600	0.4429	0.4197	0.1380	0.0928	0.0712	0.0639
3700	0.4418	0.3586	0.0907	0.0495	0.0307	0.0235
3800	0.4080	0.3553	0.1051	0.0606	0.0418	0.0347
3900	0.4452	0.3468	0.1005	0.0645	0.0479	0.0411
4000	0.4039	0.2852	0.1145	0.0855	0.0726	0.0684

Tabelle 4.4: Diskretisierungsfehler und Ausgabefehler in 4D-Stereobildkoordinaten, Experiment 3.

4.3.4. Softwareexperiment 4

Dieses Experiment soll die Sensitivität des erweiterten *Neural-Gas* Modells bezüglich der Nachbarschaftsreichweite $\lambda(t)$ illustrieren. Wir verwenden die Netzwerkarchitektur und das Trainingsset aus Experiment 1, lassen $\lambda(t)$ aber nun - wie auch $\varepsilon(t)$ und $\varepsilon'(t)$ - in 4000 statt wie bisher in 1000 Schritten auf seinen Minimalwert fallen, d.h. wir wählen in Gleichung (4.1):$\lambda_{start} = 10$, $\lambda_{end} = 1$ und $\lambda_{max} = 4000$. Die Ergebnisse dieser Simulation sind in Abbildung 4.9 und Tabelle 4.5 dargestellt.

Das Netzwerk konvergiert anfänglich (wenn auch deutlich langsamer als in Experiment 1), nach ungefähr 1500 Lernschritten beginnt der open loop-Positionierfehler jedoch wieder zu steigen, wohingegen die Positionierfehler der Korrekturbewegungen annähernd konstant bleiben. Eine Verbesserung der Positioniergenauigkeit tritt erst wieder nach 4000 Lernschritten, sobald $\lambda(t)$ seinen Endwert 1 erreicht hat, ein. Dieser „Inversionseffekt“ - also die Verschlechterung der Positioniergenauigkeit nach anfänglicher Konvergenz - fällt i.a. umso krasser aus, je weiter λ_{start} und λ_{end} auseinanderliegen und je größer λ_{max} ist, kann jedoch bis zu einem gewissen Grad durch geeignete Skalierung der Gewichtsdeltas für die Ausgabegewichte (siehe Abschnitt **3.3.1**) gedämpft werden.

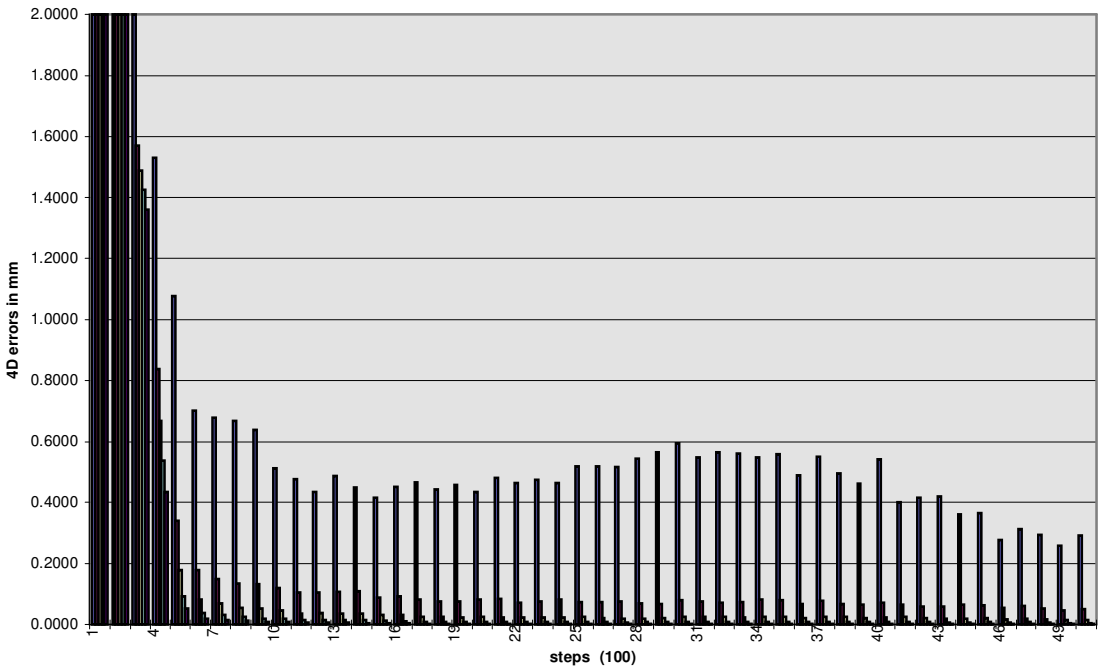

Abb. 4.9: Ausgabefehler in 4D-Stereobildkoordinaten, Experiment 4.

steps	competing l.	open loop	iteration 1	iteration 2	iteration 3	iteration 4
100	3.2083	40.3152	23.6881	22.9509	23.9438	23.9794
200	2.8752	13.1623	3.3860	3.1302	2.8394	2.5822
300	2.9289	2.6891	1.5710	1.4881	1.4251	1.3594
400	2.2482	1.5316	0.8386	0.6663	0.5360	0.4352
500	2.6960	1.0776	0.3396	0.1775	0.0926	0.0525
600	2.3131	0.7012	0.1777	0.0822	0.0376	0.0196
700	2.3807	0.6788	0.1491	0.0684	0.0310	0.0157
800	2.2175	0.6674	0.1354	0.0552	0.0243	0.0125
900	2.0098	0.6383	0.1338	0.0517	0.0206	0.0097
1000	2.1022	0.5118	0.1188	0.0454	0.0188	0.0081
1100	1.9193	0.4766	0.1054	0.0356	0.0133	0.0054
1200	1.8540	0.4359	0.1056	0.0384	0.0158	0.0068
1300	1.8128	0.4869	0.1068	0.0369	0.0147	0.0060
1400	1.4694	0.4488	0.1088	0.0365	0.0132	0.0050
1500	1.7045	0.4151	0.0877	0.0314	0.0115	0.0045
1600	1.4752	0.4511	0.0912	0.0325	0.0112	0.0042
1700	1.4796	0.4655	0.0808	0.0263	0.0095	0.0036
1800	1.3920	0.4428	0.0773	0.0249	0.0086	0.0031
1900	1.3000	0.4581	0.0761	0.0233	0.0071	0.0022
2000	1.3110	0.4335	0.0819	0.0243	0.0071	0.0024
2100	1.1898	0.4815	0.0836	0.0234	0.0074	0.0025
2200	1.2045	0.4643	0.0725	0.0231	0.0077	0.0026
2300	1.1406	0.4731	0.0747	0.0228	0.0074	0.0024
2400	0.9647	0.4650	0.0810	0.0229	0.0073	0.0024
2500	1.0933	0.5185	0.0742	0.0238	0.0075	0.0025
2600	0.9640	0.5175	0.0735	0.0219	0.0073	0.0024
2700	0.9627	0.5167	0.0749	0.0202	0.0066	0.0023
2800	0.9111	0.5439	0.0684	0.0198	0.0070	0.0025
2900	0.8566	0.5653	0.0670	0.0207	0.0069	0.0023
3000	0.8529	0.5926	0.0798	0.0240	0.0081	0.0027
3100	0.7670	0.5485	0.0764	0.0257	0.0088	0.0032
3200	0.7863	0.5647	0.0709	0.0242	0.0086	0.0030
3300	0.7486	0.5603	0.0742	0.0226	0.0079	0.0027
3400	0.6436	0.5481	0.0832	0.0252	0.0084	0.0029
3500	0.7302	0.5584	0.0795	0.0254	0.0086	0.0029
3600	0.6532	0.4885	0.0671	0.0211	0.0073	0.0025
3700	0.6505	0.5506	0.0781	0.0249	0.0090	0.0032
3800	0.6194	0.4945	0.0674	0.0238	0.0084	0.0030
3900	0.5783	0.4634	0.0646	0.0209	0.0072	0.0024
4000	0.5822	0.5414	0.0725	0.0228	0.0078	0.0026
4100	0.5254	0.4014	0.0658	0.0254	0.0092	0.0034
4200	0.5596	0.4156	0.0590	0.0211	0.0076	0.0028
4300	0.5589	0.4201	0.0605	0.0191	0.0064	0.0022
4400	0.5066	0.3619	0.0643	0.0197	0.0066	0.0022
4500	0.5949	0.3666	0.0631	0.0210	0.0071	0.0024
4600	0.5499	0.2774	0.0542	0.0171	0.0056	0.0019
4700	0.5663	0.3127	0.0612	0.0196	0.0071	0.0025
4800	0.5576	0.2936	0.0517	0.0175	0.0058	0.0020

Tabelle 4.5: Diskretisierungsfehler und Ausgabefehler in 4D-Stereobildkoordinaten, Experiment 4 (die letzten beiden Zeilen wurden aus Platzgründen weggelassen).

4.4. Hardwareexperiment

Die Netzwerkarchitektur und das Trainingsset sind identisch mit jenen aus Softwareexperiment 1. Das Netzwerk wurde zunächst 700 Lernschritte auf der emulierten Umgebung vortrainiert, die Resultate sind daher bis inklusive Lernschritt 700 mit jenen von Softwareexperiment 1 identisch. Anschließend wurde dem Netzwerk die Kontrolle über das Hardwareenvironment übergeben.[38] Wie man in Abbildung 4.10 bzw. 4.11 sieht, steigen die Ausgabefehler zunächst leicht an, sind aber nach insgesamt 900 Lernschritten bereits geringer als zum „Umschaltzeitpunkt". Die mittlere Positioniergenauigkeit nach der vierten Korrekturbewegung liegt nach 1000 Lernschritten bei 0.29 mm (in Weltkoordinaten), was in etwa dem aufgrund der begrenzten Kameraauflösung erzielbaren theoretischen Minimum von 0.28 mm entspricht[39]; die genauen Werte können den Tabellen 4.6 und 4.7 entnommen werden. Der Zeitaufwand für 100 Musterpräsentationen betrug zwischen 15 und 20 Minuten.

Um in der Realität vorkommende Fehlerquellen zu simulieren, wurden beide Kameras nach 1000 Musterpräsentationen von den Stativen abgenommen und neu montiert, ein Vorgang, der unweigerlich zu kleinen Abweichungen in den Kamerapositionen führt. Das Netzwerk wurde anschließend noch über 100 Musterpräsentationen weitertrainiert, wobei sich die mittlere Positioniergenauigkeit jedoch nur geringfügig verschlechterte.

---

[38] Die während der Kalibrierung ermittelten radialen Linsenverzerrungen wurden während des Hardwaretrainings nicht softwaremäßig kompensiert.

[39] Die effektive Kameraauflösung wurde wie folgt ermittelt: der Schwerpunkt G der Verteilung (500, 220, 110) wurde zunächst in Kamerakoordinaten transformiert und der z-Wert G_{camz} des transformierten Vektors als mittlerer Abstand der Trainingsvektoren von der x-y-Ebene des Kamerakoordinatensystems angenommen. Die Pixelbreite (ca 0.01 mm) wurde nun mit dem Faktor G_{camz} / f skaliert, was einer Umrechnung von Bild- in Kamerakoordinaten entspricht; für Kamera 1 ergab sich dabei ein Wert von 0.25 mm, für Kamera 2 ein Wert von 0.28 mm, wobei klarerweise der größere Wert die „Gesamtauflösung" nach unten begrenzt.

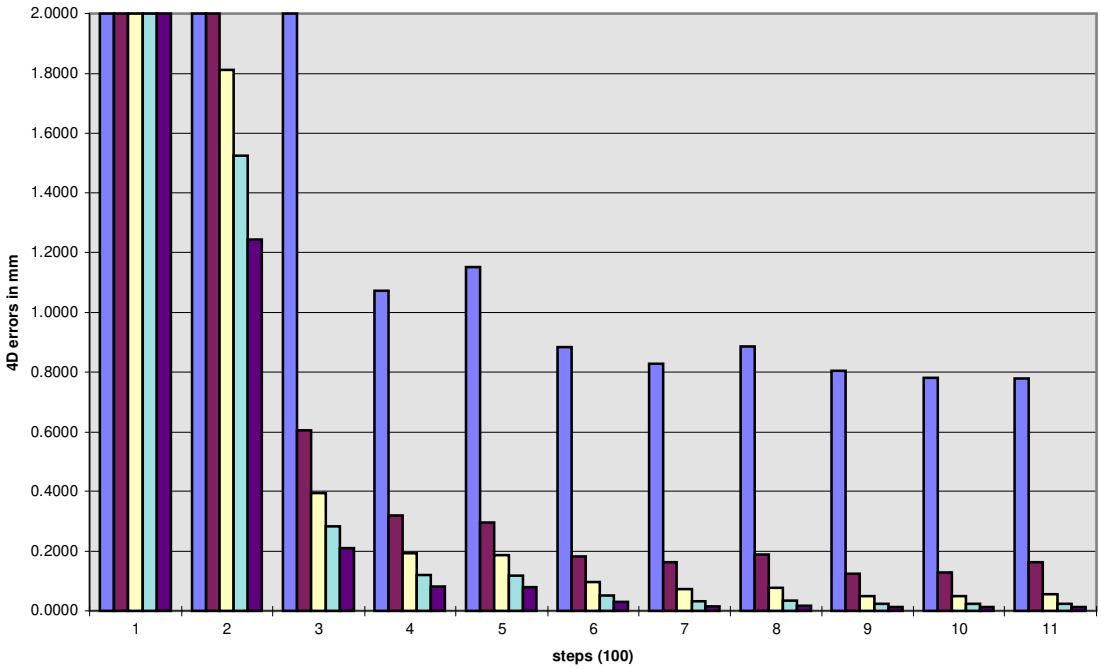

Abb. 4.10: Ausgabefehler in 4D-Stereobildkoordinaten, Hardwareexperiment.

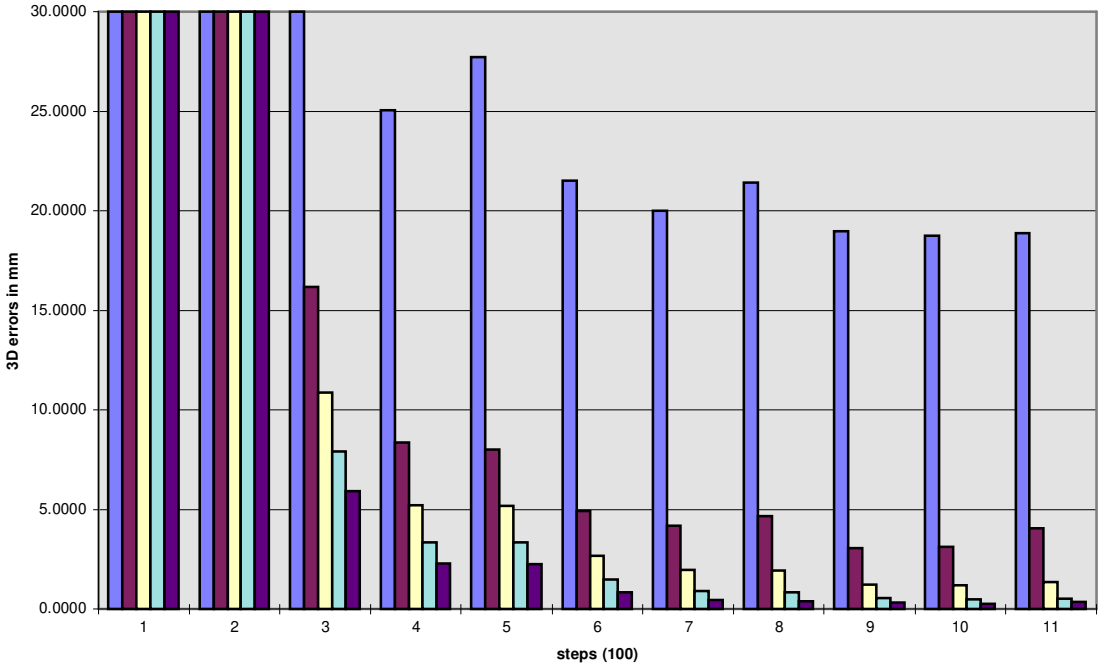

Abb. 4.11: Ausgabefehler in 3D-Weltkoordinaten, Hardwareexperiment.

steps	open loop	iteration 1	iteration 2	iteration 3	iteration 4
100	40.1656	23.6909	22.3903	23.1579	23.0716
200	12.0914	2.2727	1.8129	1.5247	1.2437
300	2.2873	0.6051	0.3940	0.2834	0.2094
400	1.0720	0.3207	0.1934	0.1213	0.0824
500	1.1524	0.2966	0.1884	0.1197	0.0805
600	0.8843	0.1835	0.0969	0.0528	0.0299
700	0.8282	0.1629	0.0740	0.0335	0.0164
800	0.8858	0.1889	0.0778	0.0357	0.0187
900	0.8037	0.1245	0.0504	0.0243	0.0144
1000	0.7816	0.1302	0.0500	0.0230	0.0145
1100	0.7786	0.1641	0.0568	0.0241	0.0135

Tabelle 4.6: Ausgabefehler in 4D-Stereobildkoordinaten, Hardwareexperiment.

steps	open loop	iteration 1	iteration 2	iteration 3	iteration 4	invalid
100	834.8649	480.7450	493.3638	522.7972	532.6141	0
200	235.0320	56.6094	46.1535	39.1565	32.6050	0
300	51.4058	16.1755	10.8839	7.9031	5.9051	0
400	25.0581	8.3715	5.2130	3.3511	2.3116	0
500	27.7323	8.0272	5.1984	3.3524	2.2712	0
600	21.5370	4.9181	2.6651	1.4718	0.8433	0
700	20.0265	4.2130	1.9658	0.9141	0.4537	0
800	21.4257	4.6804	1.9579	0.8475	0.4066	1
900	18.9919	3.0494	1.2224	0.5744	0.3326	3
1000	18.7500	3.1206	1.1917	0.5028	0.2876	2
1100	18.8985	4.0548	1.3707	0.5370	0.3471	2

Tabelle 4.7: Ausgabefehler in 3D-Weltkoordinaten, Hardwareexperiment. In der letzen Spalte ist die Anzahl der Positioniervorgänge angegeben, die während der letzten 100 Musterpräsentationen abgebrochen werden mußten (diese wurden nicht in Berechnung des mittleren Positionierfehlers miteinbezogen); die Fehlerursache lag immer im Verlust des visuellen Feedbacks, d.h. die LED befand sich in diesem Fall nicht mehr im Sichtbereich beider Kameras.

4.5. Diskussion

Wir fassen die Ergebnisse der Experimente nochmals kurz zusammen:

- In Softwareexperiment 1 wurde demonstriert, daß ein erweitertes *Neural Gas*-Netzwerk mit nur 40 *competing units* in einer simulierten Umgebung in der Lage ist, die inverse Kinematik eines Robotersystems auf einem ca. 1 dm^3 großen Arbeitsbereich in 2000 Lernschritten bis auf einen Residualfehler von weniger als 0.05 mm zu erlernen; die im Hardwareversuch erzielte Positioniergenauigkeit von ca. 0.3 mm entsprach dem aufgrund der gegebenen Kameraauflösung möglichen theoretischen Minimalwert und wurde nach nur 1000 Lernschritten erreicht.

- Eine Verkleinerung des Eingabesignalbereichs in Softwareexperiment 2 führte - bei konstanter Anzahl der *competing units* - zu einer deutlichen Verbesserung der Diskretisierungsgüte. Dies brachte zwar eine Verkleinerung des open loop-Positionierfehlers, aber auch eine Verringerung der mittels der Korrekturbewegungen erzielbaren Positioniergenauigkeit mit sich. Wir führten dieses Phänomen darauf zurück, daß das Netzwerk aufgrund des Zugewinns an lokaler Information dazu tendiert, auch die zu erlernende Ein-Ausgaberelation vorwiegend lokal (d.h. durch Generierung geeigneter Ausgabestützstellen) zu approximieren, was sich jedoch - wie auch aufgrund einer genaueren Betrachtung der Lernregeln zu erwarten war - nachteilig auf die zeitliche Entwicklung der für die Generierung der Gelenkwinkel der Korrekturbewegungen zuständigen Ausgabegewichte $\mathbf{a}_{ij}$ auswirkte.

- Die Tatsache, daß das erweiterte *Neural Gas*-Verfahren die Topologie des Eingabesignalraums nicht explizit berücksichtigt und in jedem Lernschritt alle Ausgabegewichte modifiziert, führte bei der Softwareexperiment 3 zugrundeliegenden, unzusammenhängenden Eingabesignalverteilung zu Unstetigkeitseffekten bei der Adaption der Ausgabegewichte: es bildete sich in der Folge ein Bereich mit persistenten, hohen Ausgabefehlern, welche auch nach 4000 Trainingsschritten deutlich über jenen der vorangegangenen Experimente lagen.

- Experiment 4 veranschaulichte die Sensitivität des erweiterten *Neural Gas*-Modells bezüglich des zeitlichen Verlaufs der Nachbarschaftsreichweite $\lambda(t)$: werden die Funktionsparameter nicht sorgfältig gewählt, so kann dies nach anfänglicher Konvergenz des Verfahrens zu einer fortschreitenden Verschlechterung der Ausgabefehler mit der Präsentation weiterer Trainingsvektoren führen. Es wurde angeführt, daß diese Sensitivität auch in Zusammenhang mit der Skalierung der Gewichtsdeltas der Ausgabegewichte steht.

5. Zusammenfassung

Im Rahmen der vorliegenden Diplomarbeit wurde gezeigt, daß es dem um vektorielle Ausgabewerte erweiterten *Neural Gas*-Modell möglich ist, den Zusammenhang zwischen der visuellen Repräsentation des Effektors eines Industrieroboters in Stereobildkoordinaten und den vom Arm eingenommenen Gelenkwinkelstellungen (die *hand eye coordination*) ohne jegliches Vorwissen über Kamerapositionen und Roboterarchitektur zu erlernen; die Aufgabe besteht also im Auffinden einer Transformation, welche zunächst Stereobild- in Weltkoordinaten und anschließend Welt- in Gelenkwinkelkoordinaten (**inverse Kinematik**) umwandelt. Durch den adaptiven Charakter des neuralen Netzwerks und die schrittweise Positionierung des Effektors mittels Korrekturbewegungen können kleinere Änderungen in der Kamera- oder Roboterhardware selbsttätig ausgeglichen und nachkalibriert werden.

Kapitel 2 führte zunächst die im Zusammenhang mit der erforderlichen Diskretisierung und topologischen „Erschließung“ des Arbeitsbereichs wichtigen Begriffe der **nachbarschafts**- und **topologieerhaltenden Abbildungen** ein. Neben dem erweiterten *Neural Gas*-Algorithmus selbst wurden auch einige verwandte Verfahren - insbesondere der *Kohonen*-Algorithmus und TRN - ausführlich diskutiert. Es wurde weiters festgestellt, daß das erweiterten *Neural Gas*-Verfahren die topologische Struktur der Eingabesignalmenge S lediglich implizit berücksichtigt (also keine Kanten zwischen benachbarten *units* aufbaut), dies jedoch i.a. ausreicht, um eine gewisse Stetigkeit der vom Netzwerk erlernten Abbildung zu gewährleisten.

Die konkreten Details der Hardwareimplementierung einer auf dem *erweiterten Neural Gas*-Modell basierenden adaptiven Robotersteuerung wurden in **Kapitel 3** besprochen. Wir sahen, daß es, um die inverse Kinematik eines realen Systems erlernen zu können, notwendig ist, das Netzwerk zunächst auf einer softwaremäßig emulierten Umgebung *offline* vorzutrainieren, bevor diesem die Kontrolle über die Roboterhardware übergeben wird. Die im Zusammenhang mit Bildverarbeitung und Effektorpositionierung vorhandenen Fehlerquellen wurden identifiziert und die implementierten Fehlerbehandlungsmechanismen vorgestellt.

In **Kapitel 4** wurde anschließend an die Diskussion der Netzwerkparameter und der verschiedenen Leistungs- bzw. Fehlermaßzahlen das Lernverhalten des erweiterten *Neural Gas*-Modells zunächst anhand von vier Softwareexperimenten diskutiert. Die Ergebnisse geben unter anderem Anlaß zu folgenden Aussagen:

- Eine Verkleinerung des Arbeitsbereichs bei unveränderter Netzwerkarchitektur führt nicht notwendigerweise zu schnellerer Konvergenz bzw. größerer Positioniergenauigkeit; die Ursache ist darin zu sehen, daß das Netzwerk bei einer zu feinen Diskretisierung des Arbeitsbereichs dazu tendiert, die zugrundeliegende Ein-Ausgaberelation auswendig zu lernen.

- Da das erweiterte *Neural Gas*-Verfahren die Topologie des Eingabesignalraums nicht explizit berücksichtigt und in jedem Lernschritt alle Ausgabegewichte modifiziert werden, kann dies bei unzusammenhängenden Verteilungen zu Unstetigkeiten in den Ausgabewerten führen, d.h. es bilden sich ein oder mehrere Bereiche mit persistenten, hohen Ausgabefehlern, welche auch nach einer großen Anzahl von Trainingsschritten bestehen bleiben.

- Das erweiterte *Neural Gas*-Verfahren ist sensitiv bezüglich des Verlaufs der Nachbarschaftsreichweite $\lambda(t)$; werden die Funktionsparameter nicht sorgfältig gewählt, so tritt nach anfänglicher Konvergenz des Verfahrens ein „Inversionseffekt“ (Verschlechterung der Positioniergenauigkeit mit jeder weiteren Musterpräsentation) ein.

- Die in einer Hardwareumgebung erzielbare Positioniergenauigkeit wird im wesentlichen nur durch die Auflösung der Kameras nach unten beschränkt.

Im letzten Experiment wurde ein erweitertes *Neural Gas*-Netzwerk auf einer realen Hardwareumgebung trainiert; für den gewählten Versuchsaufbau wurde bereits nach nur 1000 Lernschritten eine Positioniergenauigkeit von weniger als 0,3 mm nach Durchführung der letzten Korrekturbewegung erzielt, was dem aufgrund der verwendeten Kameraauflösung zu erwartenden Minimalwert entspricht. Die Orientierung beider Kameras wurde nach 1000 Lernschritten geringfügig verändert, was sich jedoch nicht merklich auf die vom Netzwerk erreichte Positioniergenauigkeit auswirkte.

Zusammenfassend läßt sich feststellen, daß das erweiterte *Neural Gas*-Modell einen leistungsfähigen und äußerst vielversprechenden adaptiven Ansatz zur Lösung des *hand-eye* Problems darstellt; um tatsächlich in industriellen Applikationen eingesetzt werden zu können, müßte das Modell jedoch zumindest um die folgenden Eigenschaften erweitert werden:

- Erweiterung des Ausgabebereichs auf fünf oder sechs Dimensionen
  Typische Industrieroboter verfügen über fünf oder sechs Gelenkachsen; diese zusätzlichen Freiheitsgrade werden benötigt, um den Effektor im Arbeitsbereich beliebig orientieren zu können. Eine entsprechende Erweiterung des *Neural Gas*-Modells könnte z.B. auf der Grundlage hierarchischer Netzwerkmodelle erfolgen.

- Einbinden einer Greifstrategie
  Die meisten Objekte können nicht direkt angefahren werden; vielmehr muß zunächst aufgrund ihrer Form und Orientierung ein Annäherungsvektor berechnet werden, entlang welchem sich der Effektor dann auf das Objekt zubewegt (was wiederum voraussetzt, daß der Effektor die entsprechende Orientierung überhaupt annehmen kann). Auch hier bietet sich der Einsatz hierarchischer Netzwerkmodelle an.

- Berücksichtigung von Constraints
  Es könnte z.B. gefordert werden, daß die Änderung der Gelenkwinkel beim Anfahren eines neuen Punktes minimal sein soll; eine andere sinnvolle Einschränkung bestünde in der Vorgabe, die Gelenkwinkel möglichst in der Ruheposition zu belassen und somit „Extrempositionen“ des Armes - welche zu übermäßiger Belastung und Abnutzung der Hardware führen - zu vermeiden. Constraints lassen sich z.B. durch entsprechende Modifikation der Lernregeln (Wahl der zu minimierenden Kostenfunktion) einbinden.

- Berücksichtigung der Topologie der Eingabesignalmenge
  Wie gezeigt wurde, führen unzusammenhängende Teilmengen oder „Löcher“ im Trainingsset dazu, daß sich einzelne *units* gegenseitig in unstetiger Weise beeinflussen, was zur Bildung von Gebieten mit konstant hohen Ausgabefehler führt. Die explizite Berücksichtigung der Topologie der Eingabesignalmenge z.B. durch TRN könnte hier Abhilfe schaffen.

Literatur

[Craig, 1989] Introduction to Robotics: Mechanics and Control. Addison-Wesley, 2. Auflage.

[Fahlman & Lebiere, 1990] The Cascade-Correlation Learning Architecture. Advances in Neural Information Processing Systems, 2: 524 - 532. Morgan Kaufmann, San Mateo.

[Freeman & Skapura, 1991] Neural Networks: Algorithms, Applications and Programming Techniques. Addison-Wesley.

[Fritzke, 1994] Growing Cell Structures - a Self-Organizing Network for Unsupervised and Supervised Learning. Neural Networks, 7(9): 1441 - 1460.

[Fritzke, 1995] A Growing Neural Gas Network Learns Topologies. Advances in Neural Information Processing Systems, 7: 625 - 632.

[Fritzke, 1997] Technischer Report ICSI TR 93-026. Universität Bochum.

[Hertz & Krogh & Palmer, 1991] Introduction to the Theory of Neural Computation. Addison Wesley.

[Horn, 1986] Robot Vision. McGraw-Hill.

[Jain & Kasturi & Schunck, 1995] Machine Vision. Mc Graw-Hill.

[Kohonen, 1989] Self Organization and Associative Memory. Springer, Berlin Heidelberg, 3. Auflage.

[Linde & Buzo & Gray, 1980] An Algorithm for Vector Quantizer Design. IEEE Transactions on Communication, COM-28: 84-95.

[Gray, 1984] Vector Quantization. IEEE ASSP Magazine, 1: 4-29.

[Kirkpatrick & Vecchi, 1983] Optimization by Simulated Annealing. Science, 220.

[Melzer, 1995] Xerion Neural Gas Simulator. Praktikum I. PRIP, TU Wien.

[Melzer, 1996] TROL - TU Vienna Robot Control Library. Praktikum II. PRIP, TU Wien.

[Martinetz, 1992] Selbstorganisierende neuronale Netzwerkmodelle zur Bewegungssteuerung. Infix, Sankt Augustin.

[Martinetz & Ritter & Schulten, 1991] Neuronale Netze: eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke. Addison-Wesley, Bonn. 2. Auflage.

[Martinetz & Schulten, 1991] A Neural Gas Network Learns Topologies. Artificial Neural Networks, 397-402. North-Holland, Amsterdam.

[Martinetz & Schulten, 1994] Topology Representing Networks. Neural Networks, 7(3):507-522.

[Press & Teukolsky & Vetterling & Flannery, 1992] Numerical Recipes in C. Cambridge University Press. 2. Auflage.

[Rumelhart & Hinton & Williams, 1986] Learning Internal Representations by Error Backpropagation. In Parallel Distributed Processing 1(8). MIT Press, Cambridge.

[Tsai, 1987] A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses. IEEE Journal of Robotics and Automation, Vol. RA-3, No. 4: 323-344.

[Werbos, 1974] Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph. D. Thesis, Harvard University.