Bestimmung der Gesichtspose mit künstlichen neuronalen Netzen

Diplomarbeit
am Lehrstuhl für Kognitive Systeme
des Instituts für Informatik und Praktische Mathematik
der Christian–Albrechts–Universität zu Kiel
Prof. Dr. G. Sommer

Betreuer:
Dipl.-Inform. Jörg Bruske

vorgelegt von
Erika Ábrahám-Mumm
Kiel, im April 1998

Zusammenfassung

Das Ziel dieser Arbeit ist die Bestimmung der Gesichtsorientierung und die Identifikation verschiedener Gesichter in Bildern mittels künstlicher neuronaler Netze.
Die Gesichter werden durch Farbsegmentierung lokalisiert und auf eine vorgegebene Größe skaliert. Aus den skalierten Bildern werden durch Faltung mit Gaborfiltern an einigen rasterförmig angeordneten Aufpunkten Merkmalsvektoren extrahiert, die die Eingabevektoren der neuronalen Netze bilden. Die neuronalen Netze lernen, die Gesichtsorientierungen mit den Merkmalsvektoren zu assoziieren. Bei mehreren Objekten erfolgt zusätzlich eine Identifikation.
Zu einigen Teilaufgaben der Gesichtslokalisierung und Merkmalsextraktion werden verschiedene Lösungswege dargestellt. Die unterschiedliche Methoden werden in Versuchen miteinander verglichen, um ein optimiertes Verfahren zur Merkmalsextraktion zu bestimmen.
Es werden unterschiedliche dynamische Netzstrukturen vorgestellt. Durch Experimente wird die Genauigkeit der Posenbestimmung durch die unterschiedlichen Netztypen angegeben. Ein Vergleich der Testergebnisse ermöglicht die Bestimmung optimierter Netzstrukturen.

Inhaltsverzeichnis

Kapitel 1

Einführung

Konrad Zuse konstruierte 1938 die erste, noch mechanische Binärziffernrechenmaschine Z1. Heute, 60 Jahre später, ist der Computer im alltäglichen Leben unerläßlich und wird in den verschiedensten Bereichen angewendet.
Die Entwicklung von modernen Computersystemen hat in vielen Wissenschaftsgebieten neue Möglichkeiten eröffnet. Im Bereich der Informatik ist Künstliche Intelligenz ein aktuelles Forschungsgebiet. Das Ziel der Forschung ist die Entwicklung von Systemen, die Informationen aus der Umgebung wahrnehmen und darauf reagieren können. Für die Kommunikation zwischen Mensch und Maschine wäre es dabei sehr hilfreich, wenn Position und Orientierung eines menschlichen Gesichtes aus einem Kamerabild ermittelt werden könnten. Diese Aufgabe ist das Thema der vorliegenden Arbeit.

Die Posenbestimmung eines Gesichtes in einem Bild besteht aus zwei separaten Aufgaben: Der Lokalisierung des Gesichtes im Bild und der Bestimmung der Gesichtsorientierung. Beide Aufgaben können mittels unterschiedlicher Verfahren gelöst werden.
Einige Möglichkeiten zur Gesichtslokalisierung sind Farbsegmentierung, Konturanalyse oder Bewegungsverfolgung. Diese können auch miteinander kombiniert werden.
In dieser Arbeit verwenden wir die Farbsegmentierung, die auf der relativ konstanten Farbe der Gesichtshaut basiert. Die Gesichtsfarbe ist durch Farbschwellenwerte $\mathbf{s_{min}}$ und $\mathbf{s_{max}}$ definierbar, so daß die Pixelwerte im Gesichtsbereich zwischen den beiden Schwellenwerten liegen. Dadurch wird die Position des Gesichtes bestimmt.
Durch Konturanalyse können die Konturen von Objekten ermittelt werden (z.B. durch Faltung der Bilder mit Sobelfiltern). Anschließend werden die Konturen analysiert, um die gesuchten Objekte zu lokalisieren. Die Lokalisierung des Gesichtes kann durch Lokalisierung einzelner Gesichtsmerkmale wie z.B. Augen, Mund und Nase erfolgen.
Die oberen beiden Methoden können durch Bewegungsverfolgung ergänzt werden. Nachdem das Gesicht einmal lokalisiert worden ist, können die Gesichtskonturen in einer Bildsequenz verfolgt werden. Um die Gesichtsverfolgung zu ermöglichen, müssen die Bilder in der Sequenz in relativ kleinen Zeitabständen aufgenommen worden sein.
Kubisch und Ritter benutzen neuronale Netze für die Segmentierung [KR96]. Das überwachte Lernen der neuronalen Netze setzt eine Trainingsmenge voraus, die in diesem Fall manuell

erstellt werden muß, d.h. die Gesichter in den Trainingsbildern müssen manuell lokalisiert werden. Für größere Trainingsmengen ist die manuelle Erzeugung der Daten sehr aufwendig. Ferner ist die manuelle Lokalisierung ungenau, was zu relativ großen Fehlern in der Orientierungsbestimmung führt.

Nach der Lokalisierung des Gesichtes wird dessen Orientierung bestimmt. Auch zur Orientierungsbestimmung stehen unterschiedliche Methoden zur Verfügung.
Ein geometrisches Modell besteht aus der Lokalisierung von Gesichtsmerkmalen wie Augen, Mund und Nase und der anschliessenden Berechnung der Gesichtsnormalen aufgrund von konstanten Abständen zwischen den Merkmalen. Aus der Gesichtsnormale kann die Orientierung bestimmt werden ([GC94],[BS95]).
Murase und Nayar bestimmen Eigenräume der verwendeten Objekte [MN95]. Durch Projektion der Objektbilder auf den Eigenraum wird die Objektorientierung bestimmt. Die Gesichtslokalisierung geschieht mit klassischen Segmentierungsmethoden.
Wir werden zur Posenbestimmung neuronale Netze anwenden.

In dieser Arbeit dienen Puppen als Versuchsobjekte, die mit einem Roboterarm gedreht werden, um Bilder mit verschiedenen Gesichtsorientierungen aufnehmen zu können. Die Kopforientierung variiert in 2 Richtungen (horizontales Drehen und vertikales Kippen).
Im zweiten Kapitel beschäftigen wir uns mit der Bildverarbeitung. Für einige Teilaufgaben werden verschiedene Lösungswege vorgestellt, deren Ergebnisse in Kapitel 4 durch Experimente verglichen werden.
Die Gesichtslokalisierung in den Bildern geschieht mittels Farbsegmentierung. Ein Nachteil dieser Methode ist die Empfindlichkeit gegen Änderungen der Lichtverhältnisse. Deshalb halten wir die Lichtverhältnisse bei den Bildaufnahmen konstant.
In der vorliegenden Arbeit wird die Gesichtsorientierung mittels neuronaler Netze bestimmt. Die Eingabevektoren der neuronalen Netze sind Merkmalsvektoren des Gesichtes. Um aus den Bildern Merkmalsvektoren extrahieren zu können, wird das durch Farbsegmentierung lokalisierte Gesicht auf eine bestimmte Größe skaliert. Bei den Bildaufnahmen ist die Entfernung des Kopfes von der Kamera relativ konstant. Kleine Änderungen werden durch die Skalierung ausgeglichen.
Das skalierte Farbbild wird in ein Grauwertbild umgewandelt und an einigen Aufpunkten mit Gaborfiltern verschiedener Orientierungen und evtl. verschiedener Durchlaßbereiche gefaltet. Die Faltung kann auch direkt mit den Farbbildern durchgeführt werden. Die Antworten werden zu einem Merkmalsvektor zusammengefaßt, der als Eingabe der neuronalen Netze dient.
Im dritten Kapitel werden die angewendeten neuronalen Netze beschrieben. Die Netze lernen, bei Eingabe der Merkmalsvektoren die Orientierungen der Gesichter als Ausgabe berechnen. Beim Training mit mehreren Puppen erfolgt zusätzlich eine Identifikation.
Wir werden mehrere verschiedene Netztypen anwenden, die sich sowohl in der Struktur als auch in der Lernmethode unterscheiden. Im Mittelpunkt unserer Versuche stehen die dynamischen Netzstrukturen. Es werden Netze sowohl für das *offline*-Training mit festen Trainingsmengen als auch solche für das *online*-Training realisiert.
Abbildung 1.1 zeigt den Ablauf der Orientierungsbestimmung.

Im vierten Kapitel wenden wir uns der Durchführung der Versuche zu. Nach der Beschreibung des Versuchsaufbaus und der technischen Realisierung werden Versuche mit den verschiedenen, in den Kapiteln 2 und 3 vorgestellten Bildverarbeitungsmethoden und neuronalen Netzen durchgeführt und miteinander verglichen. Anschließend werden verschiedene Anwendungsmöglichkeiten dargestellt.
Im fünften Kapitel werden die Versuchsergebnisse ausgewertet und mit den Ergebnissen anderer Arbeiten verglichen. Anschließend werden Weiterentwicklungsmöglichkeiten diskutiert.

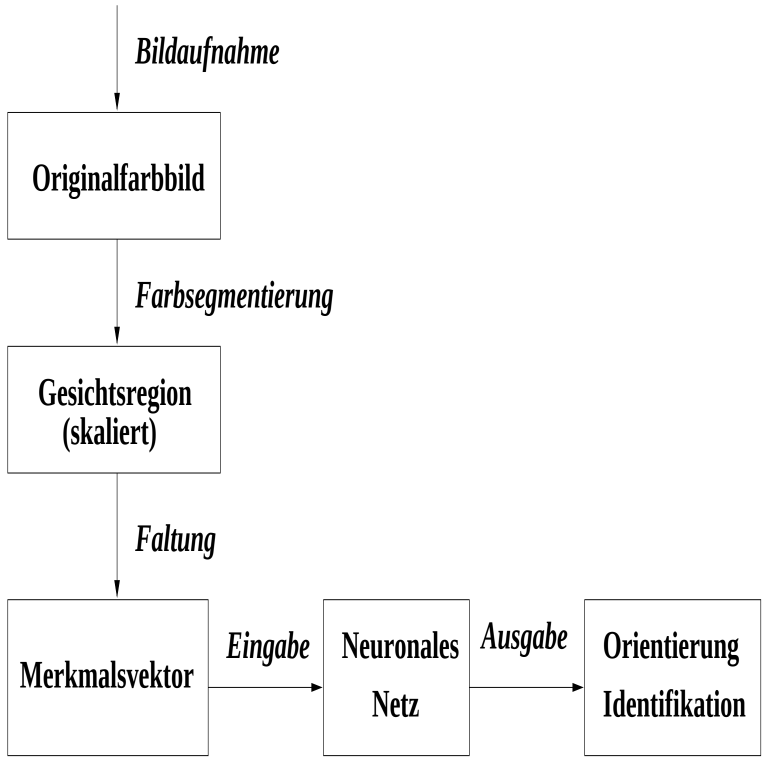


Abbildung 1.1: Die einzelnen Schritte der Orientierungsbestimmung.

Kapitel 2

Bildverarbeitung

Das Ziel dieser Arbeit ist die Bestimmung der Orientierung eines Gesichtes in einem Bild. Die Gesichtsorientierung kann auf verschiedenen Wegen berechnet werden. Hier werden künstliche neuronale Netze benutzt, die in einem überwachten Lernverfahren lernen, die Orientierung des Gesichtes zu berechnen (s. Kapitel 3). Überwachtes Lernen bedeutet, daß den neuronalen Netzen Trainingsdaten zur Verfügung gestellt werden, die für verschiedene Eingabevektoren die dazugehörigen erwünschten Ausgaben des Netzes angeben.
Dieses Kapitel beschäftigt sich mit der Erzeugung der Trainingsdaten aus vorhandenen Bildern.
Im Abschnitt 2.1 wird der Vorgang der Bildaufnahme beschrieben. Für eine möglichst genaue Bestimmung der Gesichtsorientierung durch die neuronalen Netze ist eine gute Bildqualität sehr wichtig.
Um Trainingsdaten zu erzeugen, müssen aus den Bildern kodierte Informationen über die Gesichtspose (sogenannte Merkmalsvektoren) extrahiert werden. Dieses Verfahren wird Merkmalsextraktion genannt.
Die Berechnung des Merkmalsvektors eines aufgenommenen Bildes geschieht in zwei Schritten: Im ersten Schritt wird das Gesicht im Bild mittels Farbsegmentierung lokalisiert und auf eine vorgegebene Größe skaliert. Die verwendeten Methoden werden im Abschnitt 2.2 vorgestellt. Die eigentliche Merkmalsextraktion wird dann im zweiten Schritt mittels Faltung durchgeführt. Bei der Faltung werden Gaborfilterpaare verwendet. Die Merkmalsextraktion ist das Thema des Abschnittes 2.3.
Bei den einzelnen Schritten der Bildverarbeitung werden verschiedene Methoden dargestellt, die später in Versuchen miteinander verglichen werden. Auf eine Einführung in die Grundlagen der Bildverarbeitung (wie z.B. die Definition der Faltung oder Filtermasken) wird hier verzichtet und auf Fachliteratur verwiesen (siehe z.B. [Jäh93]).

2.1 Bildaufnahme

Als Versuchsobjekt dient ein Puppenkopf, der mit einem Roboterarm in 2 Richtungen gedreht werden kann: In eine horizontale Dreh- und eine vertikale Kipprichtung. Die Ori-

entierung des Kopfes wird durch den Dreh- und Kippwinkel beschrieben.
Die Bildaufnahme geschieht mit Hilfe einer Farbkamera. Die Bilder sind 8-bit RGB-Farbbilder, d.h. die Farben der Bildpixel werden in 3 Farbbändern (Farbkomponente rot, grün und blau) dargestellt, wobei die Pixel in jedem Band Werte zwischen 0 und 255 annehmen können.
Um den Rechenaufwand zu verringern, werden die Kamerabilder mit dem Skalierungsfaktor 1/2 skaliert (d.h. ein $2N \times 2M$ Pixel großes Bild wird auf die Größe von $N \times M$ Pixel skaliert). Das Gesicht wird immer noch ausreichend detailliert dargestellt und der Rechenaufwand verringert sich auf ein Viertel.
Nach der Aufnahme und der Skalierung liegen die Farbbilder in Matrizen $\mathbf{g}$ der Größe $N \times M$ mit Werten $\mathbf{g}(x,y) = (g_0(x,y), g_1(x,y), g_2(x,y)) \in [0,255] \times [0,255] \times [0,255]$ vor.
Bei der Bildaufnahme ist wichtig, daß die Lichtverhältnisse möglichst konstant bleiben. In Kapitel 2.3.3 wird gezeigt, daß durch Logarithmieren der Pixelwerte die Merkmalsextraktion helligkeitsinvariant wird. Da aber die der Merkmalsextraktion vorangehende Farbsegmentierung helligkeitsabhängig ist, hilft dieses nur bei kleinen Helligkeitsschwankungen (siehe Abschnitt 2.2.1).
Bei unseren Versuchen bleibt die Entfernung des Gesichtes von der Kamera ungefähr konstant. Nach der Gesichtslokalisierung wird das Gesicht auf eine vorgegebene Größe skaliert. Dadurch werden kleine Änderungen der Gesicht-Kamera-Entfernung ausgeglichen.

2.2 Gesichtslokalisierung

Für die spätere Merkmalsextraktion muß das Gesicht im Bild mittels Farbsegmentierung lokalisiert und auf einen quadratischen Ausschnitt vorgegebener Größe skaliert werden.

2.2.1 Farbsegmentierung

Die Lokalisierung eines Gesichtes in einem Bild ist eine komplexe Aufgabe. Mit Hilfe eines Segmentierungsverfahrens werden die Bildpixel in zwei disjunkte Pixelmengen (Gesichts- und Hintergrundsregion) aufgeteilt.
Es gibt verschiedene bekannte Segmentierungsmethoden. Mittels Konturanalyse sind z.B. Gesichtsteile wie Augen, Mund und Nase lokalisierbar. Mit Hilfe dieser Informationen erfolgt die Segmentierung des Gesichtes. Für die Identifizierung der Gesichtsteile und eine eventuelle Orientierungsbestimmung des Gesichtes werden die konstanten Abstände zwischen den Gesichtsmerkmalen ausgenutzt.
Eine andere Methode beruht auf Bewegungsverfolgung. Das Gesicht wird einmal manuell lokalisiert und dann seine Bewegungen verfolgt. Hierfür ist aber eine kontinuierliche Folge von Bildern nötig, die uns in unseren Anwendungen nicht zur Verfügung steht.
In dieser Arbeit wird die Farbsegmentierung verwendet. Diese basiert auf der relativ konstanten Farbe der Gesichtshaut. Die Gesichtsfarbe wird durch Schwellenwerte $\mathbf{s}^{\mathbf{min}}$ und $\mathbf{s}^{\mathbf{max}}$ definiert, zwischen denen sich die Farbe der Gesichtshaut bewegt. Da die Pixelwerte $\mathbf{g}(x,y) = (g_0(x,y), g_1(x,y), g_2(x,y))$ in den RGB-Farbbildern dreidimensional sind, sind

die Schwellenwerte $\mathbf{s^{min}} = (s_0^{min}, s_1^{min}, s_2^{min})$ und $\mathbf{s^{max}} = (s_0^{max}, s_1^{max}, s_2^{max})$ ebenfalls dreidimensional.
Die Schwellenwerte der Gesichtsfarbe sind dabei helligkeitsabhängig. Wenn sich die Helligkeit ändert, ändern sich auch die optimierten Schwellenwerte. Da aber in unseren Versuchen die Bildaufnahmen unter konstanten Lichtverhältnissen stattfinden, können wir von konstanten Schwellenwerten ausgehen.
Die Gesichtslokalisierung geschieht nun in zwei Schritten. Im ersten Schritt werden in dem $N \times M$ Pixel großen Bild $\mathbf{g}$ alle Pixel mit Farbwerten innerhalb des durch die Schwellenwerte definierten Bereiches dem Vordergrund (V) und alle anderen dem Hintergrund (H) zugeordnet:

$$\begin{aligned} V &= \{(x,y) \in \{1,\cdots,N\} \times \{1,\cdots,M\} | \forall 0 \leq i \leq 2 : s_i^{min} \leq g_i(x,y) \leq s_i^{max}\} \\ H &= \{(x,y) \in \{1,\cdots,N\} \times \{1,\cdots,M\} | \exists 0 \leq i \leq 2 : g_i(x,y) < s_i^{min} \vee s_i^{max} < g_i(x,y)\}\,. \end{aligned}$$

Um andere gesichtsfarbene Objekte (wie z.B. eine Hand) vom Gesicht trennen zu können, wird in einem zweiten Schritt in der Vordergrundmenge die größte in einer 8er Nachbarschaft zusammenhängende Teilmenge gesucht und als Gesichtsbereich G definiert.

Histogramme

Bei der Bestimmung optimierter Schwellenwerte sind Histogramme wertvolle Hilfsmittel. Das Histogramm eines zweidimensionalen $N \times M$ Grauwertbildes $\mathbf{g}$ mit positiven, ganzzahligen Pixelwerten $g(x,y) \in [0,\cdots,k-1]$ ist ein k-dimensionaler Vektor $\mathbf{h} = (h_0,\cdots,h_{k-1})$, wobei jede Komponente h_i, $i = 0,\cdots,k-1$ die Anzahl der Pixel im Bild mit dem Grauwert i angibt:

$$\forall 0 \leq i \leq k-1 : h_i = |\{(x,y) \in \{1,\cdots,N\} \times \{1,\cdots,M\}\, | g(x,y) = i\}|\,. \tag{2.1}$$

Für Farbbilder mit l Farbbändern (d.h. mit l-dimensionalen Farbwerten) wird entweder eine l-dimensionale Histogramm-Matrix berechnet, oder es wird für jedes Farbband ein separates Histogramm erstellt. Wegen ihrer Übersichtlichkeit wird in dieser Arbeit die zweite Darstellung verwendet.
Im Idealfall hat kein anderes Objekt im Bild gesichtsähnliche Farben. Dann weist das Histogramm im Gesichtsfarbenbereich ein deutliches und klar getrenntes Maximum auf. Wenn dieses Maximum nicht deutlich oder von anderen Maxima nicht klar getrennt ist, muß für die Schwellenwertbestimmung eine andere Methode verwendet werden.
Die Schwellenwertbestimmung wird besonders schwer, wenn das Gesicht ohne Haare lokalisiert werden soll, da die Haarfarbe von der Gesichtsfarbe oft nur schwer zu unterscheiden ist.
Trotzdem ist es möglich, mittels Histogrammen Farbbereiche zu finden, die die Gesichtsfarbe mit ausreichender Genauigkeit beschreiben.
Als Beispiel sind in Abbildung 2.1 ein Bild und die Histogramme der drei Farbkomponenten abgebildet.

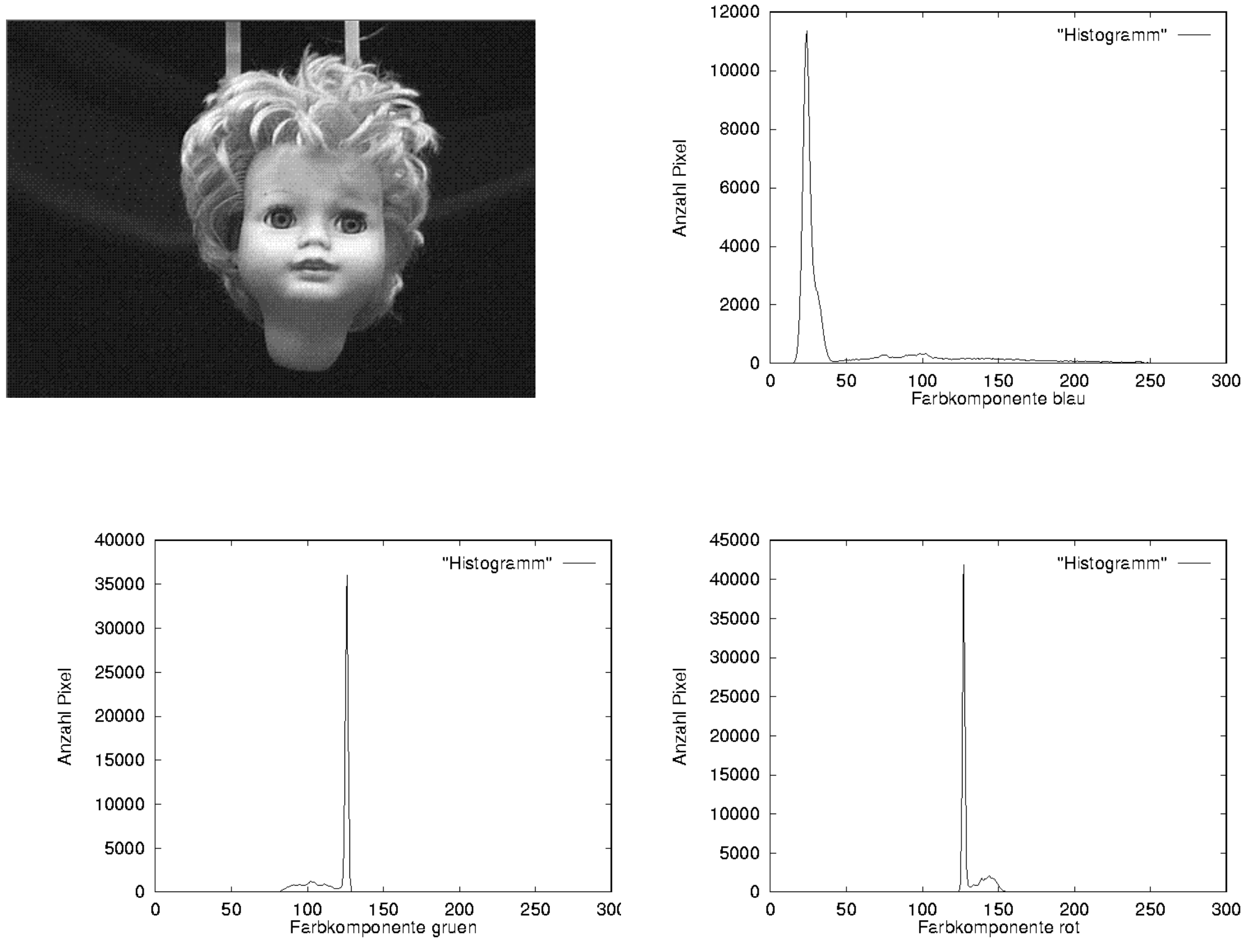


Abbildung 2.1: Ein Bild von einer Puppe und die zugehörigen Histogramme der drei Farbkomponenten blau (oben rechts), grün (unten links) und rot (unten rechts).

2.2.2 Skalierung

Für die Merkmalsextraktion soll das Gesicht in Form eines quadratischen Ausschnittes von $K \times K$ Pixeln (K gerade) vorliegen.
Der Schwerpunkt (x_s, y_s) der Gesichtsregion G ist gegeben durch

$$(x_s, y_s) = \frac{\sum_{(x,y)\in G} (x, y)}{|G|}. \tag{2.2}$$

Um das Gesicht auf eine bestimmte Größe skalieren zu können, bestimmen wir den Radius des Gesichtes:

$$r = \max_{(x,y)\in G} \{||(x, y) - (x_s, y_s)||_2\}. \tag{2.3}$$

Ist der gewünschte Gesichtsradius $r_s = K/2$, so soll das Gesicht mit dem Faktor

$$f = \frac{r_s}{r} \tag{2.4}$$

skaliert werden. Die Skalierung geschieht mittels bilinearer Interpolation.
Auf die Berechnung des Schwerpunktes kann auch verzichtet werden. Statt der schwerpunktbezogenen Lokalisierung wird dann eine exakte Lokalisierung durchgeführt. In diesem Fall werden die obersten, untersten, linkesten und rechtesten Gesichtspunkte (x_o, y_o), (x_u, y_u), (x_l, y_l), (x_r, y_r) bestimmt. Die Gesichtsbreite ist dann

$$dx = x_r - x_l \tag{2.5}$$

und die Gesichtshöhe

$$dy = y_u - y_o. \tag{2.6}$$

Der Gesichtsausschnitt wird auf ein Quadrat ergänzt, in dem zu der schmaleren Seite von beiden Richtungen entsprechend viele Zeilen bzw. Spalten hinzugefügt werden. Der so gewonnene quadratische Ausschnitt hat die Seitenlänge $max\{dx, dy\}$.
Der Faktor f der Skalierung ist in diesem Fall

$$f = \frac{K}{max\{dx, dy\}}. \tag{2.7}$$

Um die beiden Methoden vergleichen zu können, wird in den Versuchen sowohl schwerpunktbezogene als auch exakte Lokalisierung angewendet.

2.2.3 Der Hintergrund

Es stellt sich noch die Frage, wie in den Gesichtsausschnitten die Hintergrundpixel behandelt werden sollen. Im folgenden werden einige Verfahren beschrieben, die in Kapitel 4 auf ihre Wirkungen auf die Orientierungsbestimmung getestet werden.
Der natürlichste Weg ist, die Hintergrundpixel ihre Originalfarbwerte behalten zu lassen. Wenn sich der Hintergrund während der Aufnahmen nicht ändert, ändern sich die Faltungsantworten an den Hintergrundpixeln ebenfalls nicht.
Wenn sich der Hintergrund während der Aufnahmen ändert, wird die Faltung verschiedener Bilder mit der gleichen Gesichtspose, aber mit unterschiedlichem Hintergrund unterschiedliche Antworten liefern. In diesem Fall muß der Hintergrund vorverarbeitet werden.
Die Hintergrundpixel können z.B. mit einem konstanten Farbwert versehen werden. Da es aber Gesichtsteile gibt, wie z.B. die Augen, die keine Gesichtsfarbe haben, also zum Hintergrund zählen, trotzdem aber zum Gesicht gehören sollen, muß der Bereich innerhalb des Gesichtes mit den Originalfarbwerten aufgefüllt werden. Dies geschieht zeilenweise. Allen Pixeln zwischen den linkesten und den rechtesten Gesichtspixeln einer Zeile werden die Originalfarbwerte zugeordnet und allen anderen der konstante Hintergrundwert.

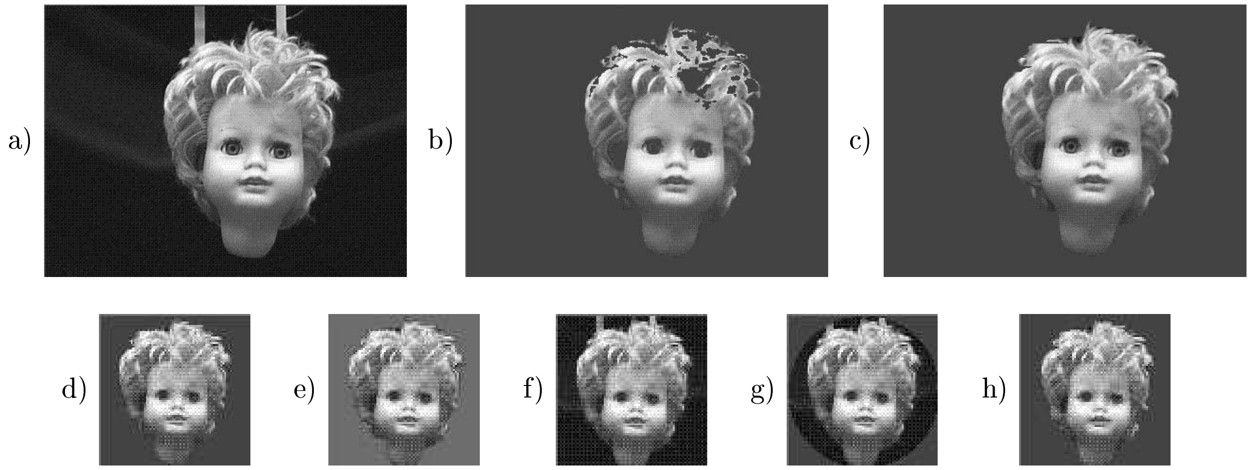


Abbildung 2.2: Ein Beispiel zur Gesichtslokalisierung. a) Originalbild b) Die gesichtsfarbenen Pixel im Bild c) Das Innere des größten zusammenhängenden gesichtsfarbenen Bereiches wird mit den Originalfarbwerten aufgefüllt. d)-g) Der skalierte Gesichtsausschnitt mit exakter Lokalisierung und d) mit konstantem Hintergrund e) mit durchschnittlichem Hintergrund f) mit Originalhintergrund g) im Kreisausschnitt. h) Der skalierte Gesichtsausschnitt mit schwerpunktbezogener Lokalisierung und mit konstantem Hintergrund.

Eine weniger rechenintensive Methode wäre es, die Gesichtsregion auf einen Kreis mit dem Schwerpunkt als Mittelpunkt und mit dem Gesichtsradius als Radius zu erweitern. Den Pixeln innerhalb des Kreises werden die Originalfarbwerte zugewiesen und den restlichen Pixeln im Bild der konstante Hintergrundwert.
Wenn die Hintergrundpixel eine konstante Farbe haben und sich die Lichtverhältnisse ändern, ändern sich die Pixelwerte in der Gesichtsregion, nicht aber im Hintergrund. Die Faltung wird also mit teilweise helligkeitsunabhängigen und teilweise helligkeitsabhängigen Pixelwerten durchgeführt. Dies führt zu fehlerhaften Netzantworten. Deshalb ist es bei Helligkeitsschwankungen sinnvoll, die Werte der Hintergrundpixel auf den Durchschnittsfarbwert des Hintergrundes zu stellen. Wenn sich aber der Hintergrund selbst (also die Objekte im Hintergrund) ändert, führt dies zu unterschiedlichen, szenenabhängigen Hintergrundwerten.

Das Beispiel in Abbildung 2.2 veranschaulicht die einzelnen Schritte der Gesichtslokalisierung. Das erste Bild a) zeigt ein Puppengesicht nach der Aufnahme. Der Hintergrund ist einfach gestaltet. Das zweite Bild b) zeigt die gesichtsfarbenen Bereiche. Alle anderen Pixel werden mit einer konstanten Farbe dargestellt. Im Bild c) wird der größte zusammengenhängende gesichtsfarbene Bereich bestimmt und dessen Inneres mit den Originalfarbwerten aufgefüllt. Die nächsten Bilder zeigen die skalierten Gesichtsausschnitte, die mit der Methode der exakten Lokalisierung mit den vier verschiedenen Verfahren zur Hintergrundsbehandlung (d)-(g) und mit der Methode der Schwerpunktsuche mit konstantem Hintergrund (h) erzeugt wurden.
Es muß noch einmal betont werden, daß die Genauigkeit der Gesichtslokalisierung und die

Verfahren, mit denen das Gesicht an das vorgegebene Quadrat angepaßt wird, sehr starke Auswirkungen auf die Genauigkeit der Bestimmung der Gesichtsorientierung haben. Es ist sehr wichtig, daß sich bei einer kontinuierlichen Drehung des Gesichtes das skalierte Gesichtsbild auch kontinuierlich bewegt. Ferner soll das Verfahren, angewendet auf verschiedene Aufnahmen mit der gleichen Gesichtsorientierung, das gleiche Ergebnis liefern, d.h. die Gesichter sollen sich in den quadratischen skalierten Ausschnitten an der gleichen Position befinden.

2.3 Merkmalsextraktion

Aus den skalierten Bildern der aufgenommenen Gesichter sollen Merkmalsvektoren extrahiert werden, die als Eingabevektoren der neuronalen Netze dienen.
Theoretisch kann der ganze skalierte Gesichtsausschnitt als Netzeingabe fungieren. Problematisch ist hierbei, daß die Lokalisierung eines Gesichtes mit der gleichen Orientierung in zwei verschiedenen Bildern nicht immer exakt das gleiche Ergebnis liefert, d.h. die Lage des Gesichtes im Bild ist nicht exakt dieselbe. Eine Verschiebung von einem Pixel kann schon zu einer fehlerhaften Netzausgabe führen.
Um dies zu vermeiden, werden die für die Anwendung wichtigsten Informationen (Merkmale) aus dem Bild extrahiert und zu einem Merkmalsvektor zusammengefaßt. Dies entspricht einer Projektion der Bilddaten auf einen niederdimensionalen Unterraum. Die Merkmalsvektoren müssen dabei genügend Informationen zur Identifizierung und Orientierungsbestimmung des Gesichtes enthalten.
Murase und Nayar [MN95] haben ein Verfahren zur Merkmalsextraktion entwickelt, das auf der Methode der Hauptachsenanalyse basiert. Für die Orientierungsbestimmung eines Objektes gehen sie von einer Menge von k Bildern $\mathbf{g^1}, \cdots \mathbf{g^k}$ des betreffenden Objektes mit verschiedenen Orientierungen aus. Sie berechnen den Durchschnitt aller Bilder $\mathbf{c}$ durch

$$\mathbf{c} = \frac{1}{k} \sum_{i=1}^{k} \mathbf{g^i} \tag{2.8}$$

und subtrahieren diesen Durchschnitt aus jedem Bild. Dadurch ergibt sich eine neue Bildmenge $\mathbf{X}$, definiert durch

$$\mathbf{X} = \{\mathbf{g^1} - \mathbf{c}, \cdots, \mathbf{g^k} - \mathbf{c}\}. \tag{2.9}$$

Aus den so vorverarbeiteten Bildern wird die Kovarianzmatrix

$$\mathbf{Q} = \mathbf{X}\mathbf{X^T} \tag{2.10}$$

gebildet. Die l größten Eigenwerte $\lambda_1, \cdots \lambda_l$ und die dazugehörigen Eigenvektoren $\mathbf{e_1}, \cdots \mathbf{e_l}$ der Kovarianzmatrix mit

$$\forall 1 \le i \le l \ (\lambda_i \mathbf{e_i} = \mathbf{Q}\mathbf{e_i}) \tag{2.11}$$

werden durch Hauptkomponentenanalyse bestimmt. Diese Eigenvektoren spannen den Eigenraum des Objektes auf. Durch Projektion jedes Bildes $\mathbf{g^i}$, $i = 1, \cdots, k$ auf den Eigenraum ergeben sich k Vektoren $\mathbf{g^i_{eigen}}$, $i = 1, \cdots, k$ mit

$$\mathbf{g^i_{eigen}} = [\mathbf{e_1}, \cdots, \mathbf{e_l}]^{\mathbf{T}}(\mathbf{g^i} - \mathbf{c}), \tag{2.12}$$

die mit kubischer Spline-Interpolation zu einer Fläche **F** verbunden werden. Jeder Punkt dieser Fläche repräsentiert die Ansicht des Objektes unter einer bestimmten Orientierung. Für die Bestimmung der Objektorientierung wird vom Bild des Objektes wiederum der Durchschnitt aller Bilder abgezogen und das so vorverarbeitetes Bild auf den Eigenraum projiziert. Der dem Projektionspunkt am nächsten liegende Punkt der Fläche **F** gibt die Orientierung des Objektes an.
Dieses Verfahren ist zwar effektiv, setzt aber eine Menge vorgegebener Bildansichten voraus, was ein online-Training der neuronalen Netze (s. Kapitel 3) ausschließt. Aus diesem Grund wird in dieser Arbeit ein anderes Verfahren verwendet. Die Merkmalsextraktion geschieht hier mittels Faltung der Bilder mit Filtermasken. Die Grundidee dieser Methode ist der von Murase und Nayar nicht fremd. Auch hier wird ein Eigenraum der möglichen Gesichtsposen bestimmt, der durch die Merkmalsvektoren der einzelnen Bilder aufgespannt wird. Die Projektion der Bilder auf den Eigenraum wird durch die Bildfaltung repräsentiert. Jedem Punkt des Eigenraumes wird eine bestimmte Gesichtspose zugeordnet.
Jedes Gesicht weist einige typische Merkmale konstanter Größe auf. Es bietet sich daher an, zur Merkmalsextraktion ein Bandpaßfilter anzuwenden. Neben ihrem Beitrag zur Objektlokalisation eignen sich Bandpaßfilter auch zur Rauschunterdrückung.
Da wir an der Orientierung des Gesichtes interessiert sind, sollte der Filter auch orientierungsselektiv sein. Gabor-Quadraturfilterpaare erfüllen diese Forderungen.

2.3.1 Das Gabor-Quadraturfilterpaar

Ein Quadraturpaar ist eine Funktion und deren Hilberttransformierte. Beide Teilfunktionen haben die gleiche Amplitudenfunktion, unterscheiden sich aber in der Phasenlage. Quadraturpaare sind in der Lage, die lokale Energie und die Phasenlage eines Signals zu erfassen.
Die bekanntesten Quadraturfilterpaare sind die Gaborfilter. Sie selektieren einen Wellenzahlbereich mit einer Gaußschen Glockenkurve, die ihren Schwerpunkt bei einer Wellenlänge λ_x hat. Ein zweidimensionales Gaborfilterpaar besteht aus einem geraden $q_+ : \mathcal{R}^2 \rightarrow \mathcal{R}$ und einem ungeraden $q_- : \mathcal{R}^2 \rightarrow \mathcal{R}$ Filterteil. Die Filter lassen sich im Ortsraum beschreiben durch

$$q_+(\mathbf{x}) = \frac{1}{2\pi\sigma_x^2} \cos\left(\mathbf{k^T}(\mathbf{R}_\theta \mathbf{x})\right) \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) \tag{2.13}$$

$$q_-(\mathbf{x}) = \frac{1}{2\pi\sigma_x^2} \sin\left(\mathbf{k^T}(\mathbf{R}_\theta \mathbf{x})\right) \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) \tag{2.14}$$

mit

$$\mathbf{k} = \left(\frac{2\pi}{\lambda_x}, 0\right)^T. \tag{2.15}$$

Die Funktionen haben die folgenden Parameter:

- λ_x ist die Gaborwellenlänge. Sie bestimmt die Wellenlänge der Sinus- und Kosinusfunktionen in x-Richtung und damit das Maximum des Durchlaßbereichs des Filters.
- σ_x gibt die Varianz der Gaußschen Glockenkurve und damit die Breite des Filters an.
- $\mathbf{R}_\theta$ ist eine Rotationsmatrix und bestimmt die Orientierung des Filters:

$$\mathbf{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}. \tag{2.16}$$

Die lokale Energie $q(\mathbf{x}) : \mathcal{R}^\mathbf{2} \to \mathcal{R}$ ist definiert als die Summe der quadrierten Faltungsantworten der beiden Teilfilter:

$$q(\mathbf{x}) = q_+(\mathbf{x})^2 + q_-(\mathbf{x})^2. \tag{2.17}$$

Damit Gaborfilter mit verschiedenen Durchlaßbereichen die gleiche Charakteristik haben, werden in unseren Versuchen die Parameter σ_x und λ_x gekoppelt:

$$\sigma_x = \frac{\lambda_x}{4}. \tag{2.18}$$

In Abbildung 2.3 a)-b) sind ein ein- und ein zweidimensionales Gaborfilterpaar dargestellt.

2.3.2 Gleichanteilsfreiheit

Die lokale Energie soll auf homogene Bildwertbereiche null sein. Dies ist nur möglich, wenn das Filter gleichanteilsfrei ist, d.h. der Integrand des Filterpaars über den ganzen Raum $\mathcal{R}^2$ null ist:

$$\int_{\mathcal{R}^2} q_+(\mathbf{x})\, d\mathbf{x} = \int_{\mathcal{R}^2} q_-(\mathbf{x})\, d\mathbf{x} = 0. \tag{2.19}$$

Der ungerade Anteil q_- ist zum Ursprung punktsymmetrisch und damit gleichanteilsfrei, der gerade Anteil q_+ aber nicht. Um Gleichanteilsfreiheit zu erreichen, muß die Cosinusfunktion im geraden Anteil um ein Skalar d verschoben werden, so daß gilt:

$$\int_{\mathcal{R}^2} \frac{1}{2\pi\sigma_x^2} \left(\cos\left(\mathbf{k}^\mathbf{T}(\mathbf{R}_\theta \mathbf{x})\right) - d\right) \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x} = 0. \tag{2.20}$$

Für den Verschiebungsfaktor d gilt:

$$
\begin{aligned}
& \int_{\mathcal{R}^2} \frac{1}{2\pi\sigma_x^2} \left(\cos\left(\mathbf{k}^\mathbf{T}(\mathbf{R}_\theta \mathbf{x})\right) - d\right) \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x} && = 0 \\
\Leftrightarrow\ & \int_{\mathcal{R}^2} \left(\cos\left(\mathbf{k}^\mathbf{T}(\mathbf{R}_\theta \mathbf{x})\right) - d\right) \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x} && = 0 \\
\Leftrightarrow\ & \int_{\mathcal{R}^2} \cos\left(\mathbf{k}^\mathbf{T}(\mathbf{R}_\theta \mathbf{x})\right) \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) - d \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x} && = 0 \\
\Leftrightarrow\ & \int_{\mathcal{R}^2} \cos\left(\mathbf{k}^\mathbf{T}(\mathbf{R}_\theta \mathbf{x})\right) \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x} - \int_{\mathcal{R}^2} d \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x} && = 0 \\
\Leftrightarrow\ & \int_{\mathcal{R}^2} \cos\left(\mathbf{k}^\mathbf{T}(\mathbf{R}_\theta \mathbf{x})\right) \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x} - d \int_{\mathcal{R}^2} \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x} && = 0 \\
\Leftrightarrow\ & d = \frac{\int_{\mathcal{R}^2} \cos\left(\mathbf{k}^\mathbf{T}(\mathbf{R}_\theta \mathbf{x})\right) \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x}}{\int_{\mathcal{R}^2} \exp\left(-\frac{\mathbf{x}^2}{2\sigma_x^2}\right) d\mathbf{x}}.
\end{aligned}
\tag{2.21}
$$

2.3.3 Helligkeitsinvarianz

Bei zwei verschiedenen Aufnahmen vom gleichen Kopf mit der gleichen Orientierung sollte die Faltung die gleichen Filterantworten liefern. Wenn sich die Helligkeit zwischen den beiden Aufnahmen ändert, kann dies nicht mehr gesichert werden.
Die Helligkeit wirkt multiplikativ auf die Farben und damit auch auf die Faltungsantworten. D.h., wird die Helligkeit verdoppelt, so werden die Farbwerte und die Faltungsantworten auch doppelt so groß. Wenn h die Helligkeit bezeichnet, so gilt für die Farbe eines Pixels (x, y):

$$
\mathbf{g}(x,y) = (g_0(x,y), g_1(x,y), g_2(x,y)) = (\tilde{g}_0(x,y) \cdot h, \tilde{g}_1(x,y) \cdot h, \tilde{g}_2(x,y) \cdot h), \tag{2.22}
$$

wobei $(\tilde{g}_0(x,y), \tilde{g}_1(x,y), \tilde{g}_2(x,y))$ die helligkeitsunabhängige Farbe bezeichnet.
Eine Möglichkeit zur helligkeitsinvarianten Merkmalsextraktion ist die Faltung mit logarithmierten Bilddaten. Um die Auswirkungen von Helligkeitsänderungen auf die Faltungsantworten zu verringern, werden die Farbwerte der einzelnen Pixel im Bild logarithmiert und die Faltung mit dem logarithmierten Bild durchgeführt. Für jeden Bildpixel (x, y) mit $\mathbf{g}(x,y) = (g_0(x,y), g_1(x,y), g_2(x,y))$ und für jedes Farbband $0 \leq i \leq 2$ gilt:

$$
\begin{aligned}
\log(g_i(x,y)) &= \log(\tilde{g}_i(x,y) \cdot h) \\
&= \log(\tilde{g}_i(x,y)) + \log(h),
\end{aligned}
\tag{2.23}
$$

wobei $\tilde{g}_i(x,y)$ wieder den helligkeitsunabhängigen Farbwert bezeichnet.
Nach dem Logarithmieren wirkt die Helligkeit also additiv auf die Pixelfarbwerte. Aufgrund der Gleichanteilsfreiheit des Gaborfilters hat sie damit bei einer helligkeitsinvarianten Farbsegmentierung auf die Faltungsantworten keine Auswirkung.
Leider sind aber die Schwellenwerte der Farbsegmentierung helligkeitsabhängig, was zu

Ungenauigkeiten in der Gesichtslokalisierung führt. (Näheres dazu siehe Kapitel 5.) Aus diesem Grund werden nur Experimente unter konstanten Lichtverhältnissen durchgeführt. Die Faltungen geschehen immer mit den ursprünglichen Pixelfarbwerten.

2.3.4 Die Faltung

Die Filtermasken

Um die diskrete Bildsignale mit einem Gaborfilterpaar falten zu können, werden die kontinuierlichen zweidimensionalen Gaborfunktionen in gleichmäßigen Abständen abgetastet. In den Versuchen werden Filtermasken von der Größe des Bildes (80×80 Pixel) verwendet. Falls aber die Gaußfunktionen im Vergleich zur Bildbreite eine relativ kleine Breite haben, wären auch kleinere Masken anwendbar.
Abbildung 2.3 c)-f) zeigt ein Gaborfilterpaar mit 4 verschiedenen Orientierungen.

Gleichanteilsfreiheit

Im diskreten zweidimensionalen Fall berechnet sich der Verschiebungsfaktor der Cosinusfunktion im geraden Gaborfilterteil entsprechend (2.21):

$$d = \frac{\sum_{x_1=-K/2}^{K/2-1} \sum_{x_2=-K/2}^{K/2-1} \cos\left(\mathbf{k}^{\mathbf{T}} \left(\mathbf{R}_\theta (x_1, x_2)^T\right)\right) \exp\left(-\frac{(x_1,x_2)(x_1,x_2)^T}{2\sigma_x^2}\right)}{\sum_{x_1=-K/2}^{K/2-1} \sum_{x_2=-K/2}^{K/2-1} \exp\left(-\frac{(x_1,x_2)(x_1,x_2)^T}{2\sigma_x^2}\right)} \quad . \tag{2.24}$$

Randbehandlung

Für die Randbehandlung bei der Faltung gibt es zwei gängige Methoden: entweder wird das Bild über dessen Ränder mit Nullen erweitert (was gleichbedeutend ist mit dem Abschneiden von Maskenelementen), oder es wird eine zyklische Faltung durchgeführt, d.h. das Bild wird zyklisch in allen Richtungen erweitert. In den Versuchen wird die zweite Methode gewählt, damit die Gleichanteilsfreiheit des Gaborfilters nicht verletzt wird.

Konvertierung der RGB-Farbbilder in Grauwertbilder

Es ist auch möglich, die Faltungen auf Farbbildern durchzuführen. In diesem Fall müssen alle 3 Farbbänder gefaltet werden. Damit liegen für jeden Aufpunkt 3 Faltungsantworten vor.
Eine andere Möglichkeit ist, die Farbbilder vor der Faltung in Grauwertbilder zu konvertieren. Eine einfache Methode ist, für jedes Pixel (x, y) mit dem Farbwert $\mathbf{g}(x, y) = (R, G, B)$ die 3 Farbkomponenten zu mitteln, um den Grauwert dieses Pixels $\hat{\mathbf{g}}(x, y)$ zu berechnen[1]:

$$\hat{\mathbf{g}}(x, y) = \left[\frac{R + G + B}{3}\right] . \tag{2.25}$$

[1] Mit $[\cdot]$ wird der ganzzahlige Teil eines reellen Wertes bezeichnet.

Bei der Faltung mit Farbbildern liegen dreimal so viele Antworten, also mehr Information vor, als bei der Faltung mit Grauwertbildern. Andererseits führt dies auch zu einem größeren Rechenaufwand sowohl bei der Faltung als auch beim Training der neuronalen Netze. Bei der Grauwertbildfaltung wird das Rauschen in den 3 Farbbänden gemittelt, was die Rauschunterdrückung des Gaborfilters zusätzlich unterstützt.
Um die Auswirkungen der beiden Faltungsmethoden auf die Orientierungsbestimmung vergleichen zu können, werden wiederum Versuche mit beiden Verfahren durchgeführt.

Erzeugung der Merkmalsvektoren

Die skalierten Gesichtsbilder werden mit k Gaborfilterpaaren verschiedener Orientierungen, aber gleicher Durchlaßbereiche gefaltet. Wenn mehrere unterschiedliche Objekte erkannt werden sollen, kann es hilfreich sein, mehrere Gaborfilterpaare mit verschiedenen Breiten und Orientierungen anzuwenden.
Die Bilder werden nicht an allen Punkten mit den Filtern gefaltet, sondern nur an rasterförmig, gleichabständig angeordneten $n \times n$ Aufpunkten. Diese Antworten liefern genügend Informationen, um die Gesichtsorientierung zu bestimmen.
Die Bestimmung der Breite des Gaborfilters geschieht in den Versuchen experimentell, d.h. die Breite, mit der die Gesichtsorientierung mit dem kleinsten Durchschnittsfehler gelernt werden kann, wird gewählt.
Abbildung 2.4 zeigt die Energie eines Bildes für 5 Gaborfilterpaare verschiedener Breite und je 4 verschiedenen Orientierungen.
Wenn ein Bild mit m Gaborfilterpaaren mit verschiedenen Durchlaßbereichen und je k Orientierungen an $n \times n$ Aufpunkten gefaltet wird, ergeben sich auf einem Grauwertbild $m \cdot k \cdot n^2$, auf einem RGB-Farbbild $3 \cdot m \cdot k \cdot n^2$ lokale Energiewerte.
Die so gewonnenen Werte werden zu einem Merkmalsvektor zusammengefaßt. Dieser bildet die Eingabe der neuronalen Netze. Wie aus diesen Eingaben die Gesichtsorientierung als Ausgabe berechnet werden kann, ist das Thema des nächsten Kapitels.

In Abbildung 2.5 sind die einzelnen Schritte der Bildverarbeitung in einem Flußdiagramm zusammengefaßt.

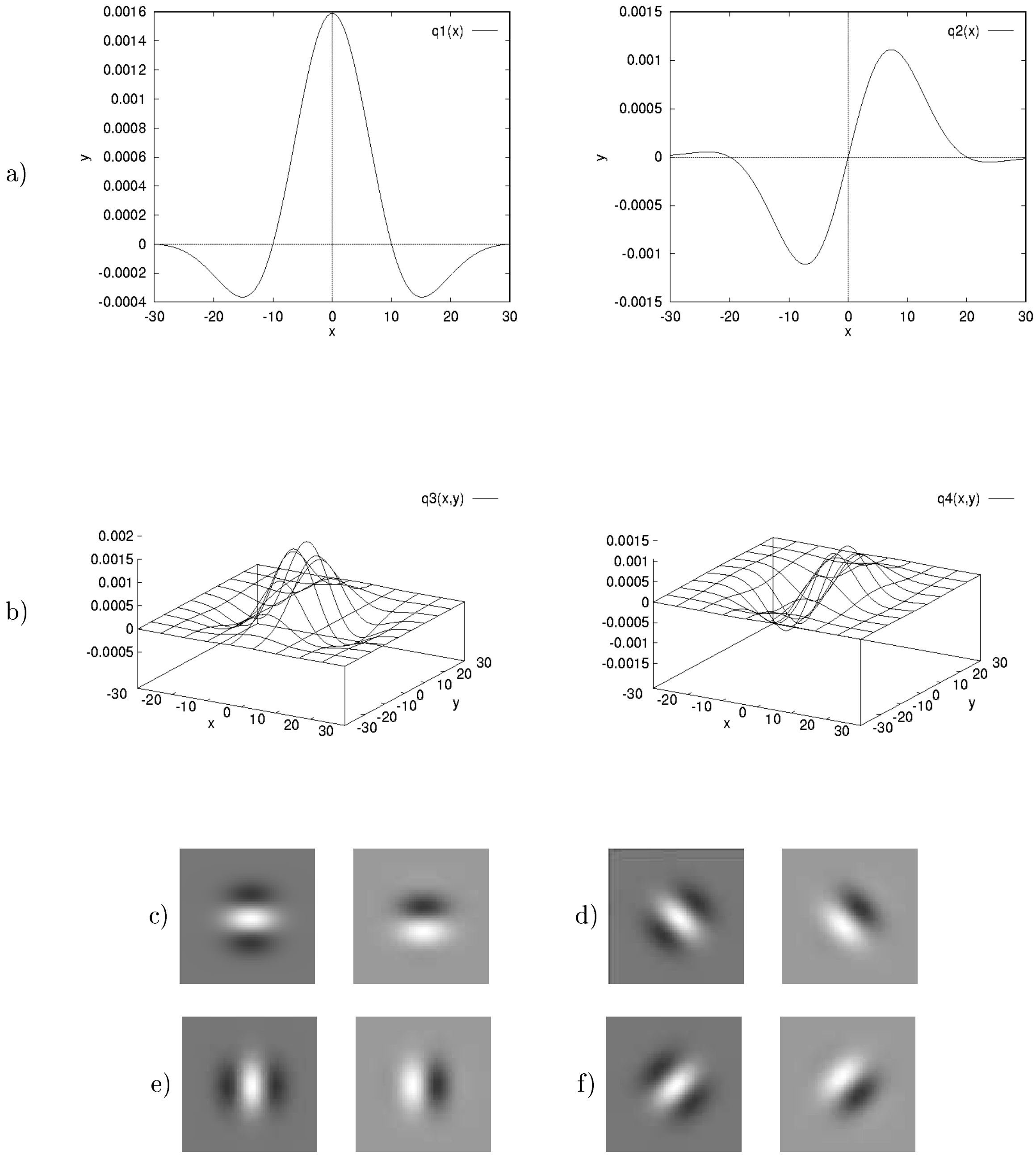


Abbildung 2.3: a) Ein eindimensionales und b) ein zweidimensionales Gaborfilterpaar. Links wird der gerade, rechts der ungerade (nicht gleichanteilsfreie) Filterteil dargestellt. (Parameter: $\lambda_x = 40$, $\sigma_x = 10$, $\theta = 0$.) Die unteren beiden Reihen zeigen 80×80 Filtermasken des (gleichanteilsfreien) zweidimensionalen Gaborfilterpaars in 4 verschiedenen Orientierungen (c) $\theta = 0$, d) $\theta = \frac{\pi}{4}$, e) $\theta = \frac{2\pi}{4}$, f) $\theta = \frac{3\pi}{4}$).

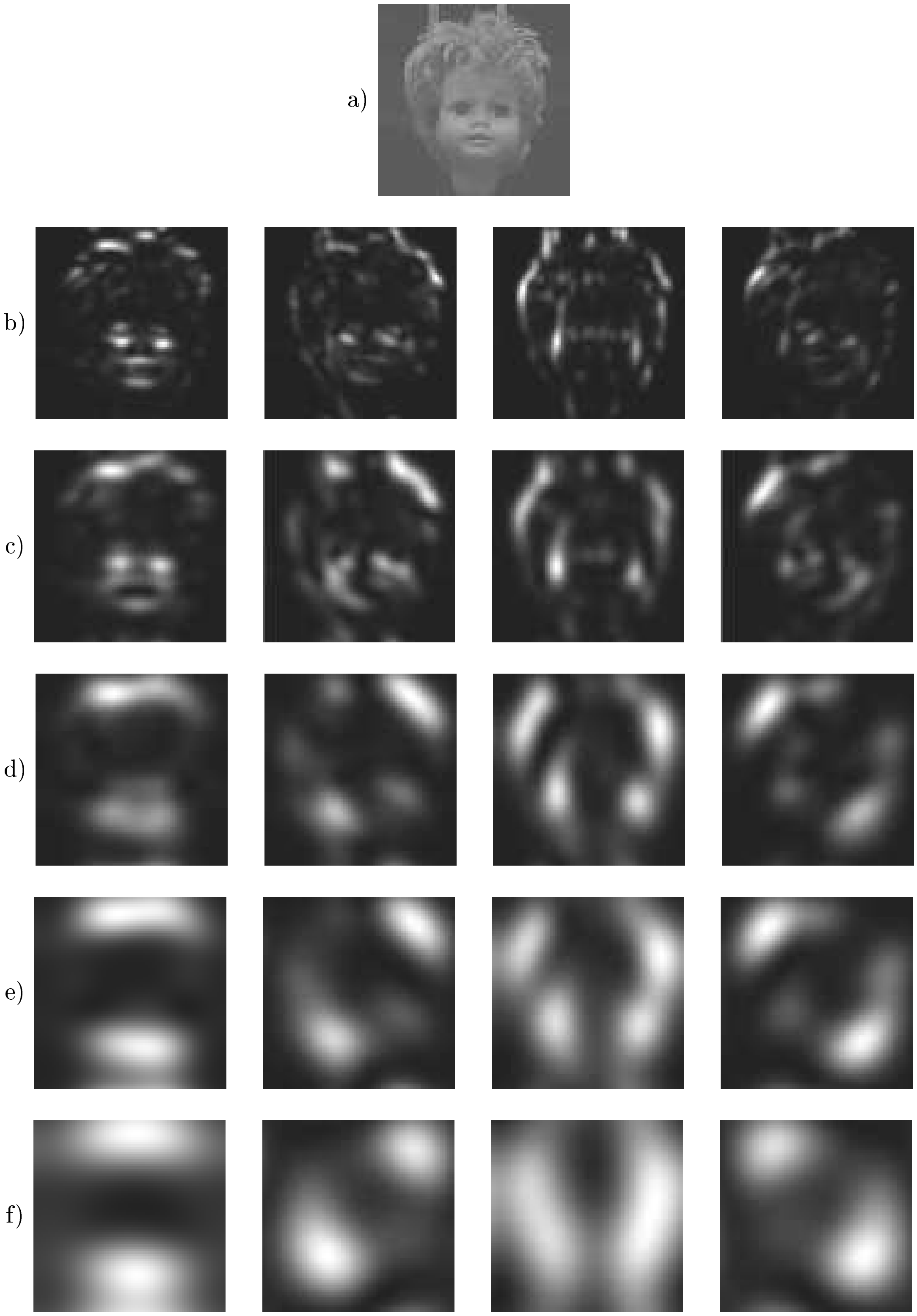


Abbildung 2.4: Faltung eines 80×80 Pixel großen Bildes mit 5 Gaborfiltern verschiedener Breite und je 4 verschiedenen Orientierungen. a) Das Originalbild. Darunter die Faltungsergebnisse des Bildes mit Gaborfiltern der Breite b) $\lambda_x = 8$, c) $\lambda_x = 15$, d) $\lambda_x = 30$, e) $\lambda_x = 40$ und f) $\lambda_x = 60$. In jeder der Zeilen b)-f) von links nach rechts haben die Filter die Orientierungen $\theta = 0, \frac{\pi}{4}, \frac{2\pi}{4}$ und $\frac{3\pi}{4}$.

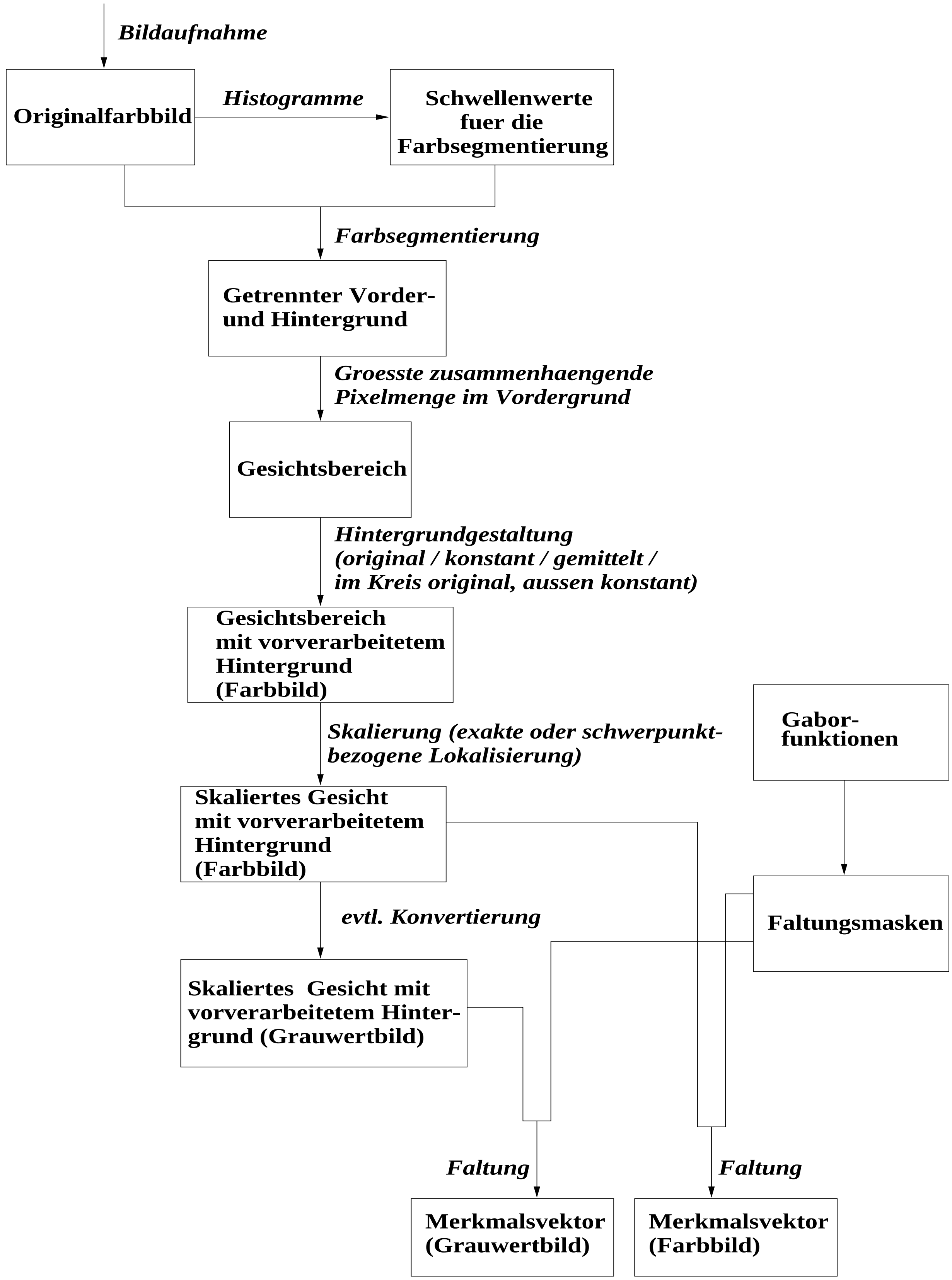


Abbildung 2.5: Die einzelnen Schritte der Merkmalsextraktion.

Kapitel 3

Neuronale Netze

Die Entwicklung von autonomen Systemen ist ein aktuelles Forschungsgebiet der Informatik. Autonome Systeme sind in der Lage, Signale aus ihrer Umgebung wahrzunehmen und darauf zu reagieren. Bei der Verarbeitung der Umgebungssignale werden zwei Methoden unterschieden: Entweder werden dem System alle Informationen über dessen Umgebung vorgegeben, die es für die weiteren Berechnungen braucht, oder man läßt das System die Informationen, die es braucht, aus den Signalen extrahieren. Mit anderen Worten, entweder wird das System an die Umgebung angepaßt, oder man läßt es lernen, sich an die Umgebung anzupassen. Formal entspricht letzteres der Approximation einer unbekannten Funktion $F : \mathcal{R}^m \rightarrow \mathcal{R}^n$ aufgrund vorgegebener Stützstellen.
Bei der Entwicklung lernfähiger Systeme werden oft *künstliche neuronale Netze* verwendet. Es folgt im Abschnitt 3.1 eine kurze Einführung in die Theorie der künstlichen neuronalen Netze (im folgenden neuronale Netze). Für eine ausführlichere Beschreibung siehe z.B. [RMS91].
In den darauffolgenden Abschnitten werden spezielle neuronale Netzwerke vorgestellt, die in dieser Arbeit Anwendung finden.

3.1 Einführung

3.1.1 Die Netzstruktur

Neuronale Netze sind graphähnliche Strukturen. Sie bestehen aus N *Knoten* (*Neuronen*) und K *Kanten*. Die Kanten haben Gewichtungen, die die Stärken der Verbindungen angeben. Eine Kante, die vom Neuron i zum Neuron j verläuft, wird mit (i, j) bezeichnet, seine Gewichtung mit g_{ij}. Die Menge aller Knoten ist V, die Menge aller Kanten ist $E = \{(i, j)|i, j \in V, g_{ij} \neq 0\}$[1]. Ist $(i, j) \in E$, so sind die Knoten i und j *benachbart*. Die *Nachbarschaft* eines Neurons $i \in V$ ist definiert durch

$$Nh(i) = \{j \in V|(i, j) \in E\}. \tag{3.1}$$

[1] $g_{ij} = 0 \Leftrightarrow$ Es verläuft keine Kante zwischen den Neuronen i und j.

Bei den von uns verwendeten Netzen sind die Verbindungen ungerichtet, d.h.

$$(i,j) \in E \Leftrightarrow (j,i) \in E. \tag{3.2}$$

Die Struktur eines Netzes wird durch die Anzahl der Neuronen und der Verbindungsstruktur zwischen den Neuronen bestimmt. Diese ist bei den klassischen Netzwerken meistens vorinitialisiert und wird während der Lernphase nicht geändert (z.B. bei Kohonens Karte [Koh84]). Wenn sich die Struktur während der Trainingsphase ändert, spricht man von einem *dynamischen* Netz.

3.1.2 Das Zentrum des rezeptiven Feldes

Die Neuronen liegen in dem Eingaberaum, der durch die Menge der möglichen Eingabevektoren aufgespannt wird. Das *Zentrum des rezeptiven Feldes* $\mathbf{w_i} \in \mathcal{R}^m$ eines Neurons $i \in V$ gibt seine Lage im Eingaberaum an.
Die Zentren werden während der Trainingsphase an die Lage der Trainingsdaten angepaßt. Für einen *Eingabevektor* $\mathbf{x} \in \mathcal{R}^m$ wird das zu $\mathbf{x}$ naheliegendste Neuron mit $bmu(\mathbf{x})$ (*best matching unit*), das zweitnaheliegendste mit $second(\mathbf{x})$ bezeichnet. Es gilt:

$$\forall i \in V : ||\mathbf{w_i} - \mathbf{x}||_2 \geq ||\mathbf{w_{bmu(x)}} - \mathbf{x}||_\mathbf{2} \tag{3.3}$$

und

$$(second(\mathbf{x}) \neq bmu(\mathbf{x})) \wedge \left(\forall i \in V \backslash \{bmu(\mathbf{x})\} : ||\mathbf{w_i} - \mathbf{x}||_2 \geq ||\mathbf{w_{second(x)}} - \mathbf{x}||_\mathbf{2}\right). \tag{3.4}$$

Die Lage des Zentrums und die Aktivierungsfunktion eines Neurons bestimmen dessen Aktivierung für eine bestimmte Eingabe.

3.1.3 Die Aktivierung

Ein Neuron $i \in V$ des Netzes wird durch einen *Eingabevektor* (Stimulus) $\mathbf{x} \in \mathcal{R}^m$ *aktiviert*. Die Aktivierung $a_i(\mathbf{x}) \in \mathcal{R}$ hängt von der *Aktivierungsfunktion* $a_i(\mathbf{x}) : \mathcal{R}^m \to [0,1]$ des Neurons ab. Je kleiner der Abstand $||\mathbf{w_i} - \mathbf{x}||_2$ zwischen dem Zentrum des rezeptiven Feldes $\mathbf{w_i}$ eines Neurons i und dem Eingabevektor $\mathbf{x}$ ist, desto größer ist die Aktivierung $a_i(\mathbf{x})$ dieses Neurons. Die Aktivierung hat immer einen Wert zwischen 0 und 1. Das *rezeptive Feld* oder der *Einzugsbereich* eines Neurons ist die Umgebung, in der die Aktivierung des Neurons größer als null ist.
Oft wird die Gaußfunktion als Aktivierungsfunktion verwendet:

$$a_i(\mathbf{x}) = \exp\left(-\frac{||\mathbf{w_i} - \mathbf{x}||_2^2}{\sigma_i^2}\right). \tag{3.5}$$

In diesem Fall ist die Aktivierungsfunktion symmetrisch, d.h. die Aktivierung hängt nur von dem Abstand $||\mathbf{w_i} - \mathbf{x}||_2$ ab. Zwei verschiedene Eingabevektoren mit der gleichen Entfernung vom Zentrum des Neurons i erzeugen die gleiche Aktivierung. Dabei bestimmt σ_i

die Breite des rezeptiven Feldes von Neuron i. Diese Breiten sollten so gewählt werden, daß sich die Einzugsbereiche benachbarter Neuronen etwas überlappen, aber die Aktivierung eines Neurons für eine Eingabe in der Nähe des Zentrums eines benachbarten Neurons nur noch sehr klein ist. Andererseits sollen die Einzugsbereiche auch nicht zu klein gewählt werden, sonst sind die Aktivierungen für Eingaben, die in der Mitte zwischen zwei Neuronen liegen, nahe null, was zu einer fehlerhaften Netzausgabe führt.
Abbildung 3.1 veranschaulicht die Bedeutung der Parameter σ_i. Die Aktivierungsfunktionen zweier benachbarten Neuronen mit den Zentren $w_1 = 2$ und $w_2 = 3$ sind mit verschiedenen Breiten dargestellt. Mit $\sigma_1 = \sigma_2 = 1$ (entspricht der Entfernung zwischen den Neuronen) überlappen sich die Aktivierungsfunktionen zu stark. Mit $\sigma_1 = \sigma_2 = 0.1$ sind die Breiten zu klein gewählt und die mittleren Punkte zwischen den Neuronen fallen weder in den Einzugsbereich des ersten noch des zweiten Neurons. Eine ideale Breite wird mit $\sigma_1 = \sigma_2 = 0.5$ erreicht (die Hälfte der Entfernung zwischen den beiden Zentren). Bei dynamischen Netzwerken ist die Anpassung der Aktivierungsfunktionen an die Netzstruktur

a)

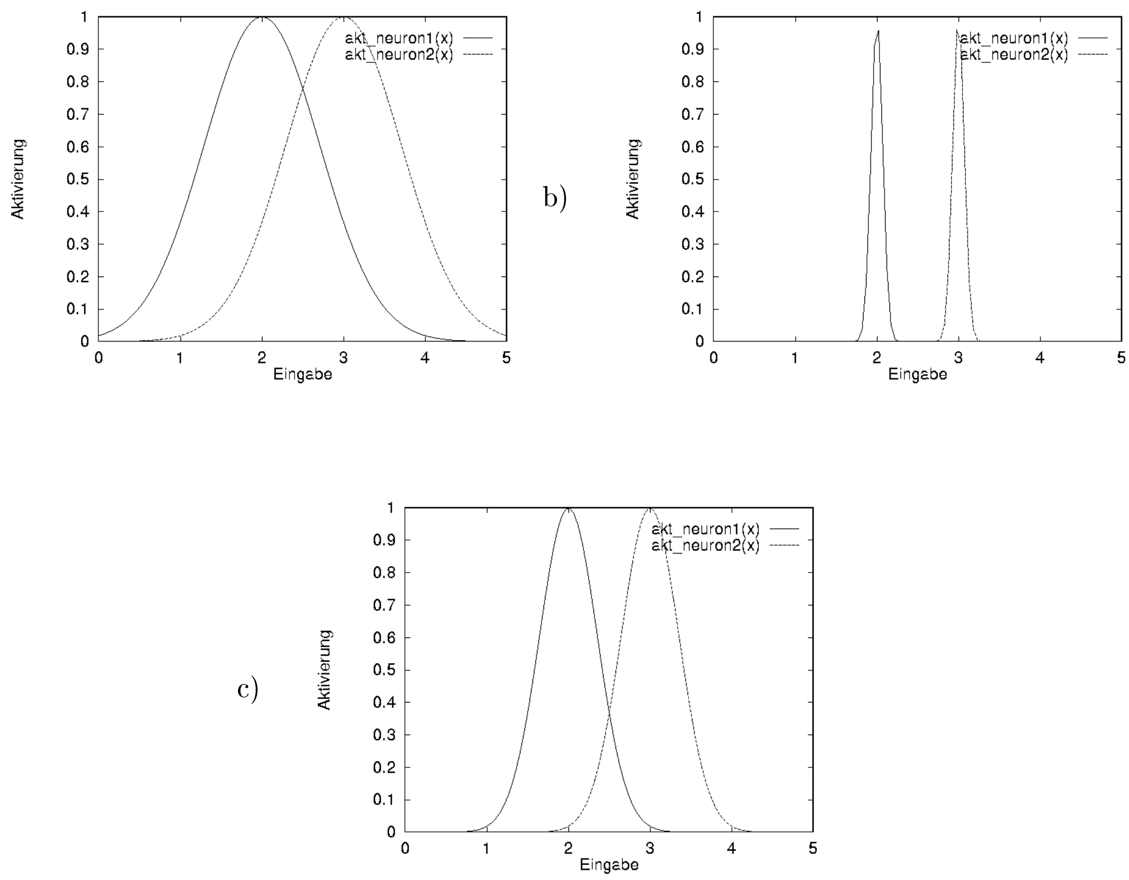


Abbildung 3.1: Die Aktivierungsfunktionen zweier benachbarten Neuronen mit den Zentren $w_1 = 2$ und $w_2 = 3$ mit verschiedenen Breiten der Gaußfunktion: a) $\sigma_1 = \sigma_2 = 1$ b) $\sigma_1 = \sigma_2 = 0.1$ c) $\sigma_1 = \sigma_2 = 0.5$.

sehr wichtig. Ein geeignetes Verfahren zur Bestimmung der Breiten der Gaußfunktionen wird in Kapitel 3.2 vorgestellt.

3.1.4 Die Netzausgabe

In dieser Arbeit werden neuronale Netzwerke mit überwachtem Lernverfahren verwendet (s.u.). Beim überwachten Lernen wird jedem Neuron i ein *Ausgabevektor* $\mathbf{o_i} \in \mathcal{R}^n$ zugeordnet. Die Aktivierung $a_i(\mathbf{x})$ eines Neurons i für eine Eingabe $\mathbf{x}$ legt fest, wie stark sich dieses Neuron mit seinem Ausgabevektor in der Netzausgabe beteiligt. Die Netzausgabe ist bei den meisten Netzen die mit den Aktivierungen gewichtete Summe der Ausgabevektoren der einzelnen Neuronen:

$$\mathbf{o_{net}}(\mathbf{x}) = \sum_{i \in V} a_i(\mathbf{x})\mathbf{o_i}. \tag{3.6}$$

3.1.5 Das Training

Der technischen Anwendung eines neuronalen Netzes geht eine *Trainingsphase* oder *Lernphase* voraus, in der die Approximation der unbekannten Funktion $F : \mathcal{R}^m \to \mathcal{R}^n$ gelernt wird. Lernen bedeutet die Entwicklung der Netzstruktur (Anzahl der Neuronen und die Verbindungsstruktur zwischen den Neuronen) und die Berechnung der Zentren und der Ausgaben der Neuronen, so daß ein minimaler Approximationsfehler erreicht wird.
Das Lernen geschieht *unüberwacht* oder *überwacht*. Beim unüberwachten Lernen wird nur die Topologie des Eingaberaumes gelernt, das Netz erzeugt keine Ausgaben. Beim überwachten Lernen müssen zusätzlich noch die Ausgaben der Neuronen gelernt werden.
Das Training kann *offline* oder *online* verlaufen. Das *offline-Training* geschieht mit Hilfe einer *Trainingsmenge* $Train$, die aus Abtastpunkten (*Trainingspunkte* oder *Trainingsdaten*) der zu lernenden Funktion F besteht:

$$Train \subset \{(\mathbf{x}, \mathbf{y}) \in \mathcal{R}^m \times \mathcal{R}^n | F(\mathbf{x}) = \mathbf{y}\}. \tag{3.7}$$

Dabei wird $\mathbf{x}$ der *Eingabevektor* und $\mathbf{y}$ der *Ausgabevektor* des Trainingsdatums genannt.
Das *online-Training* basiert auch auf Abtastpunkten der Funktion F, das Netz hat aber zu jedem Zeitpunkt t nur auf einen Trainingspunkt $(\mathbf{x^t}, \mathbf{y^t})$ Zugriff, d.h. die Trainingspunkte werden nicht gespeichert.
Die Genauigkeit der Approximation der Funktion F durch das Netz kann mit dem durchschnittlichen Ausgabefehler des Netzes auf der *Testmenge* gemessen werden:

$$err = \frac{\sum_{(\mathbf{x},\mathbf{y}) \in Test} ||\mathbf{o_{net}}(\mathbf{x}) - \mathbf{y}||_2}{|Test|}, \tag{3.8}$$

wobei

$$Test \subset \{(\mathbf{x}, \mathbf{y}) \in \mathcal{R}^m \times \mathcal{R}^n | F(\mathbf{x}) = \mathbf{y}\} \tag{3.9}$$

eine der Trainingsmenge ähnliche Abtastpunktmenge der Funktion F ist. (Der Ausgabefehler wird wegen eines möglichen Auswendiglernens der Trainingsdaten nicht auf der Trainingsmenge gemessen.)
Das Training wird beendet, wenn eine vorgegebene *Abbruchsbedingung* erfüllt ist. Diese kann z.B. das Unterschreiten eines minimalen Testfehlers Θ sein:

$$err < \Theta. \tag{3.10}$$

Im Folgenden werden einige neuronale Netzwerke vorgestellt.

3.2 Das DCS-Netz

Das DCS-Netz (Dynamic Cell Structure) ist ein dynamisches neuronales Netz, d.h. seine Struktur ändert sich während der Lernphase [BS95]. Diese Eigenschaft ermöglicht es dem Netz, sich der Topologie des Eingaberaumes anzupassen.
Dynamische Netzwerke stellen aber auch besondere Anforderungen an die Aktivierungsfunktionen. Damit sie während des ganzen Trainings optimiert gestaltet bleiben, müssen sie sich bei jeder Änderung der Struktur selbst ändern, d.h. selsbt dynamisch sein.

3.2.1 Die Aktivierungsfunktion

Die Aktivierungsfunktionen der Neuronen im Netz sind modifizierte Gaußfunktionen der Form

$$a_i(\mathbf{x}) = \begin{cases} \exp\left(-\frac{||\mathbf{w_i}-\mathbf{x}||_2^2}{\sigma_i^2} \cdot (-\ln p)\right) & : \ i \in \{bmu(\mathbf{x})\} \cup Nh(bmu(\mathbf{x})) \\ 0 & : \ sonst \end{cases}, \tag{3.11}$$

wobei $\mathbf{x}$ die Eingabe und $\mathbf{w_i}$ das Zentrum des Neurons i bezeichnet. Die Breite des rezeptiven Feldes von Neuron i wird durch σ_i angegeben. Ferner ist $0 \leq p \leq 1$ ein konstanter Faktor.
Die Fallunterscheidung bewirkt, daß die Netzausgabe nur von den Ausgaben der $bmu(\mathbf{x})$ und ihrer Nachbarn abhängt. Dies ist sinnvoll, weil weiter entfernte Neuronen aufgrund der Lokalität der Aktivierungsfunktionen eine sehr geringe Aktivierung aufweisen.
Im DCS-Netz sind die Aktivierungsfunktionen der einzelnen Neuronen verschieden und an die lokale Struktur des Netzes angepaßt. Um die Aktivierungsfunktionen optimiert zu gestalten, definieren wir die Breiten der Gaußfunktionen in Abhängigkeit von der mittleren quadratischen Entfernung zu den benachbarten Neuronen:

$$\sigma_i^2 = \frac{\sum_{j \in Nh(i)} ||\mathbf{w_i} - \mathbf{w_j}||_2^2}{|Nh(i)|}. \tag{3.12}$$

Wenn die Neuronen gleichmäßig verteilt sind und die Eingabe $\mathbf{x}$ in der Nähe des Zentrums eines mit i benachbarten Neurons liegt, so gilt $\sigma_i^2 \approx ||\mathbf{w_i} - \mathbf{x}||_2^2$ und $a_i(\mathbf{x}) \approx \exp(\ln p) = p$.

Der konstante Faktor p gibt also an, wie groß die Aktivierung des Neurons i in der Nähe der Zentren der benachbarten Neuronen ist, d.h. wie stark sich die Aktivierungsfunktionen benachbarter Neuronen überlappen.
Die Gesamtaktivierung des Netzes für eine Eingabe $\mathbf{x}$

$$sumAct(\mathbf{x}) = \sum_{l \in V} a_l(\mathbf{x}) \tag{3.13}$$

hängt u.a. von der Anzahl der Nachbarn der *best matching unit*, $bmu(\mathbf{x})$, ab. Da die Neuronen im Allgemeinen unterschiedlich viele Nachbarn haben, kann die Gesamtaktivierung für verschiedene Eingaben ganz unterschiedliche Werte annehmen. Für ein erfolgreiches Training wäre eine konstante Gesamtaktivierung von großem Vorteil. Die Gesamtaktivierung des Netzes wird deshalb auf 1 normiert. Somit ändert sich die Aktivierungsfunktion des Neurons i zu

$$a_i(\mathbf{x}) = \begin{cases} \exp\left(-\frac{\|\mathbf{w_i}-\mathbf{x}\|_2^2}{\sigma_i^2} \cdot (-\ln p)\right) \cdot \frac{1}{sumAct(\mathbf{x})} & : \ i \in \{bmu(\mathbf{x})\} \cup Nh(bmu(\mathbf{x})) \\ 0 & : \ sonst \end{cases} , \tag{3.14}$$

wobei

$$sumAct(\mathbf{x}) = \sum_{l \in \{bmu(\mathbf{x})\} \cup Nh(bmu(\mathbf{x}))} \exp\left(-\frac{\|\mathbf{w_l}-\mathbf{x}\|_2^2}{\sigma_l^2} \cdot (-\ln p)\right) \tag{3.15}$$

die Summe der unnormierten Aktivierungen bezeichnet.
Die Netzantwort auf eine bestimmte Eingabe $\mathbf{x}$ kann auch nur aufgrund der Ausgabe der *best matching unit*, $bmu(\mathbf{x})$, ohne ihre Nachbarn erfolgen. Damit die Gesamtaktivierung des Netzes 1 bleibt, wird die Aktivierung der *best matching unit* auf 1 und die Aktivierungen aller anderen Neuronen auf 0 gesetzt. Die Aktivierungsfunktion vereinfacht sich damit zu

$$a_i(\mathbf{x}) = \begin{cases} 1 & : \ i = bmu(\mathbf{x}) \\ 0 & : \ sonst \end{cases} . \tag{3.16}$$

Die Netzausgabe (3.6) ist in diesem Fall $\mathbf{o_{net}}(\mathbf{x}) = \mathbf{o_{bmu(x)}}$. Die Berechnung der Netzausgabe wird also zu einem einfachen *table lookup* Verfahren.

Das DCS-Netz kann sowohl *offline* als auch *online* trainiert werden. Beim *offline*-Training steht dem Netz eine Trainingsmenge zur Verfügung. Zu jedem Zeitpunkt hat das Netz Zugriffsmöglichkeit auf die gesamte Trainingsmenge. Dadurch können die Netztopologie und die Ausgaben in jedem Trainingsschritt global auf dem gesamten Ein- und Ausgaberaum gelernt werden. Beim *online*-Training liegt zu jedem Zeitpunkt nur ein Trainingsdatum vor. Das Lernen geschieht schrittweise in einer lokalen Umgebung des vorliegenden Trainingsdatums mittels eines Gradientenabstiegsverfahrens.

3.2.2 Das *offline*-Training des DCS-Netzes

Lernen der Topologie

Im Laufe des Trainings ändert sich sowohl die Anzahl der Neuronen als auch die Verbindungsstruktur zwischen den Neuronen.

Im ersten Trainingsschritt wird das Netz mit einem Neuron initialisiert, dessen Zentrum und Ausgabevektor auf den Ein- und den Ausgabevektor des ersten Trainingsdatums gesetzt wird.
In jedem weiteren Trainingsschritt k wird dem Netz ein neues Neuron n_k zugefügt. Es gibt unterschiedliche Methoden zur Bestimmung des Zentrums und des Ausgabevektors des neuen Neurons. In dieser Arbeit wird immer dort ein neues Neuron eingefügt, wo der durchschnittliche lokale Trainingsfehler am größten ist.
Dafür wird der durchschnittliche Trainingsfehler in dem Einzugsbereich jedes Neurons n_i, $i = 1, \cdots, k-1$ berechnet:

$$error_{n_i} = \frac{1}{|T_{n_i}|} \sum_{(\mathbf{x},\mathbf{y}) \in T_{n_i}} ||\mathbf{o_{net}}(\mathbf{x}) - \mathbf{y}||_2, \tag{3.17}$$

wobei $T_{n_i} = \{(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \in Train | bmu(\tilde{\mathbf{x}}) = n_i\}$ ist. Das neue Neuron n_k wird dann in die Nähe desjenigen Neurons n^* eingefügt, dessen durchschnittlicher Trainingsfehler am größten ist:

$$\forall n_i \in V : error_{n_i} \leq error_{n^*}, \tag{3.18}$$

wobei V die Menge der Neuronen bezeichnet.
Das Zentrum des neuen Neurons $\mathbf{w_k}$ ist der Eingabevektor des Trainingspunktes mit dem größten Fehler im Einzugsbereich des Neurons n^*, d.h. $\mathbf{w_k} = \mathbf{x}$ mit

$$\exists \mathbf{y} \in \mathcal{R}^n : \left((\mathbf{x}, \mathbf{y}) \in T_{n^*} \wedge ||\mathbf{o_{net}}(\mathbf{x}) - \mathbf{y}||_2 = max_{(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \in T_{n^*}} \{||\mathbf{o_{net}}(\tilde{\mathbf{x}}) - \tilde{\mathbf{y}}||_2\}\right). \tag{3.19}$$

In Abbildung 3.2 ist der Algorithmus zum Einfügen eines neuen Neurons in das *offline* DCS-Netz zusammengefaßt.
Damit die Neuronen im Eingaberaum gleichmäßig verteilt sind, werden die Zentren aller Neuronen nach dem *K-means*-Verfahren [HH93] neu bestimmt. Dieser verschiebt die Zentren aller Neuronen in den Schwerpunkt der in den Einzugsbereich des Neurons fallenden Trainingsdaten. Das neue Zentrum des Neurons n_i berechnet sich durch

$$\mathbf{w_{n_i}^{neu}} = \frac{1}{|T_{n_i}|} \sum_{(\mathbf{x},\mathbf{y}) \in T_{n_i}} \mathbf{x}. \tag{3.20}$$

Die Verschiebung aller Zentren wird solange wiederholt, bis keine Änderungen mehr vorkommen.
Abbildung 3.3 zeigt den *K-means*-Algorithmus.

Nach der Adaption der Zentren wird die Verbindungsstruktur des Netzes neu gestaltet. In einem ersten Schritt werden alle bisherigen Verbindungen zwischen den Neuronen gelöscht:

$$\forall (i,j) \in E : g_{ij} = 0. \tag{3.21}$$

In einem zweiten Schritt werden für alle Trainingspunkte $(\mathbf{x}, \mathbf{y}) \in Train$ die Neuronen $bmu(\mathbf{x})$ und $second(\mathbf{x})$ bestimmt und miteinander verbunden. (Die Gewichtungen der Verbindungen spielen in der *offline*-Anwendung keine Rolle.)

$$\forall (\mathbf{x}, \mathbf{y}) \in Train : \left(g_{bmu(\mathbf{x}),second(\mathbf{x})} = 1 \wedge g_{second(\mathbf{x}),bmu(\mathbf{x})} = 1\right). \tag{3.22}$$

```
/* Neues Neuron einfuegen */
/* Bezeichnungen: */
/*  k gibt die Anzahl der Neuronen im Netz an, */
/*  V = {n_1, ..., n_k} ist die Menge der Neuronen und */
/*  Train = {(x_1, y_1), ..., (x_p, y_p)} ist die Trainingsmenge. */
/*  insert_new_neuron() fuegt ein neues Neuron in das Netz ein */
/*  und gibt einen Zeiger auf das neue Neuron zurueck. */
   proc new_neuron()
   begin
      for i := 1 to k do
      begin
         error_{n_i} := 0;
         anzahl_{n_i} := 0;
         max_error_{n_i} := 0;
         max_error_index_{n_i} := 0;
      end
      for i := 1 to |Train| do
      begin
         error_{bmu(x_i)} := error_{bmu(x_i)} + ||o_net(x_i) - y_i||_2;
         anzahl_{bmu(x_i)} := anzahl_{bmu(x_i)} + 1;
         if (max_error_{bmu(x_i)} < ||o_net(x_i) - y_i||_2) then
         begin
            max_error_{bmu(x_i)} := ||o_net(x_i) - y_i||_2;
            max_error_index_{bmu(x_i)} := i;
         end
      end
      n* := n_1;
      for i := 1 to k do
      begin
         error_{n_i} := error_{n_i}/anzahl_{n_i};
         if (error_{n*} < error_{n_i}) then
            n* := n_i;
      end
      /* Fuege ein neues Neuron in das Netz ein */
      n^neu := insert_new_neuron();
      k := k + 1;
      w_{n^neu} := x_{max_error_index_{n*}};
      o_{n^neu} := y_{max_error_index_{n*}};
   end
```

Abbildung 3.2: Algorithmus zum Einfügen eines neuen Neurons in das *offline* DCS-Netz.

```
/* K − means − Algorithmus */
/* Bezeichnungen : */
/*  k gibt die Anzahl der Neuronen im Netz an, */
/*  V = {n_1, ··· , n_k} ist die Menge der Neuronen und */
/*  Train = {(x_1, y_1), ··· , (x_p, y_p)} ist die Trainingsmenge. */
    proc K − means()
    begin
        do
            ready := TRUE;
            for i := 1 to k do
            begin
                summe_{n_i} := 0;
                anzahl_{n_i} := 0;
            end
            for i := 1 to |Train| do
            begin
                summe_{bmu(x_i)} := summe_{bmu(x_i)} + x_i;
                anzahl_{bmu(x_i)} := anzahl_{bmu(x_i)} + 1;
            end
            for i := 1 to k do
                if (w_{n_i} <> summe_{n_i} / anzahl_{n_i}) then
                begin
                    w_{n_i} := summe_{n_i} / anzahl_{n_i};
                    ready := FALSE;
                end
        until (ready)
    end
```

Abbildung 3.3: Der *K-means*-Algorithmus.

Abbildung 3.4 zeigt die Bestimmung einer optimierten Verbindungsstruktur in Prozedurform.

Lernen der Ausgaben

In jedem Trainingsschritt, nachdem ein neues Neuron in das Netz eingefügt und die Zentren und die Netzstruktur adaptiert worden sind, werden die Ausgaben der Neuronen neu berechnet.
Für jeden Trainingspunkt $(\mathbf{x}, \mathbf{y}) \in Train$ sollte das Netz im Idealfall auf die Eingabe $\mathbf{x}$ die Netzausgabe $\mathbf{y}$ erzeugen. Versehen wir die Trainingsdaten mit Indices

```
/* Bestimmung der Verbindungsstruktur */
/* Bezeichnungen: */
/*  k gibt die Anzahl der Neuronen im Netz an, */
/*  V = {n_1, ..., n_k} ist die Menge der Neuronen und */
/*  Train = {(x_1, y_1), ..., (x_p, y_p)} ist die Trainingsmenge. */
    proc connections()
    begin
        for i := 1 to k do
            for j := 1 to k do
                g_{n_i,n_j} := 0;
        for i := 1 to |Train| do
        begin
            g_{bmu(x_i),second(x_i)} := 1;
            g_{second(x_i),bmu(x_i)} := 1;
        end
    end
```

Abbildung 3.4: Algorithmus zur Bestimmung der optimierten Verbindungsstruktur des *offline* DCS-Netzes.

$Train = \{(\mathbf{x_i}, \mathbf{y_i}) | i = 1, \cdots, t\}$, $t = |Train|$, so sollte für alle $i = 1, \cdots, t$ gelten:

$$\mathbf{y_i} = \mathbf{o_{net}}(\mathbf{x_i}) = \sum_{j \in V} a_j(\mathbf{x_i})\mathbf{o_j}, \tag{3.23}$$

wobei V die Menge der Neuronen im Netz bezeichnet. Die t Gleichungen werden zu einem Gleichungssystem zusammengefaßt:

$$\begin{pmatrix} a_1(\mathbf{x_1}) & \cdots & a_N(\mathbf{x_1}) \\ \vdots & & \vdots \\ a_1(\mathbf{x_t}) & \cdots & a_N(\mathbf{x_t}) \end{pmatrix} \begin{pmatrix} \mathbf{o_1^T} \\ \vdots \\ \mathbf{o_N^T} \end{pmatrix} = \begin{pmatrix} \mathbf{y_1^T} \\ \vdots \\ \mathbf{y_t^T} \end{pmatrix}, \tag{3.24}$$

oder Komponentenweise

$$\begin{pmatrix} a_1(\mathbf{x_1}) & \cdots & a_N(\mathbf{x_1}) \\ \vdots & & \vdots \\ a_1(\mathbf{x_t}) & \cdots & a_N(\mathbf{x_t}) \end{pmatrix} \begin{pmatrix} o_{1_1} & \cdots & o_{1_n} \\ \vdots & & \vdots \\ o_{N_1} & \cdots & o_{N_n} \end{pmatrix} = \begin{pmatrix} y_{1_1} & \cdots & y_{1_n} \\ \vdots & & \vdots \\ y_{t_1} & \cdots & y_{t_n} \end{pmatrix}, \tag{3.25}$$

wobei n die Ausgabedimension, N die Anzahl der Neuronen im Netz und t die Anzahl der Trainingsdaten angibt. Durch o_{i_j} wird die j-te Komponente des Ausgabevektors des Neurons i bezeichnet und y_{i_j} ist die j-te Komponente der Soll-Ausgabe $\mathbf{y_i}$ des i-ten Trainingsdatums.
Ist $t < N$, so ist das Gleichungssystem unterbestimmt. In diesem Fall existiert ein Lösungsraum. Die Trainingsdaten werden dann "auswendig gelernt", eine gute Approximation der zu lernende Funktion F zwischen den Stützstellen wird aber meistens nicht

```
/* DCS – Training offline */
/* Bezeichnungen : */
/*  Test = {(x̃_1, ỹ_1), ··· , (x̃_q, ỹ_q)} ist die Testmenge und */
/*  calculate_outputs() ist eine Procedur, die die Ausgabevektoren */
/*  der Neuronen berechnet. */
    proc DCS – offline()
    begin
        k := 0;
        do
            /* Lernen der Topologie */
            new_neuron();
            K – means();
            connections();
            /* Berechnung der Ausgabevektoren */
            /* der Neuronen */
            calculate_outputs();
            /* Testfehlerberechnung */
            test_error := 0;
            for i := 1 to |Test| do
                test_error := test_error + ||o_net(x̃_i) – ỹ_i||_2;
        until (test_error / |Test| < θ)
    end
```

Abbildung 3.5: Trainingsalgorithmus eines *offline* DCS-Netzes.

erreicht. Im Normalfall ist $N \leq t$ und es liegt ein lineares Approximationsproblem vor. Die beste Approximation wird mittels Singulärwertzerlegung bestimmt. Die Approximationsgüte hängt dabei von der Kondition der Aktivierungsmatrix ab.

Nachdem die Ausgaben der Neuronen berechnet worden sind, wird nach (3.8) der durchschnittliche Testfehler berechnet. Liegt der Fehler unter einem vorgegebenen Schwellenwert, so wird das Training beendet. Anderenfalls wird ein neuer Trainingsschritt durchgeführt. Der Schwellenwert sollte dabei über dem durch das Netz erreichbaren minimalen Fehler liegen.
In Abbildung 3.5 ist das gesamte *offline*-Training eines DCS-Netzes zusammengefaßt.

3.2.3 Das *online*-Training des DCS-Netzes

Beim *offline*-Training wird eine zur Verfügung stehende Trainingsmenge vorausgesetzt. In der Praxis ist eine Trainingsmenge aber nicht immer vorhanden.
Beim *online*-Training hat das Netz zu jedem Zeitpunkt nur auf einen Trainingspunkt Zugriff. Diese Tatsache bringt einige Nachteile mit sich. Der Ausgabefehler kann nicht mehr global auf der gesamten Trainingsmenge minimiert werden. Das Lernen geschieht hier

schrittweise und immer nur in einer lokalen Umgebung um die aktuelle Eingabe. Die Zentren bzw. die Ausgaben der Neuronen in dieser Umgebung werden adaptiert, indem sie in Abhängigkeit von ihrer Aktivierung in die Richtung des Ein- bzw. Ausgabevektors des aktuellen Trainingspunktes gezogen werden.
Beim *offline*-Training werden nach dem Einfügen eines neuen Neurons die ganze Verbindungsstruktur, die Zentren und die Ausgaben aller Neuronen neu berechnet. Das ist sinnvoll, weil die optimale Verteilung der Neuronen von deren Anzahl abhängig ist und die optimalen Ausgaben wiederum u.a. von den Zentren der Neuronen bestimmt werden.
Wenn beim *online*-Training ein neues Neuron in ein vortrainiertes Netz eingefügt wird, ist die Netzstruktur nicht mehr optimiert. D.h. die Zentren, die Verbindungsstruktur und die Ausgaben müssen korrigiert werden. Nach jedem Einfügen eines neuen Neurons wäre eine bestimmte Anzahl von weiteren Lernschritten nötig. Je größer das Netz ist, desto mehr Lernschritte müssen nach dem Einfügen eines neuen Neurons ausgeführt werden, um das Zentrum und die Ausgaben jedes Neurons zu optimieren.
Hinzu kommt, daß die Ausgaben der Neuronen in den Anwendungen dieser Arbeit wegen des Rauschens in den Eingabedaten nur mit kleinen Lernschritten verbessert werden können. Dadurch erhöht sich die für eine optimierte Approximation nötige Anzahl der Lernschritte.
Um den Rechenaufwand des Trainings möglichst klein zu halten, kann das *online* DCS-Netz mit einer vorgegebenen Anzahl von Neuronen vorinitialisiert werden. In diesem Fall werden die Neuronen nicht schrittweise nacheinander, sondern alle in einem Schritt eingefügt. In der Trainingsphase werden dann die Verbindungsstruktur, die Zentren und die Ausgaben der Neuronen schrittweise gelernt. Dadurch wird der Rechenaufwand deutlich verringert.
Um den Unterschied im Rechenaufwand des dynamisch wachsenden und des vorinitialisierten *online* DCS-Netzes zu veranschaulichen, führen wir einen Versuch durch. Die Aufgabe ist die Posenbestimmung eines Puppengesichtes, wobei die Pose in zwei Freiheitsgraden variieren kann (horizontales Drehen, vertikales Kippen). Es stehen 500 Trainingsbilder und 100 Testbilder zur Verfügung. Aus den Bildern werden mit den in Kapitel 2 beschriebenen Methoden Merkmalsvektoren extrahiert, die die Eingabevektoren der Trainingsdaten bilden. Die Ausgabevektoren sind die zugehörigen Orientierungen.
Um einen minimalen durchschnittlichen Testfehler (1.34 Grad/Drehrichtung) in der Posenbestimmung zu erreichen, braucht ein vorinitialisiertes LSDCS-Netz[2] mit *online*-Training 15000 Lernschritte. Um ähnliche Ergebnisse zu erzielen (1.26 Grad/Drehrichtung), sind beim dynamisch wachsenden *online* LSDCS-Netz 133000 Lernschritte nötig, die eine CPU-Zeit in der Größenordnung von ein bis zwei Stunden in Anspruch nehmen (15000 Lernschritte beim vorinitialisiertem Netz brauchen an dem gleichen Rechner 7 bis 8 Minuten).
Ein Nachteil der Vorinitialisierung ist, daß die optimale Anzahl von Neuronen nicht mehr so einfach bestimmt werden kann wie bei dynamisch wachsenden Strukturen, bei denen solan-

[2]Das LSDCS-Netz wird in Kapitel 3.4 eingeführt. Es handelt sich um ein dem DCS-Netz ähnliches Netzkonstrukt. Die Ausgaben der Neuronen werden linear erweitert. Die Berechnungen werden in niedrigdimensionalen lokalen intrinsischen Unterräumen durchgeführt.

```
/* Initialisierung des Netzes */
/* Bezeichnungen : */
/*  k gibt die Anzahl der zu initialisierenden Neuronen im Netz an, */
/*  V = {n_1, ..., n_k} ist die Menge der Neuronen im Netz,
/*  t ist die Anzahl der bisher durchgefuehrten Lernschritte. */
/*  new_trainingsdata() liest ein neues Trainingsdatum ein, */
/*  insert_new_neuron() fuegt ein neues Neuron in das Netz ein */
/*  und gibt einen Zeiger auf das neue Neuron zurueck. */
    proc online_initialize()
    begin
        t := 0;
        for i := 1 to k do
        begin
            (x, y) := new_trainingsdata();
            n^neu := insert_new_neuron();
            w_{n^neu} := x;
            o_{n^neu} := y;
        end
        for i := 1 to k do
            for j := 1 to k do
                g_{n_i,n_j} := 0;
        for i := 2 to k do
        begin
            g_{n_i,n_{i-1}} := 1;
            g_{n_{i-1},n_i} := 1;
        end
    end
```

Abbildung 3.6: Algorithmus zur Initialisierung eines *online* DCS-Netzes.

ge neue Neuronen in das Netz einfügt und die Netzparameter anschließend neu berechnet werden, bis der Ausgabefehler minimal wird.

Lernen der Topologie

Wie oben beschrieben, wird das Netz mit einer vorgegebenen Anzahl von k Neuronen initialisiert. Die Neuronen werden nacheinander in das Netz eingefügt, wobei die Zentren und die Ausgaben der Neuronen auf die Ein- bzw. Ausgabevektoren von k nacheinander eingelesenen Trainingsdaten gesetzt werden. Um eine anfängliche (willkürliche) Verbindungsstruktur herzustellen, wird jedes neu eingefügte Neuron (bis auf das erste) mit dem vorher eingefügten Neuron verbunden. Die Gewichtung der Verbindungen wird auf 1 gesetzt.
Die Initialisierungprozedur ist in Abbildung 3.6 verdeutlicht.

Das Training des Netzes geschieht in t_{max} Lernschritten. In jedem Trainingsschritt $0 < t \leq t_{max}$ wird ein neuer Trainingspunkt $(\mathbf{x^t}, \mathbf{y^t}) \in Train$ eingelesen und $bmu(\mathbf{x^t})$ und $second(\mathbf{x^t})$ bestimmt. Danach werden die Zentren der Neuronen gemäß der Lernregel

$$\begin{aligned} \mathbf{w}^{\mathbf{neu}}_{\mathbf{bmu(x^t)}} &= \mathbf{w}^{\mathbf{alt}}_{\mathbf{bmu(x^t)}} + a_{bmu(\mathbf{x^t})}(\mathbf{x^t})\epsilon_{bmu}(t)(\mathbf{x^t} - \mathbf{w}^{\mathbf{alt}}_{\mathbf{bmu(x^t)}}) \\ \forall i \in V\backslash\{bmu\} : \mathbf{w}^{\mathbf{neu}}_{\mathbf{i}} &= \mathbf{w}^{\mathbf{alt}}_{\mathbf{i}} + a_i(\mathbf{x^t})\epsilon_{Nh}(t)(\mathbf{x^t} - \mathbf{w}^{\mathbf{alt}}_{\mathbf{i}}) \end{aligned} \tag{3.26}$$

adaptiert, wobei die Aktivierungsfunktionen a_i wie beim *offline*-Training nach (3.14) bzw. (3.16) bestimmt werden und V die Menge der Neuronen bezeichnet.
Die Lernparameter $0 < \epsilon^{min}_{bmu} \leq \epsilon_{bmu}(t) \leq \epsilon^{max}_{bmu} \leq 1$ und $0 < \epsilon^{min}_{Nh} \leq \epsilon_{Nh}(t) \leq \epsilon^{max}_{Nh} \leq 1$ geben die Lernschrittweiten an, d.h. das Maß dafür, wie stark die Zentren in die Richtung der Eingabe des Trainingsdatums gezogen werden. Diese Parameter sind abhängig von der Anzahl der durchgeführten Lernschritten und werden bestimmt nach

$$\begin{aligned} \epsilon_{bmu}(t) &= \epsilon^{max}_{bmu} \left(\frac{\epsilon^{min}_{bmu}}{\epsilon^{max}_{bmu}}\right)^{\frac{t}{t_{max}}} \\ \epsilon_{Nh}(t) &= \epsilon^{max}_{Nh} \left(\frac{\epsilon^{min}_{Nh}}{\epsilon^{max}_{Nh}}\right)^{\frac{t}{t_{max}}} \quad . \end{aligned} \tag{3.27}$$

Sie haben vor dem ersten Trainingsschritt ihre Maximalwerte $\epsilon_{bmu}(0) = \epsilon^{max}_{bmu}$ und $\epsilon_{Nh}(0) = \epsilon^{max}_{Nh}$ und nehmen mit der Anzahl der durchgeführten Trainingsschritte exponentiell ab. Am Ende des Trainings mit $t = t_{max}$ erreichen sie ihre Minimalwerte $\epsilon_{bmu}(t_{max}) = \epsilon^{min}_{bmu}$ und $\epsilon_{Nh}(t_{max}) = \epsilon^{min}_{Nh}$. Man sagt, die Lernparameter werden während des Trainings "eingefroren". Die Lernschrittweite nimmt ab, damit beim Abstieg auf einer Fehlerfunktion am Anfang des Trainings relativ schnell in die Nähe des Minimums gelangt wird, um danach den Tiefpunkt mit kleineren Schritten finden zu können.

Nachdem die Zentren adaptiert worden sind, werden die Verbindungen aktualisiert. Falls noch keine Verbindung zwischen $bmu(\mathbf{x^t})$ und $second(\mathbf{x^t})$ existiert, wird eine neue Kante zwischen den beiden Neuronen mit der Gewichtung 1 eingefügt. Falls $bmu(\mathbf{x^t})$ und $second(\mathbf{x^t})$ schon miteinander verbunden waren, wird die Gewichtung auf 1 gesetzt:

$$\begin{aligned} g_{bmu(\mathbf{x^t}),second(\mathbf{x^t})} &= 1 \\ g_{second(\mathbf{x^t}),bmu(\mathbf{x^t})} &= 1 \quad . \end{aligned} \tag{3.28}$$

Danach werden alle Verbindungen geschwächt, indem die Gewichte vermindert werden nach der Lernregel

$$\forall (i,j) \in E : g^{neu}_{ij} = \begin{cases} \alpha g^{alt}_{ij} & : \ \alpha g^{alt}_{ij} > \theta \\ 0 & : \ \alpha g^{alt}_{ij} \leq \theta \end{cases} , \tag{3.29}$$

wobei θ ein Schwellenwert und $0 < \alpha < 1$ eine Vergessenskonstante ist. Ist die neue Gewichtung einer Verbindung kleiner als der Schwellenwert θ, so wird diese Verbindung gelöscht. Auf diese Weise wird es dem Netz ermöglicht, die Topologie des Eingaberaumes mit einer dynamischen Verbindungsstruktur zu lernen, ohne daß nach jedem Trainingsschritt alle Verbindungen gelöscht und aufgrund einer Trainingsmenge neu erstellt werden müssten.

Lernen der Ausgaben

Die Ausgabevektoren der Neuronen werden, ebenso wie die Zentren, schrittweise verbessert nach der Lernregel

$$\forall i \in V : \mathbf{o_i^{neu}} = \mathbf{o_i^{alt}} + a_i(\mathbf{x^t})\epsilon_{out}(t)(\mathbf{y^t} - \mathbf{o_{net}}(\mathbf{x^t})), \tag{3.30}$$

wobei $0 \leq \epsilon_{out}^{min} \leq \epsilon_{out}(t) \leq \epsilon_{out}^{max} \leq 1$ wieder eine mit der Zeit abnehmende Lernschrittweite bezeichnet, die nach

$$\epsilon_{out}(t) = \epsilon_{out}^{max} \left(\frac{\epsilon_{out}^{min}}{\epsilon_{out}^{max}} \right)^{\frac{t}{t_{max}}} \tag{3.31}$$

berechnet wird.

Falls die Anzahl der durchgeführten Lernschritte t kleiner ist als die maximale Anzahl t_{max}, wird ein weiterer Trainingsschritt durchgeführt, indem die Topologie und die Ausgaben wieder neu adaptiert werden.
Abbildung 3.7 zeigt einen DCS *online*-Trainingsschritt. Abbildung 3.8 faßt das gesamte *online*-Training zusammen.

3.3 Das LDCS-Netz

Jedem Neuron des DCS-Netzes war bisher nur ein Ausgabevektor zugeordnet. Wenn die Eingaben zwischen den Zentren der Neuronen liegen, müssen die Funktionswerte der unbekannten Funktion F auf den als Stützstellen fungierenden Neuronenzentren basierend approximiert werden. Die Approximation wird verbessert, indem auch die partiellen Ableitungen von F an der Stelle der Zentren berechnet werden.
Deshalb wird jedem Neuron i zusätzlich zu dem Ausgabevektor $\mathbf{o_i}$ eine Matrix $\mathbf{J_i}$ zugeordnet, die Jacobi-Matrix zu F an der Stelle $\mathbf{w_i}$. Diese Matrizen beschreiben den Zusammenhang zwischen kleinen Änderungen in den Eingabedaten in der Nähe der Zentren und den entsprechenden Änderungen in den Ausgabedaten. Dabei wird vorausgesetzt, daß F lokal in der Nähe der Neuronenzentren linear approximierbar ist. Ist die zu approximierende Funktion $F : \mathcal{R}^m \rightarrow \mathcal{R}^n$, so ist $\mathbf{J_i}$ eine $n \times m$-Matrix.
Die Berechnung der Netzausgabe für die Eingabe $\mathbf{x}$ nach (3.6) wird erweitert zu

$$\mathbf{o_{net}}(\mathbf{x}) = \sum_{i \in V} a_i(\mathbf{x}) \left(\mathbf{o_i} + \mathbf{J_i}(\mathbf{x} - \mathbf{w_i}) \right), \tag{3.32}$$

wobei V die Menge der Neuronen bezeichnet.
Das Training des LDCS-Netzes läuft analog zum Training des DCS-Netzes, nur die Berechnung der Ausgaben wird ergänzt mit der Berechnung der Jacobi-Matrizen.

```
/* DCS online – Trainingsschritt */
/* Bezeichnungen: */
/*   k gibt die Anzahl der Neuronen im Netz an, */
/*   V = {n_1, ..., n_k} ist die Menge der Neuronen im Netz, */
/*   t ist die Anzahl der bisher durchgefuehrten Lernschritte. */
/*   new_trainingsdata() liest ein neues Trainingsdatum ein. */
proc online_training_step()
begin
    t := t + 1;
    (x^t, y^t) := new_trainingsdata();
    for i := 1 to k do
        if (n_i = bmu(x^t)) then
            w_{n_i} := w_{n_i} + a_{n_i}(x^t) ε_{bmu}(t)(x^t − w_{n_i})
        else
            w_{n_i} := w_{n_i} + a_{n_i}(x^t) ε_{Nh}(t)(x^t − w_{n_i});
    g_{bmu(x^t),second(x^t)} := 1;
    g_{second(x^t),bmu(x^t)} := 1;
    for i := 1 to k do
        for j := 1 to k do
        begin
            g_{n_i,n_j} := α g_{n_i,n_j};
            if (g_{n_i,n_j} ≤ θ) then g_{n_i,n_j} := 0;
        end
    for i := 1 to k do
    begin
        o_{n_i} := o_{n_i} + a_{n_i}(x^t) ε_{out}(t)(y^t − o_{net}(x^t));
    end
end
```

Abbildung 3.7: Algorithmus eines *online* DCS Trainingsschrittes.

```
/* DCS online – Training */
/*   t_max bezeichnet die maximale Anzahl der Lernschritte. */
proc DCS − online()
begin
    online_initialize();
    for i := 1 to t_max do
    begin
        online_training_step();
    end
end
```

Abbildung 3.8: Trainingsalgorithmus eines *online* DCS-Netzes.

3.3.1 Das *offline*-Training des LDCS-Netzes

Das Lernen der Topologie geschieht wie im Abschnitt 3.2.2 beschrieben.
Die Berechnung der Ausgaben und der Jacobi-Matrizen der Neuronen ist ein zu (3.25) ähnliches, aber etwas komplexeres Ausgleichsproblem.
Für jeden Trainingspunkt $(\mathbf{x_k}, \mathbf{y_k}) \in Train$, $k = 1, \cdots, |Train|$ soll gelten:

$$\begin{aligned}
\mathbf{y_k} &= \sum_{i \in V} a_i(\mathbf{x_k})\left(\mathbf{o_i} + \mathbf{J_i}(\mathbf{x_k} - \mathbf{w_i})\right) \\
&= \sum_{i \in V} \left(a_i(\mathbf{x_k})\mathbf{J_i}(\mathbf{x_k} - \mathbf{w_i}) + a_i(\mathbf{x_k})\mathbf{o_i}\right) \\
&= \sum_{i \in V} (\mathbf{J_i}|\ \mathbf{o_i}) \begin{pmatrix} a_i(\mathbf{x_k})(\mathbf{x_k} - \mathbf{w_i}) \\ a_i(\mathbf{x_k}) \end{pmatrix} \\
&= (\mathbf{J_1}|\ \mathbf{o_1}|\ \cdots |\ \mathbf{J_N}|\ \mathbf{o_N}) \begin{pmatrix} a_1(\mathbf{x_k})(\mathbf{x_k} - \mathbf{w_1}) \\ a_1(\mathbf{x_k}) \\ \vdots \\ a_N(\mathbf{x_k})(\mathbf{x_k} - \mathbf{w_N}) \\ a_N(\mathbf{x_k}) \end{pmatrix},
\end{aligned} \tag{3.33}$$

wobei N die Anzahl der Neuronen im Netz bezeichnet.
Fassen wir die Gleichungen für alle Trainingsdaten zusammen, so ergibt sich

$$\begin{aligned}
&(\mathbf{y_1}|\ \cdots |\ \mathbf{y_t}) = \\
&(\mathbf{J_1}|\ \mathbf{o_1}|\ \cdots |\ \mathbf{J_N}|\ \mathbf{o_N}) \begin{pmatrix} a_1(\mathbf{x_1})(\mathbf{x_1} - \mathbf{w_1}) & \cdots & a_1(\mathbf{x_t})(\mathbf{x_t} - \mathbf{w_1}) \\ a_1(\mathbf{x_1}) & \cdots & a_1(\mathbf{x_t}) \\ \vdots & & \vdots \\ a_N(\mathbf{x_1})(\mathbf{x_1} - \mathbf{w_N}) & \cdots & a_N(\mathbf{x_t})(\mathbf{x_t} - \mathbf{w_N}) \\ a_N(\mathbf{x_1}) & \cdots & a_N(\mathbf{x_t}) \end{pmatrix},
\end{aligned} \tag{3.34}$$

wobei $t = |Train|$ die Anzahl der Trainingsdaten bezeichnet.
Durch Transponieren beider Seiten der Gleichung erhält man

$$\begin{aligned}
&\begin{pmatrix} \mathbf{y_1^T} \\ \vdots \\ \mathbf{y_t^T} \end{pmatrix} = \\
&\begin{pmatrix} a_1(\mathbf{x_1})(\mathbf{x_1} - \mathbf{w_1})^\mathbf{T} & a_1(\mathbf{x_1}) & \cdots & a_N(\mathbf{x_1})(\mathbf{x_1} - \mathbf{w_N})^\mathbf{T} & a_N(\mathbf{x_1}) \\ \vdots & \vdots & & \vdots & \vdots \\ a_1(\mathbf{x_t})(\mathbf{x_t} - \mathbf{w_1})^\mathbf{T} & a_1(\mathbf{x_t}) & \cdots & a_N(\mathbf{x_t})(\mathbf{x_t} - \mathbf{w_N})^\mathbf{T} & a_N(\mathbf{x_t}) \end{pmatrix} \begin{pmatrix} \mathbf{J_1^T} \\ \mathbf{o_1^T} \\ \vdots \\ \mathbf{J_N^T} \\ \mathbf{o_N^T} \end{pmatrix}.
\end{aligned} \tag{3.35}$$

Die komponentenweise Aufteilung des oberen Gleichungssystems führt zu einem linearen Ausgleichsproblem.

Wie die Testergebnisse in Kapitel 4 zeigen, werden die Netzausgaben durch die lineare

Erweiterung eines Netzes stark verbessert, d.h. die Funktion F kann besser approximiert werden.
Wenn die Trainingsdaten verrauscht sind, wird der Rauschanteil mitgelernt. Enthält das Netz genug Neuronen, um die Trainingsdaten auswendig zu lernen, so wird es auf den Eingabevektoren der Testmenge, die nicht den gleichen Rauschanteil wie die Trainingsdaten haben, falsche Antworten liefern. Das Training, d.h. das Einfügen neuer Knoten, sollte noch vor dem Auswendiglernen der Trainingsdaten beendet werden.
Statt des Schwellenwert-Abbruchskriteriums des *offline*-Trainings mit verrauschten Daten kann auch die Minimierung des Testfehlers als Abbruchsbedingung definiert werden. Der Verlauf des durchschnittlichen Fehlers auf der Trainings- bzw. Testmenge beim *offline*-Training eines LDCS-Netzes mit verrauschten Trainings- und Testdaten ist in Abbildung 3.9 dargestellt. Die Eingabevektoren der Trainings- und Testdaten sind Merkmalsvektoren von Bildern eines Kopfes, die mit den in Kapitel 2 beschriebenen Verfahren erzeugt wurden. Die Ausgabevektoren sind die zugehörigen Kopforientierungen mit zwei Freiheitsgraden (horizontales Drehen, vertikales Kippen). Die Trainingsmenge enthält 500, die Testmenge 100 Datensätze.
Aus der Abbildung ist zu entnehmen, daß das Netz beim fortschreitenden Training den durchschnittlichen Fehler auf der Trainingsmenge immer weiter minimiert, d.h. die Trainingsdaten werden immer genauer gelernt. Der durchschnittliche Testfehler verhält sich aber anders: bis zu einer bestimmten Anzahl von Trainingsschritten (d.h. Anzahl von Knoten im Netz) liegt er meistens über dem Trainingsfehler, nimmt aber im Mittel mit jedem Trainingsschritt ab. Ab einen bestimmten Zeitpunkt (d.h. Knotenanzahl), zu dem ein Minimum der Testfehler erreicht wird, nimmt der Testfehler im Mittel mit jedem weiterem Trainingsschritt wieder zu.
Die Erklärung für diese Fehlerverläufe liegt in den verrauschten Eingaben. Wenn die Daten in der Trainingsmenge zu genau gelernt werden, liefert das Netz auf Eingaben mit einem anderen Rauschanteil falsche Antworten.

3.3.2 Das *online*-Training des LDCS-Netzes

Die Entwicklung der Topologie ist im Abschnitt 3.2.3 beschrieben.
Für die Ausgabenberechnung wird die Lernregel für die Ausgabenadaption (3.30) ergänzt mit der Regel für das Lernen der Jacobi-Matrizen:

$$\forall i \in V : \mathbf{J_i^{neu}} = \mathbf{J_i^{alt}} + a_i(\mathbf{x^t})\epsilon_{jac}(t)\frac{1}{||\mathbf{x^t}-\mathbf{w_i}||_2^2}\left(\mathbf{y^t}-\mathbf{o_{net}}(\mathbf{x^t})\right)\left(\mathbf{x^t}-\mathbf{w_i}\right)^T, \qquad (3.36)$$

wobei $0 \leq \epsilon_{jac}^{min} \leq \epsilon_{jac}(t) \leq \epsilon_{jac}^{max} \leq 1$ wieder eine mit der Zeit abnehmende Lernkonstante ist, die nach

$$\epsilon_{jac}(t) = \epsilon_{jac}^{max}\left(\frac{\epsilon_{jac}^{min}}{\epsilon_{jac}^{max}}\right)^{\frac{t}{t_{max}}} \qquad (3.37)$$

berechnet wird.

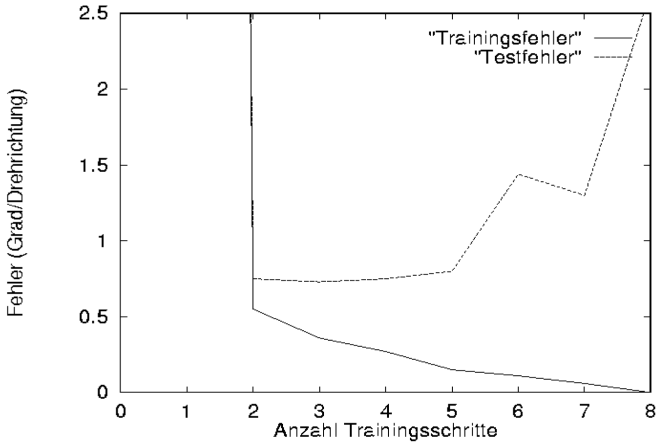


Abbildung 3.9: Der Verlauf des durchschnittlichen Trainings- bzw. Testfehlers eines *offline*-trainierten LDCS-Netzes. Als Eingaben dienen die nach Kapitel 2 bestimmten Merkmalsvektoren von Bildern eines Puppenkopfes. Als Ausgabe sollte die Kopforientierung mit zwei Freiheitsgraden (horizontales Drehen, vertikales Kippen) berechnet werden. Der durchschnittliche Fehler ist in Grad/Drehrichtung angegeben.

3.4 Das SDCS- und das LSDCS-Netz

Das SDCS- (Subspace Dynamic Cell Structure) und das LSDCS-Netz sind dem DCS- und dem LDCS-Netz ähnliche Netzkonstrukte. Die Approximation geschieht aber nicht im $\mathcal{R}^m$, wie bei den DCS- und LDCS-Netzen, sondern in einem niederdimensionalen Unterraum. Bei hochdimensionalen Eingaben führt dies zu einer deutlichen Effizienzverbesserung. Weiterhin ermöglicht die Projektion der Eingabedaten auf den Unterraum das Herausfiltern von Rauschen.

3.4.1 Die intrinsische Dimensionalität des Eingaberaumes

In den meisten praktischen Anwendungen hängt die Eingabedimension der Trainings- und Testdaten und damit die Dimension des Definitionsbereiches der zu approximierenden Funktion $F : \mathcal{R}^m \to \mathcal{R}^n$ nur von den Methoden der Trainings- und Testdatenberechnung ab. Zum Beispiel hängt die Dimension der nach Kapitel 2 erzeugten Merkmalsvektoren eines Bildes von der Anzahl der verwendeten Gaborfilter, von der Anzahl der Aufpunkte der Faltung, von der Anzahl der verschiedenen Orientierungen der Gaborfilter und von dem Bildtyp (Graubild oder Farbbild) ab. Die Dimensionalität der Eingabevektoren ist also unabhängig von der intrinsischen Dimensionalität des Eingaberaumes, die meistens viel kleiner ist als die Eingabedimension. Die intrinsische Dimensionalität einer Mannigfaltigkeit ist die minimale Anzahl der Freiheitsgrade, die nötig sind, um diese zu generieren [BS95].
Um die Berechnungen in hochdimensionalen Räumen zu umgehen, können die Eingabe-

daten auf die intrinsische Untermannigfaltigkeit des Eingaberaumes projiziert und die Berechnungen dort durchgeführt werden. Dadurch orientiert sich die Komplexität des Netzes statt an der hohen Eingabedimensionalität an der niedrigen intrinsichen Dimensionalität des Eingaberaumes.
Für die allgemeine Bestimmung der intrinsischen Dimensionalität von Mannigfaltigkeiten und für die Berechnung einer Basis der intrinsischen Untermannigfaltigkeit steht uns aber leider kein Algorithmus zur Verfügung. Jedoch kann die gelernte Topologie des LDCS-Netzes genutzt werden, um auf die Topologie des Eingaberaumes zu schließen. In dem Einzugsbereich jedes Neurons werden lokale Unterräume bestimmt, auf denen die in diesen Einzugsbereich fallenden Eingabevektoren projiziert werden.
Durch ein LDCS-Netz, bei dem die Zentren der Neuronen nach dem *K-means*-Verfahren bestimmt werden und dessen Verbindungsstruktur an den Eingabevektoren der Trainingsmenge orientiert nach (3.22) berechnet wird, kann die Topologie des Eingaberaumes erfaßt werden. Die Netztopologie ist die Grundlage zur Bestimmung der lokalen Unterräume des Eingaberaumes in dem Einzugsbereich jedes Neurons.
Hierzu wird für jedes Neuron i die Korrelationsmatrix $\mathbf{C_i}$ des Neurons bezüglich seiner Nachbarn bestimmt [BS95]. Wird die Nachbarschaft von Neuron i mit $Nh(i) = \{j_1, \cdots, j_k\}$ bezeichnet, so gilt:

$$\mathbf{C_i} = \frac{1}{|Nh(i)|}\mathbf{W^T W} \quad \text{mit} \quad \mathbf{W} = \begin{pmatrix} (\mathbf{w_{j_1}} - \mathbf{w_i})^\mathbf{T} \\ \cdots \\ (\mathbf{w_{j_k}} - \mathbf{w_i})^\mathbf{T} \end{pmatrix}. \tag{3.38}$$

Die Eigenwerte $\mu_1, \cdots, \mu_l$ und die Eigenvektoren $\mathbf{e_1}, \cdots, \mathbf{e_l}$ von $\mathbf{C_i}$ lassen sich durch Hauptachsenanalyse bestimmen. Eine geeignete Methode ist das Jacobi-Verfahren [Pre92].
Die Eigenvektoren der Korrelationsmatrix $\mathbf{C_i}$ spannen einen lokalen Unterraum $\mathbf{U_i}$ des Eingaberaumes auf, dessen Dimensionalität der lokalen intrinsischen Dimensionalität des Eingaberaumes in der Nähe des Neurons i entspricht.
Ist die Dimension des Eingaberaumes m, so ist $\mathbf{C_i}$ eine $m \times m$-Matrix. Bei einer hohen Eingabedimensionalität m ist die Berechnung der Eigenwerte und Eigenvektoren von $\mathbf{C_i}$ sehr berechnungsintensiv. Es gilt aber, daß $\mathbf{C_i}$ und

$$\tilde{\mathbf{C}}_\mathbf{i} = \frac{1}{|Nh(i)|}\mathbf{W W^T} \tag{3.39}$$

die gleichen Eigenwerte $\mu_1, \cdots, \mu_l$ haben, weshalb sich die Eigenvektoren $\mathbf{e_1}, \cdots, \mathbf{e_l}$ von $\mathbf{C_i}$ aus den Eigenvektoren $\tilde{\mathbf{e}}_\mathbf{1}, \cdots, \tilde{\mathbf{e}}_\mathbf{l}$ von $\tilde{\mathbf{C}}_\mathbf{i}$ durch

$$\forall j \in \{1, \cdots, l\} : \mathbf{e_j} = \frac{\mathbf{W^T}\tilde{\mathbf{e}}_\mathbf{j}}{\sqrt{\mu_j |Nh(i)|}} \tag{3.40}$$

ableiten lassen [MN95].
Da $\tilde{\mathbf{C}}_\mathbf{i}$ nur $|Nh(i)| \times |Nh(i)|$-dimensional ist, ist die Bestimmung der Eigenwerte und Eigenvektoren von $\tilde{\mathbf{C}}_\mathbf{i}$ mittels Hauptachsenanalyse und die anschließende Berechnung der Eigenwerte und Eigenvektoren von $\mathbf{C_i}$ nach (3.40) effizienter.

Die Größe der Eigenwerte läßt auf die Varianz der Daten in der Richtung der zugehörigen Eigenvektoren schließen. Hat ein Eigenvektor einen kleinen Eigenwert (relativ zu den anderen), so ist die Varianz der Daten in diese Richtung klein und vermutlich nur durch Rauschen verursacht. Solche Eigenvektoren sollen auf die Bestimmung der lokalen intrinsischen Unterräume keinen Einfluß haben. Der lokale Unterraum $\mathbf{U_i}$ in der Umgebung eines Neurons i soll durch diejenigen Eigenvektoren der Korrelationsmatrix $\mathbf{C_i}$ aufgespannt werden, deren Eigenwerte größer sind als ein bestimmter Bruchteil des größten Eigenwertes:

$$\mathbf{U_i} = < \mathbf{u} >_{\mathbf{u} \in H} \;\; mit \; H = \{\mathbf{e_j} | 1 \leq j \leq l \;\wedge\; \mu_j \geq \epsilon \max\{\mu_1, \cdots, \mu_l\}\}, \tag{3.41}$$

wobei $0 \leq \epsilon \leq 1$ den Schwellenwertfaktor angibt.
Jeder Eingabevektor wird nun auf die lokalen niedrigdimensionalen Unterräume der Neuronen projiziert, um die Berechnung der Netzausgabe und das Lernen effizienter durchführen zu können. Außer der Effizienzverbesserung hat dieses Verfahren aber auch eine rauschvermindernde Wirkung. Ist ein Eingabedatum verrauscht, so liegt es außerhalb des intrinsischen Unterraumes. Durch Projektion auf den Unterraum wird der Rauschanteil herausgefiltert.
Im folgenden bezeichnet

$$\mathbf{E_i} = \begin{pmatrix} \mathbf{e_{j_1}^T} \\ \vdots \\ \mathbf{e_{j_k}^T} \end{pmatrix} \tag{3.42}$$

die Matrix der normierten Eigenvektoren der Korrelationsmatrix $\mathbf{C_i}$ des Neurons i, deren zugehörige Eigenwerte größer sind als der nach (3.41) definierte Schwellenwert.

Das SDCS- und das LSDCS-Netz wurden aus dem DCS- und dem LDCS-Netz entwickelt. Der Unterschied besteht in der Projektion der Eingabedaten auf die lokalen intrinsischen Unterräume. Bei Netzen mit einfachen Ausgaben ohne lineare Erweiterung (DCS-Netz) führt die Projektion der Eingabedaten auf die lokalen intrinsischen Unterräume lediglich zu neuen Aktivierungsfunktionen (SDCS-Netz). Bei dem LSDCS-Netz werden die Jacobi-Matrizen, also die partiellen Ableitungen der zu approximierenden Funktion durch die Projektion nicht mehr in $\mathcal{R}^m$, sondern in den lokalen intrinsischen Unterräumen berechnet.
Beim LDCS-Netz hatten die Jacobi-Matrizen die Dimension $n \times m$, wobei m die Dimension der Eingabevektoren und n die Dimension der Ausgabevektoren angibt.
Ist im Einzugsbereich des Neurons i die Dimensionalität des lokalen intrinsischen Unterraumes k, so hat die Jacobi-Matrix beim LSDCS-Netz die Dimension $n \times k$, wobei i.a. $k << m$ gilt.
Es sollte beachtet werden, daß diese Methode zur Bestimmung lokaler intrinsischer Unterräume nur dann sinnvoll ist, wenn die intrinsische Dimensionalität kleiner oder gleich der Anzahl der Nachbarn der Neuronen ist.

3.4.2 Die Aktivierungsfunktion

Die Aktivierungen werden nicht mehr nach den Entfernungen in $\mathcal{R}^m$, sondern nach den Abständen in den lokalen intrinsischen Unterräumen bestimmt. Dazu werden sowohl die Neuronenzentren als auch die Eingabevektoren auf die lokalen Unterräume in der Nähe der einzelnen Neuronen projiziert.
Die Aktivierungsfunktion des Neurons i wird dadurch verändert zu

$$a_i(\mathbf{x}) = \begin{cases} \exp\left(-\frac{||\mathbf{E_i w_i} - \mathbf{E_i x}||_2^2}{\sigma_i^2} \cdot (-\ln p)\right) \cdot \frac{1}{sumAct(\mathbf{x})} & : \; i \in \{bmu(\mathbf{x})\} \cup Nh(bmu(\mathbf{x})) \\ 0 & : \; sonst \end{cases} \tag{3.43}$$

wobei

$$sumAct(\mathbf{x}) = \sum_{l \in \{bmu(\mathbf{x})\} \cup Nh(bmu(\mathbf{x}))} \exp\left(-\frac{||\mathbf{E_l w_l} - \mathbf{E_l x}||_2^2}{\sigma_l^2} \cdot (-\ln p)\right) \tag{3.44}$$

die Summe der unnormierten Aktivierungen bezeichnet und

$$\sigma_i^2 = \frac{\sum_{j \in Nh(i)} ||\mathbf{E_i w_i} - \mathbf{E_i w_j}||_2^2}{|Nh(i)|} \tag{3.45}$$

ist.
Sollte die Ausgabe nur aus den Werten der *best matching unit* berechnet werden, so vereinfacht sich die Aktivierungsfunktion wieder zu

$$a_i(\mathbf{x}) = \begin{cases} 1 & : \; i = bmu(\mathbf{x}) \\ 0 & : \; sonst \end{cases} \quad . \tag{3.46}$$

3.4.3 Die Netzausgabe

Für die Berechnung der Netzausgabe werden die Zentren und die Eingabevektoren wieder auf die lokalen Unterräume projiziert. Damit ergibt sich

$$\mathbf{o_{net}}(\mathbf{x}) = \sum_{i \in V} a_i(\mathbf{x}) \left(\mathbf{o_i} + \mathbf{J_i}(\mathbf{E_i x} - \mathbf{E_i w_i})\right), \tag{3.47}$$

wobei V die Menge der Neuronen bezeichnet.

Das Training des SDCS-Netzes ist bis auf die Aktivierungsfunktionen wie beim DCS-Netz. Das Training des LSDCS-Netzes ist dem des LDCS-Netzes ählich. Das Lernen der Topologie bleibt unverändert, nur das Lernen der Ausgaben wird duch die Projektion auf die lokalen intrinsischen Unterräume verändert.

3.4.4 Das *offline*-Training des LSDCS-Netzes

Das Lernen der Topologie geschieht wie im Abschnitt 3.2.2 beschrieben.
Für die Berechnung der Ausgaben des LSDCS-Netzes ergibt sich ein ähnliches Gleichungssystem wie in 3.3.1, nur die Differenzen zwischen den Eingabevektoren und den Neuronenzentren werden statt in $\mathcal{R}^m$ nun in den lokalen Unterräumen in der Umgebung der betroffenen Neuronen berechnet, d.h. jedes Vorkommen von $\mathbf{x_i} - \mathbf{w_j}$ wird ersetzt durch $\mathbf{E_j x_i} - \mathbf{E_j w_j}$, wobei $\mathbf{x_i}$ den Eingabevektor des i-ten Trainingsdatums und $\mathbf{w_j}$ das Zentrum von Neuron j bezeichnet.

3.4.5 Das *online*-Training des LSDCS-Netzes

Auch beim *online*-Training ist das Rechnen in lokalen intrinsischen Unterräumen sinnvoll. Die Änderung der Netzstruktur während des Lernens kann zu einer Dimensionsänderung der lokalen intrinsischen Unterräume und damit der Jacobi-Matrizen führen. In diesem Fall müssen die Jacobi-Matrizen der betroffenen Neuronen neu berechnet werden. Eine Änderung der lokalen intrinsischen Dimensionalität kann aber nur solange geschehen, wie sich die Neuronenzentren noch nicht endgültig verteilt haben. Wenn im Laufe des Trainings die Lernschrittweiten ϵ_{bmu} und ϵ_{Nh} genügend klein geworden sind, bleiben die Zentren praktisch eingefroren und die Ausgaben und die Jacobi-Matrizen der Neuronen können richtig gelernt werden.
Das Lernen selbst geschieht ähnlich wie beim *online* LDCS-Netz. Der einzige Unterschied besteht in der Projektion der Eingaben auf die lokalen intrinsischen Unterräme für die Berechnung der Aktivierungen und für die Lernregel der Jacobi-Matrizen.
Das Lernen geschieht wieder schrittweise in t_{max} Schritten.
In jedem Lernschritt wird erst die Netztopologie verbessert, d.h. die Neuronenzentren der aktivierten Neuronen werden verschoben und die Verbindungsstruktur wird adaptiert. Bis auf die Aktivierungsfunktionen geschieht dies wie beim *online* DCS- und LDCS-Netz (s. Kapitel 3.2.3). Anschließend werden die Dimension und eine neue Basis der aktuellen lokalen intrinsischen Unterräumen der aktivierten Neuronen, ihrer Nachbarn und derjenigen Neuronen, zwischen denen eine Verbindung erstellt oder gelöscht wurde, berechnet, wie in Kapitel 3.4.1 beschrieben. Hat sich die intrinsische Dimensionalität in einem der lokalen Bereiche geändert, so wird die Jacobi-Matrix des betroffenen Neurons neu berechnet.
Danach werden die Ausgabevektoren und die Ableitungsmatrizen der Neuronen gelernt. Die Lernregel für die Ausgabevektoren bleibt wie beim *online* DCS- und LDCS-Netz[3] (s. Kapitel 3.2.3), nur die Lernregel für die Jacobi-Matrizen ändert sich. Hier werden die Differenzvektoren wieder in den lokalen Unterräumen gebildet:

$$\forall i \in V : \mathbf{J_i^{neu}} = \mathbf{J_i^{alt}} + a_i(\mathbf{x})\epsilon_{jac}(t)\frac{1}{||\mathbf{E_i x^t} - \mathbf{E_i w_i}||_2^2}\left(\mathbf{y^t} - \mathbf{o_{net}}(\mathbf{x^t})\right)\left(\mathbf{E_i x^t} - \mathbf{E_i w_i}\right)^T, \tag{3.48}$$

wobei $(\mathbf{x^t}, \mathbf{y^t})$ den Trainingspunkt beim Lernschritt t, $\mathbf{w_i}$ das Zentrum des Neurons i und V die Menge der Neuronen bezeichnet. Es ist $0 \leq \epsilon_{jac}^{min} \leq \epsilon_{jac}(t) \leq \epsilon_{jac}^{max} \leq 1$ wieder eine

[3] bis auf die Aktivierungsfunktionen

mit der Zeit abnehmende Lernkonstante.

Kapitel 4

Testergebnisse

Nach den theoretischen Grundlagen wenden wir uns der Realisierung der in den vorherigen Kapiteln beschriebenen Bildverarbeitungsmethoden und neuronalen Netze zu.
Nach der Beschreibung der Hardware und des technischen Versuchsaufbaus im Abschnitt 4.1 wird die Software vorgestellt (Abschnitt 4.2). Abschnitt 4.3 enthält einige Hinweise zu den Versuchen. Im Abschnitt 4.4 werden die technische Durchführung und die Ergebnisse der Versuche beschrieben.
In den Versuchen werden die in Kapitel 3 eingeführten neuronalen Netze trainiert. Die Eingabevektoren der Trainings- und Testdaten sind Merkmalsvektoren von Bildern eines Puppenkopfes, die nach den in Kapitel 2 beschriebenen Bildverarbeitungsmethoden berechnet werden. Die Ausgabevektoren sind die zugehörigen zweidimensionalen Orientierungen.

4.1 Der Versuchsaufbau

Die Bildaufnahmen werden durchgeführt mit Hilfe eines Roboterarmes und einer Farbkamera. Beide sind mit einem SUN Arbeitsrechner verbunden. Die Versuchsanordnung ist in Abbildung 4.1 dargestellt.

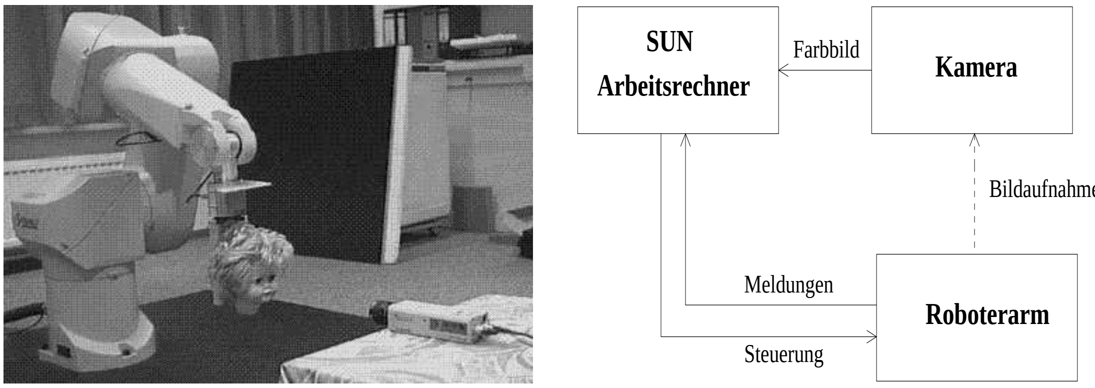


Abbildung 4.1: Die Versuchsanordnung der Experimente: Ein Puppenkopf wird von einem Roboterarm gehalten und in eine vorgegebene Orientierung gedreht. Das Bild wird von einer Farbkamera aufgenommen.

4.1.1 Der Roboterarm

In den Versuchen wird der Roboterarm RX90 der Firma Stäubli verwendet. Der Roboterarm ist am Boden montiert und besteht aus 6 Segmenten, die miteinander durch Gelenke verbunden sind. Jedes Gelenk ermöglicht eine Drehung in eine bestimmte Richtung. Die redundanten Freiheitsgrade des Roboters ermöglichen die Positionierung des Manipulators im Arbeitsbereich mit verschiedenen Orientierungen. Am Ende des Roboterarmes ist ein Greifer mit 2 Fingern befestigt.
Da der Roboterarm 6 Freiheitsgrade hat, werden die Raumpunkte durch 6 Parameter X, Y, Z, yaw, $pitch$ und $roll$ beschrieben. X, Y und Z werden in mm ausgedrückt und geben die Entfernung zwischen dem Mittelpunkt des Werkzeugflanschs und dem Roboterfuß an. Yaw, $pitch$ und $roll$ werden in Grad ausgedrückt und beschreiben die Orientierung des Manipulators. In dieser Form ist die Robotersteuerung sehr einfach und bequem. (Die inverse Kinematik des Roboters, d.h. die Bestimmung der Gelenkwinkel des Roboters, die einer bestimmten Positionierung in der oben beschriebenen Form entspricht, ist im Softwarepaket des Roboters realisiert.)
Die Steuerung des Roboterarmes geschieht über einen Steuerschrank. Der Steuerschrank des Roboters ist mit einem SUN Arbeitsrechner doppelt verbunden. An diesem Arbeitsrechner kann über die erste Schnittstelle das Terminal des Steuerschrankes emuliert werden. Eine zweite Verbindung zwischen dem Arbeitsrechner und dem Steuerschrank dient zur Übermittlung der Robotersteuerungsbefehle und eventueller Fehlerstatusmeldungen.
Die maximale Reichweite des Roboters (die maximale Entfernung zwischen dem Mittelpunkt des Werkzeugflanschs und dem Roboterfuß) ist 900 mm, die minimale Reichweite ist 290 mm.
Für die Bildaufnahmen hält der Roboter einen Puppenkopf fest und dreht diesen in die angegebene Orientierung. Dabei bleibt der Kopf an der gleichen Position, nur seine Orientierung ändert sich.

4.1.2 Die Kamera

Zur Bildaufnahme verwenden wir eine SONY Farbkamera mit einem Objektiv der Brennweite 12.5 mm. Die Kamera ist über eine Framegrabberkarte mit dem SUN Arbeitsrechner verbunden.
Die Kamera steht ca. 50 cm vor dem Robotermanipulator und ist auf den Puppenkopf gerichtet.
Die maximale Auflösung der Kamerabilder ist 576×768. Diese werden vor der Bildverarbeitung mittels bilinearer Interpolation auf die Hälfte skaliert. Somit liegen Bilder mit einer Größe von 288×384 Pixel vor.

4.2 Software

Die benutzte Software unterteilt sich in zwei Pakete. Das erste Softwarepaket ist speziell für diese Anwendung entwickelt worden und ermöglicht die Bildaufnahme und die Merk-

malsextraktion aus den Bildern, d.h. die Erzeugung der Trainings- und Testdaten der neuronalen Netze.
Das zweite Paket ist eine Erweiterung eines von J. Bruske und J. Troester entwickelten Programmes, das der Realisierung und dem Training der neuronalen Netze dient.

4.2.1 Software der Bildverarbeitung

Das erste Programmpaket bietet Funktionen zur Bildaufnahme und Bildverarbeitung an. Um Bilder von einem Puppenkopf aufnehmen zu können, der durch einen Roboterarm gedreht wird, muß das Programm sowohl mit der Kamera als auch mit dem Roboterarm kommunizieren. Die Robotersteuerung und die Kamerabildaufnahme geschehen menügesteuert. Zur Verarbeitung der Kamerabilddaten wurden die Funktionen der XIL-Bibliothek ergänzt. Außer der technischen Realisierung der in Kapitel 2 vorgestellten Bildverarbeitungmethoden wurde eine graphische Oberfläche zur Anzeige von Kamerabildern, von Zwischenergebnissen der verschiedenen Bildverarbeitungsstufen, von Filtermasken und Ganzbildfaltungen eingebaut.
Weiterhin wird für die einfache und exakte Bestimmung der Farbschwellenwerte der Farbsegmentierung außer der Berechnung und Darstellung von Histogrammen eine andere praktische Hilfe angeboten: In einem Bild wird ein Bereich markiert, innerhalb dessen sich nur Pixel mit Gesichtsfarbe befinden. Die Farbschwellenwerte werden in allen 3 Farbbänden als das Minimum und das Maximum der Farbwerte im markierten Bereich bestimmt.
Bei der Merkmalsextraktion können die verschiedenen Bildverarbeitungsmethoden optional ausgewählt werden. In den Versuchen werden die Ergebnisse der unterschiedlichen Methoden miteinander verglichen, um dadurch ein optimiertes Verfahren zu bestimmen.
Die Erstellung der Trainings- bzw. Testmengen der neuronalen Netze geschieht in zwei Schritten: Erstens werden so viele Bilder aufgenommen, wie Daten benötigt werden. Danach werden aus den Bildern Merkmalsvektoren extrahiert, die zusammen mit den entsprechenden Kopforientierungen in einer Trainings- bzw. Testdatei gespeichert werden.

4.2.2 Realisierung der neuronalen Netze

Die verwendeten neuronalen Netze wurden unter einer eigenen graphischen Benutzeroberfläche realisiert. Unter dieser Oberfläche werden u.a. die Definition und Parametereinstellung (wie z.B. Netztyp, Aktivierungsfunktion, Fehlerfunktion, Lernfunktion, evtl. Vorinitialisierung, Bestimmung der Trainings- und der Testmenge usw.) und das Training von neuronalen Netzen ermöglicht. Die neuronalen Netze sind im Ein- oder Ausgaberaum graphisch darstellbar.
Die Netzstrukturen sind mit Hilfe der LEDA-Bibliothek programmiert.

4.3 Bemerkungen

Wie schon in Kapitel 2 erwähnt, ist es wichtig, daß die Lichtverhältnisse während der Bildaufnahmen möglichst konstant bleiben. In den Versuchen wird bei den Bildaufnahmen künstliches Licht mit konstanter Helligkeit verwendet.
Die Berechnung der Merkmalsvektoren geschieht nach den in Kapitel 2 eingeführten Methoden. Da dort für die einzelnen Schritte verschiedene Verfahren vorgestellt wurden, wird bei jedem Versuch genau angegeben, auf welche Weise die Trainings- und Testdaten erzeugt werden.
Es werden Experimente mit verschiedenen Netztypen und Netzparametern durchgeführt. Auch hier wird bei jedem Versuch das angewendete Netz, seine Parameter und die Lernmethode genau beschrieben.

4.4 Testergebnisse

4.4.1 Vergleich der verschiedenen Bildverarbeitungsmethoden

Um eine optimierte Merkmalsmenge zu bestimmen, werden die verschiedenen Bildverarbeitungsmethoden anhand mehrerer Experimente verglichen. Um möglichst gleiche Trainingskonditionen zu schaffen, werden die Trainings- und Testmengen der einzelnen Experimente immer aus der gleichen Bildsequenz berechnet. Für die Trainingsmenge stehen 500, für die Testmenge 100 Bilder des in Abbildung 2.1 dargestellten Puppenkopfes mit verschiedenen zufälligen Orientierungen zur Verfügung, wobei sich der horizontale Drehwinkel zwischen $-75°$ und $+75°$ und der vertikale Kippwinkel zwischen $-30°$ und $+30°$ bewegen kann. Die hohe Anzahl der Trainingsbilder ist notwendig, um das Auswendiglernen der Trainingsdaten zu vermeiden.
Soweit nicht anders vermerkt, werden bei der Merkmalsextraktion die folgende Methoden verwendet:
Die Farbsegmentierung geschieht mit den Schwellenwerten $\mathbf{s^{min}} = (40, 75, 130)$ und $\mathbf{s^{max}} = (255, 114, 170)$ (die Farbbänder sind in der Reihenfolge grün, blau und rot aufgelistet). Diese Werte wurden aufgrund der Pixelfarbwerte eines Gesichtsausschnittes bestimmt, wie in 4.2.1 beschrieben. Bei der Skalierung wird die schwerpunktbezogene Gesichtslokalisierung nach Gl.(2.4) angewendet. Der Gesichtsbereich wird dabei auf ein 80×80 Pixel großes Bild skaliert. Die Pixel im Hintergrund behalten ihre ursprünglichen Farbwerte. Nachdem die skalierte Gesichtsregion in ein Grauwertbild konvertiert wurde, wird sie an 3×3 Aufpunkten mit 4 Gaborfiltern, die sich nur in ihren Orientierungen unterscheiden, gefaltet ($\lambda_x = 40$, $\sigma_x = 10$ und $\theta = 0$, $\pi/4$, $2\pi/4$ und $3\pi/4$). Damit entstehen 36-dimensionale Eingabevektoren. Die Ausgabevektoren (Orientierung des Gesichtes) sind zweidimensional (horizontales Drehen, vertikales Kippen).
In diesem Abschnitt verwenden wir das LSDCS-Netz, sowohl mit *offline*- als auch mit *online*-Training. Das LSDCS-Netz mit *offline*-Training ist in Kapitel 3.4.4 beschrieben. Als Abbruchskriterium des Trainings wählen wir die Minimalität des Testfehlers, d.h. die

Anzahl der Neuronen im Netz ist bei den verschiedenen Versuchen nicht gleich (sie bewegt sich zwischen 18 und 42). Bei jeder Eingabe werden die *best matching unit* und ihre Nachbarn nach Gl.(3.43) aktiviert, wobei der Faktor p auf 0.1 gesetzt ist. Der Schwellenwert für die Eigenwerte bei der Berechnung in Gl.(3.41) ist auf $\epsilon = 0.01$ gesetzt.
Das LSDCS-Netz mit *online*-Training (Kapitel 3.4.5) wird mit 40 Neuronen initialisiert. Die Anzahl der maximalen Trainingsschritte beträgt $t_{max} = 15\,000$. Die Lernschrittweiten sind $\epsilon_{bmu}^{max} = 0.9$, $\epsilon_{bmu}^{min} = 0.001$, $\epsilon_{Nh}^{max} = 0.5$, $\epsilon_{Nh}^{min} = 0.001$, $\epsilon_{out}^{max} = 0.1$, $\epsilon_{out}^{min} = 0.1$, $\epsilon_{jac}^{max} = 0.1$ und $\epsilon_{jac}^{min} = 0.1$. Auch hier werden für eine Eingabe die *best matching unit* und ihre Nachbarn nach Gl.(3.43) mit $p = 0.1$ aktiviert. Die Vergessenskonstante in Gl.(3.29) ist $\alpha = 0.995$ und der Schwellenwert $\theta = 0.01$. Der Schwellenwert für die Eigenwerte bei der Berechnung in Gl.(3.41) ist, wie beim *offline*-Training, auf $\epsilon = 0.01$ gesetzt.

Exakte und schwerpunktbezogene Gesichtslokalisierung

In dem ersten Experiment werden die Auswirkungen der exakten und der schwerpunktbezogenen Gesichtslokalisierung auf die Genauigkeit der Orientierungsbestimmung verglichen. Aus der vorliegenden Bildsequenz werden Trainings- und Testdaten mit den beiden Methoden erzeugt und beide Netze mit beiden Trainingsmengen traininert. Tabelle 4.1 zeigt die Testergebnisse.

Gesichts-lokalisierung	LSDCS offline		LSDCS online	
	mittlerer Testfehler	maximaler Testfehler	mittlerer Testfehler	maximaler Testfehler
	(Grad/Drehrichtung)			
Exakt	1.55	5.39	2.36	9.31
Schwerpunktbezogen	0.87	2.78	1.77	7.78

Tabelle 4.1: Testergebnisse der Orientierungsbestimmung eines Puppengesichtes mit zwei LSDCS-Netzen (offline, online), wobei die Gesichtslokalisierung mit der exakten und mit der schwerpunktbezogenen Methode geschah.

Aus der Tabelle ist zu entnehmen, daß die Testfehler nach dem Training mit durch schwerpunktbezogene Gesichtslokalisierung erzeugten Daten bei beiden Netztypen kleiner sind als bei der exakten Lokalisierung. Dies läßt sich einerseits dadurch erklären, daß die schwerpunktbezogene Gesichtslokalisierung die Unregelmäßigkeiten der Farbsegmentierung an dem Gesichtsrand glättet, d.h. Unebenheiten am Gesichtsrand sind nicht so schwerwiegend wie bei der exakten Lokalisierung.
Andererseits ist die Größe des Objektes im skalierten Bild bei der exakten Lokalisierung nur dann konstant, wenn das Objekt exakt kreisförmig ist. Bei länglichen Objekten, wie z.B. einem Stab, hängt die skalierte Objektgröße von der maximalen horizontalen bzw. vertikalen Ausdehnung des Objektes im Originalbild ab, und nicht vom Objektradius, wie es bei der schwerpunktbezogenen Lokalisierung der Fall ist. Da die horizontale bzw. vertikale Objektausdehnung i.a. von der Orientierung des Objektes abhängig ist, ist der Skalierungsfaktor der exakten Lokalisierung nicht konstant. Abbildung 4.2 zeigt einen

Beispiel. Ein 200×40 Pixel großes (zweidimensionales) Rechteck wird in zwei verschiedenen Orientierungen ($\theta_1 = 0$ und $\theta_2 = \pi/4$) dargestellt. Mit der exakten Lokalisierung sind die Objektgrößen in den skalierten Bildern (80×80 Pixel) unterschiedlich, mit der schwerpunktbezogenen Lokalisierung konstant.

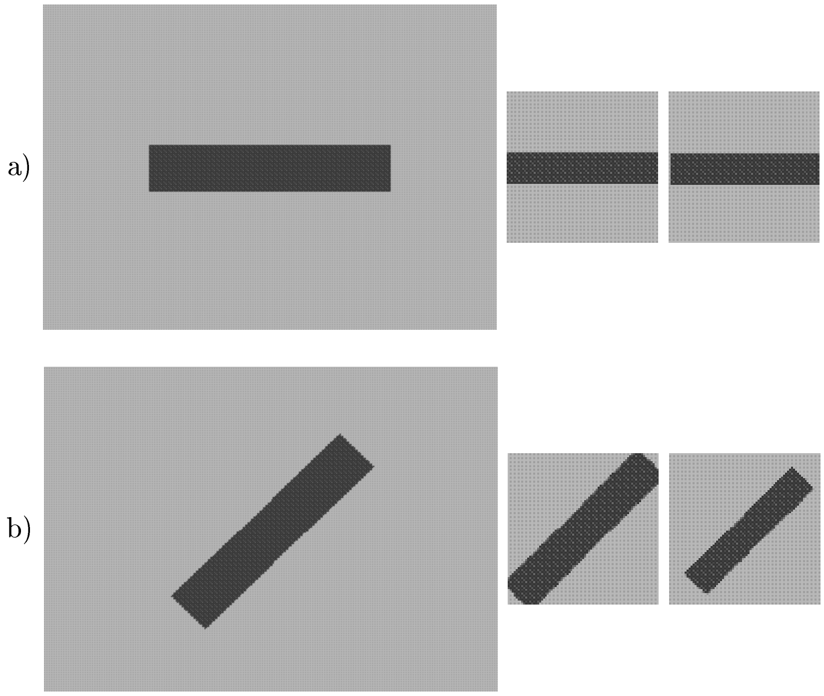


Abbildung 4.2: Die Ergebnisse der exakten und schwerpunktbezogenen Lokalisierung eines Rechtecks in zwei verschiedenen Orientierungen (a) $\theta_1 = 0$ und b) $\theta_2 = \pi/4$). Von links nach rechts werden das Originalbild, das Ergebnis der exakten und das Ergebnis der schwerpunktbezogenen Lokalisierung dargestellt.

Der Hintergrund

In Kapitel 2 sind vier verschiedene Möglichkeiten der Hintergrundbehandlung beschrieben. Mit allen vier Methoden werden Trainings- und Testdaten aus der Bildsequenz extrahiert und beide Netztypen mit allen vier Trainingsmengen trainiert. Die Testergebnisse sind in Tabelle 4.2 zusammengefaßt.

Die Helligkeitsunabhängigkeit eines konstanten Hintergrundwertes hat schon bei kleinen Helligkeitsschwankungen (z.B. Schatten) negative Auswirkungen. Wenn die Hintergrundpixel eine konstante Farbe haben und sich die Lichtverhältnisse ändern, ändern sich die Pixelwerte in der Gesichtsregion, nicht aber im Hintergrund. Die Faltung wird also mit teilweise helligkeitsinvarianten und teilweise helligkeitsabhängigen Pixelwerten durchgeführt. Dies führt zu größeren Testfehlern bei der Anwendung der Methode des konstanten Hintergrundes und der kreisförmig begrenzten Gesichtsregion (bei der der Hintergrund auch einen konstanten Farbwert hat).

Hintergrund	LSDCS offline		LSDCS online	
	mittlerer Testfehler	maximaler Testfehler	mittlerer Testfehler	maximaler Testfehler
	(Grad/Drehrichtung)			
Original	0.87	2.78	1.77	7.78
Gemittelt	1.21	3.70	2.16	10.53
Konstant	1.44	4.52	2.31	8.62
Im Kreis original, außen konstant	1.17	4.33	2.29	12.82

Tabelle 4.2: Testergebnisse der Orientierungsbestimmung eines Puppengesichtes mit zwei LSDCS-Netzen (offline, online), wobei die Hintergrundsgestaltung mit den vier in Kapitel 2 vorgestellten Methoden geschah.

Der Testfehler mit einem gemittelten Hintergrund ist auch nicht optimal. Die beste Approximation wird durch die originalen Pixelfarbwerte erreicht.
Es muß betont werden, daß der Hintergrund während der Aufnahmen unverändert bleibt. Bei einer Änderung des Hintergrundsmusters kann die Übernahme der Originalfarben der Hintergrundspixel zu schlechteren Ergebnissen führen als die anderen Methoden.

Anzahl der Aufpunkte bei der Faltung

In den vorigen Versuchen wurden die 80×80 Pixel großen Bilder mit Gaborfiltern der Breite $\lambda_x = 40$ an 3×3 gleichmäßig rasterförmig angeordneten Aufpunkten gefaltet.
Die Genauigkeit der Orientierungsbestimmung des Gesichtes hängt u.a. davon ab, ob aus den Bildern durch die Faltungen genügend Informationen extrahiert werden. Wenn die Anzahl der Aufpunkte erhöht wird, stehen den Netzen größere Merkmalsvektoren und damit mehr Information zur Verfügung. Es ist aber nicht selbstverständlich, daß zusätzliche Information immer vorteilhaft ist. Zum Beispiel kann der Rechenaufwand durch höherdimensionale Eingabevektoren deutlich erhöht werden. Auch die Wirkung des Rauschens, das in der Praxis in jedem Element der Merkmalsvektoren auftritt, ist schwer abzuschätzen.
Um die minimal notwendige Anzahl von Aufpunkten zu bestimmen, die für eine ausreichend genaue Orientierungsbestimmung notwendig ist, werden die beiden LSDCS-Netze mit Daten trainiert, die durch Faltung der Bilder an verschieden vielen Aufpunkten erzeugt wurden. Tabelle 4.3 zeigt die Testergebnisse.
Aus der Tabelle wird ersichtlich, daß die minimale Anzahl der nötigen Aufpunkte, an denen das 80×80 Bild mit Gaborfiltern ($\lambda_x = 40$) gefaltet werden muß, um genügend Informationen über die Kopforientierung zu erhalten, 4×4 ist. Dies bedeutet einen Abtastabstand von $\frac{\lambda_x}{2}$.
Eine weitere Erhöhung der Anzahl der Aufpunkte führt zu keiner deutlichen Verbesserung in der Orientierungsbestimmung.

Anzahl Aufpunkte	LSDCS offline		LSDCS online	
	mittlerer Testfehler	maximaler Testfehler	mittlerer Testfehler	maximaler Testfehler
	(Grad/Drehrichtung)			
1x1	12.17	68.78	15.40	64.87
2x2	1.40	3.97	4.73	31.71
3x3	0.87	2.78	1.77	7.78
4x4	0.64	1.88	1.26	4.42
5x5	0.64	1.91	1.42	4.94
6x6	0.61	1.74	1.33	4.03
7x7	0.67	2.31	1.61	7.83
8x8	0.58	1.82	1.16	3.72

Tabelle 4.3: Testergebnisse der Orientierungsbestimmung eines Puppengesichtes mit zwei LSDCS-Netzen (offline, online), wobei die Anzahl der Aufpunkte für die Faltung mit den Gaborfiltern ($\lambda_x = 40, \theta = 0, \frac{\pi}{4}, \frac{2\pi}{4}, \frac{3\pi}{4}$) variiert.

Der Durchlaßbereich der Gaborfilter

Nicht nur die Anzahl der Aufpunkte ist bei der Merkmalsextraktion von großer Bedeutung. Genauso wichtig ist es, Gaborfilter mit einem an die Gesichtseigenschaften angepaßten Durchlaßbereich zu verwenden.
In den nächsten Versuchen wird die Merkmalsextraktion aus den vorhandenen Bildern mittels Gaborfilter mit verschiedenen Durchlaßbereichen durchgeführt. Um den Abtastabstand auf die Gaborfilter anzupassen, wird die Faltung der 80×80 Pixel großen Bilder mit einem Gaborfilter der Breite λ_x an $\left[\frac{2\cdot 80}{\lambda_x}\right] \times \left[\frac{2\cdot 80}{\lambda_x}\right]$[1] Aufpunkten berechnet. (Dies bedeutet einen Abtastabstand von $\left[\frac{\lambda_x}{2}\right]$.)
Tabelle 4.4 zeigt die Testergebnisse von 6 Experimenten, bei denen die Eingabevektoren durch Faltung mit Gaborfiltern mit unterschiedlichen Durchlaßbereichen erzeugt wurden. Es werden wieder beide Netze (*offline*-LSDCS und *online*-LSDCS) trainiert.
Aus der Tabelle wird ersichtlich, daß für die Merkmalsextraktion aus einem 80×80 Pixel großen Gesichtsbild ein Gaborfilter der Breite $\lambda_x = 40$ am geeignetsten ist. Eine Erhöhung oder Verringerung der Filterbreite führt bei beiden Netztypen zu einem höheren Fehler in der Orientierungbestimmung.

Die Anzahl der Gaborfilter

Für die Orientierungsbestimmung kann es vorteilhaft sein, außer den oben bestimmten optimierten Gaborfiltern der Breite $\lambda_x = 40$ zusätzlich noch andere Gaborfilter mit anderen Durchlaßbereichen anzuwenden. Besonders wenn die Orientierung von verschiedenen Objekten unterschiedlicher Größe bestimmt werden soll, ist die Anwendung von mehreren

[1] Mit $[\cdot]$ wird der ganzzahlige Teil eines reellen Wertes bezeichnet.

λ_x	LSDCS offline		LSDCS online	
	mittlerer Testfehler	maximaler Testfehler	mittlerer Testfehler	maximaler Testfehler
	(Grad/Drehrichtung)			
10	1.60	5.09	2.69	9.41
20	0.94	3.26	1.99	9.50
30	0.67	1.96	1.43	5.46
40	0.64	1.88	1.26	4.42
50	0.76	2.77	3.29	11.22

Tabelle 4.4: Testergebnisse der Orientierungsbestimmung eines Puppengesichtes mit zwei LSDCS-Netzen (offline, online), wobei der Durchlaßbereich der Gaborfilter variiert.

Gaborfiltern mit verschiedenen Durchlaßbereichen empfehlenswert.
In dem nächsten Experiment werden die skalierten Gesichtsbilder mit 1 bis 3 Gaborfiltern mit unterschiedlichen Durchlaßbereichen gefaltet. Die Faltung eines Bildes mit einem Gaborfilter der Breite λ_x wird, wie in den vorigen Versuchen, wieder an $\left[\frac{2\cdot 80}{\lambda_x}\right] \times \left[\frac{2\cdot 80}{\lambda_x}\right]$ Aufpunkten berechnet.
Das LSDCS-Netz wird wieder sowohl *online* als auch *offline* trainiert. Beim *online*-Training wird die Anzahl der Trainingsschritte auf $t_{max} = 20\,000$ erhöht. Die Testergebnisse sind in der Tabelle 4.5 zusammengefaßt.

Anzahl der Gaborfilter	λ_x	LSDCS offline		LSDCS online	
		mittlerer Testfehler	maximaler Testfehler	mittlerer Testfehler	maximaler Testfehler
		(Grad/Drehrichtung)			
1	40	0.64	1.88	1.26	4.42
2	20/40	0.88	2.45	1.99	7.17
2	30/40	0.82	5.15	1.25	3.55
2	40/50	0.61	1.98	1.14	3.10
3	20/30/40	0.78	2.81	1.84	5.79
3	30/40/50	0.69	2.36	1.19	3.28

Tabelle 4.5: Testergebnisse der Orientierungsbestimmung eines Puppengesichtes mit zwei LSDCS-Netzen (offline, online), wobei die Gesichtsbilder mit mehreren Gaborfiltern mit unterschiedlichen Durchlaßbereichen gefaltet werden.

Durch die Faltung mit zusätzlichen Gaborfiltern wird keine deutliche Verbesserung der Netzausgaben erreicht. Die Anwendung zusätzlicher Gaborfilter sollte wegen des erhöhten Rechenaufwandes nur dann in Erwägung gezogen werden, wenn die Resultate (Testfehler) den Genauigkeitsanforderungen nicht genügen oder zwischen unterschiedlichen im Bild vorkommenden Objekten diskriminiert werden soll.

Anwendung der Gaborfilter auf Farb- und Grauwertbilder

Bisher wurden die Farbbilder zu Grauwertbildern konvertiert und die Bildfaltungen auf letzteren durchgeführt. Durch die Konvertierung wird der Rechenaufwand der Faltung zwar verringert, es geht aber auch Information über die Pixelfarbwerte verloren.
Ob diese zusätzliche Information über die Farbwerte zu einer Verbesserung der Orientierungsbestimmung führt, wird im nächsten Versuch getestet. Es werden Trainings- und Testmengen erstellt, sowohl mit Merkmalsextraktion aus Farb- als auch aus Grauwertbildern. Alle anderen Versuchsbedingungen sind gleich. Mit den beiden Trainingsmengen werden wieder die beiden LSDCS-Netze trainiert. Die Testergebnisse sind in Tabelle 4.6 dargestellt.

Bildtyp	LSDCS offline		LSDCS online	
	mittlerer Testfehler	maximaler Testfehler	mittlerer Testfehler	maximaler Testfehler
	(Grad/Drehrichtung)			
Farbbild	0.91	3.93	2.00	7.11
Grauwertbild	0.87	2.78	1.77	7.78

Tabelle 4.6: Testergebnisse der Orientierungsbestimmung eines Puppengesichtes mit zwei LSDCS-Netzen (offline, online), wobei die Gaborfilter sowohl auf Farb- als auch auf Grauwertbilder angewendet werden.

Die Ergebnisse der beiden Versuche sind ähnlich. Die Faltung mit Grauwertbildern schneidet dabei etwas besser ab. Daß die zusätzliche Information der Farbbildfaltung zu keiner Verbesserung der Orientierungsbestimmung führt, kann u.a. dadurch erklärt werden, daß das Netz, wie bei der Bestimmung der optimierten Anzahl der Aufpunkte bei der Faltung schon festgestellt wurde, eine notwendige Menge an Informationen braucht, um eine ausreichend genaue Gesichtsorientierung zu liefern, zusätzliche Information führt aber zu keiner Verbesserung der Genauigkeit.
Ferner wird durch die Konvertierung der Farbbilder, wobei die Farbwerte der einzelnen Bänder, und damit auch die Rauschanteile, gemittelt werden, eine Rauschverminderung erzielt.
In unseren Experimenten haben die Objekte, deren Pose bestimmt werden soll, ähnliche Farben (Gesichtsfarbe). Die Faltung mit Farbbildern ist u.U. bei der Posenbestimmung von Objekten mit deutlich unterschiedlichen Farben nützlich.

Die optimierte Methode zur Merkmalsextraktion

In den vorigen Abschnitten wurden in mehreren Versuchsreihen die in Kapitel 2 beschriebenen Methoden zur Merkmalsextraktion aus Bildern untersucht.
Im folgenden wird ein optimiertes Verfahren zur Merkmalsextraktion, das zur Fehlerminimierung in der Orientierungsbestimmung führt, zusammengefaßt:
Die Schwellenwerte der Farbsegmentierung sind $\mathbf{s}^{\mathbf{min}} = (40, 75, 130)$ und $\mathbf{s}^{\mathbf{max}} = (255{,}114,$

170) (diese sind anwendungsabhängige Werte!). Die Gesichtslokalisierung geschieht schwerpunktbezogen nach Gl.(2.4). Der Gesichtsbereich wird auf ein 80×80 Pixel großes Bild skaliert. Die Pixel im Hintergrund behalten ihre ursprünglichen Farbwerte. Die Gesichtsregion wird in ein Grauwertbild konvertiert und an 4×4 Aufpunkten mit 4 Gaborfiltern, die sich nur in ihren Orientierungen unterscheiden, gefaltet ($\lambda_x = 40$, $\sigma_x = 10$ und $\theta = 0$, $\pi/4$, $2\pi/4$ und $3\pi/4$). Damit entstehen 64-dimensionale Eingabevektoren. Die Ausgabevektoren (Orientierung des Gesichtes) sind zweidimensional (horizontales Drehen, vertikales Kippen).

4.4.2 Die neuronalen Netze

Im vorigen Abschnitt wurden die verschiedenen Methoden der einzelnen Bildverarbeitungsschritte miteinander verglichen und somit ein optimiertes Verfahren zur Merkmalsextraktion bestimmt.
In diesem Kapitel erfolgt ein Vergleich der in Kapitel 3 eingeführten neuronalen Netze.
Zur Merkmalsextraktion aus den Trainings- und Testbildern wird bei jedem Versuch das im Abschnitt 4.4.1 bestimmte optimierte Verfahren angewendet. Die zur Erzeugung der Trainings- und Testdaten benötigten Bildsequenzen sind in allen Versuchen gleich.
Alle vier in Kapitel 3 vorgestellten Netztypen (DCS-, LDCS-, SDCS- und LSDCS-Netze) werden sowohl *online* als auch *offline* trainiert. Die Parametereinstellungen der LSDCS-Netze in den Versuchen sind, soweit nicht anders vermerkt, wie im Abschnitt 4.4.1 angegeben. Die Parametereinstellungen der DCS-, LDCS- und SDCS-Netze sind, soweit nicht anders vermerkt, wie beim LSDCS-Netz.

Neuronale Netze mit und ohne lineare Erweiterung

Die Netzausgaben können entweder nur aufgrund der Ausgabevektoren oder der Ausgabevektoren und der lokalen Jacobi-Matrizen der aktivierten Neuronen berechnet werden. Welche Änderung die lineare Erweiterung bewirkt, wird aus den nächsten Versuchen ersichtlich.
Mit den vorhandenen Trainingsdaten werden DCS-, LDCS-, SDCS- und LSDCS-Netze sowohl *online* als auch *offline* trainiert.
Die Trainingsergebnisse in der Tabelle 4.7 zeigen, daß die Testfehler der Netze ohne lineare Erweiterung um ein Vielfaches höher sind als die Testfehler von Netzen mit linearer Erweiterung. Auch wenn durch die Berechnung der lokalen Jacobi-Matrizen der Rechenaufwand deutlich erhöht wird, erweisen sich neuronale Netze mit linearer Erweiterung geeigneter als Netze mit einfachen Ausgaben.

Neuronale Netze mit verschiedenen Aktivierungsfunktionen

In Kapitel 3 wurden zwei verschiedene dynamische Aktivierungsfunktionen vorgestellt: Die erste aktiviert für eine Eingabe $\mathbf{x} \in \mathcal{R}^{\mathbf{m}}$ die *best matching unit*, $bmu(\mathbf{x})$, und ihre Nachbarn. Bei der zweiten wird nur die *best matching unit*, $bmu(\mathbf{x})$, aktiviert. Die Einbeziehung der

Netztyp	mittlerer Testfehler	maximaler Testfehler	Netztyp	mittlerer Testfehler	maximaler Testfehler
	(Grad/Drehrichtung)			(Grad/Drehrichtung)	
offline DCS	2.65	6.35	offline LDCS	0.73	2.38
offline SDCS	2.50	7.19	offline LSDCS	0.64	1.88
online DCS	3.50	9.51	online LDCS	1.84	9.21
online SDCS	3.61	11.98	online LSDCS	1.26	4.42

Tabelle 4.7: Testergebnisse der Orientierungsbestimmung eines Puppengesichtes durch *offline* und *online* DCS-, LDCS-, SDCS- und LSDCS-Netze.

Nachbarn in die Netzausgabe kann sinnvoll sein, wenn die zu approximierende Funktion in der Umgebung eines Neurons annähernd linear ist. Ist das nicht der Fall, so können die Ausgaben der Nachbarn die Gesamtausgabe des Netzes verschlechtern.
Der nächste Versuch vergleicht die Approximationsgüte von Netzen mit den beiden Aktivierungsfunktionen. Es werden LDCS- und LSDCS-Netze sowohl *online* als auch *offline* mit beiden Aktivierungsfunktionen trainiert. Die Trainingsergebnisse sind in der Tabelle 4.8 zusammengefaßt.

Netztyp	nur *bmu* ist aktiviert		*bmu* und seine Nachbarn sind aktiviert	
	mittlerer Testfehler	maximaler Testfehler	mittlerer Testfehler	maximaler Testfehler
	(Grad/Drehrichtung)			
offline LDCS	0.50	1.24	0.73	2.38
offline LSDCS	0.92	3.46	0.64	1.88
online LDCS	3.74	22.15	1.84	9.21
online LSDCS	1.08	4.11	1.26	4.42

Tabelle 4.8: Testergebnisse der Orientierungsbestimmung eines Puppengesichtes mit offline und online LDCS- und LSDCS-Netzen mit zwei verschiedenen Aktivierungsfunktionen.

Die Testergebnisse sind bei den verschiedenen Netztypen unterschiedlich. Da das Lernen bei dem *online* LDCS-Netz mit der Aktivierung der *best matching unit* und ihrer Nachbarn deutlich besser ist, wird diese Methode bevorzugt.

Der Schwellenwertfaktor der Eigenwertberechnung

In Kapitel 3.4.1 wurde die Projektion der Eingabedaten auf die lokalen intrinsischen Unterräume in der Umgebung der Neuronen beschrieben. Zur Bestimmung einer Basis eines solchen Unterraumes lokal in der Umgebung eines Neurons $i \in E$ wurden die Eigenwerte und Eigenvektoren der Korrelationsmatrix $\mathbf{C_i}$ des Neurons i zu seinen Nachbarn bestimmt. Die Basis des lokalen intrinsischen Unterraumes besteht aus denjenigen Eigenvektoren von $\mathbf{C_i}$, deren zugehörige Eigenwerte größer sind als der nach Gl.(3.41) definierte Schwellenwert. Dieser Schwellenwert ist ein Bruchteil des größten Eigenwertes und wird durch einen

Schwellenwertfaktor ϵ definiert. Ist der Schwellenwertfaktor 0, so bilden alle Eigenvektoren zusammen eine Basis des Unterraumes. Die Erhöhung dieses Faktors verursacht i.a. die Erniedrigung der Dimension des Unterraumes.
Die Größe der Eigenwerte läßt auf die Varianz der Eingabedaten in die Richtung der zugehörigen Eigenvektoren schließen. Eigenvektoren mit (im Vergleich zu den anderen) sehr kleinen Eigenwerten entstehen durch Rauschen. Der Grund für die Einführung eines Schwellenwertes ist die Herausfilterung von Rauschen aus den Eingabedaten.
Die Untersuchung der Auswirkungen des Schwellenwertfaktors auf die Posenbestimmung durch die neuronalen Netze ist der Gegenstand der nächsten Versuchsreihe. Ein *offline* LSDCS-Netz wird mit verschiedenen Schwellenwertfaktoren trainiert. Die Ergebnisse sind der Abbildung 4.3 zu entnehmen.

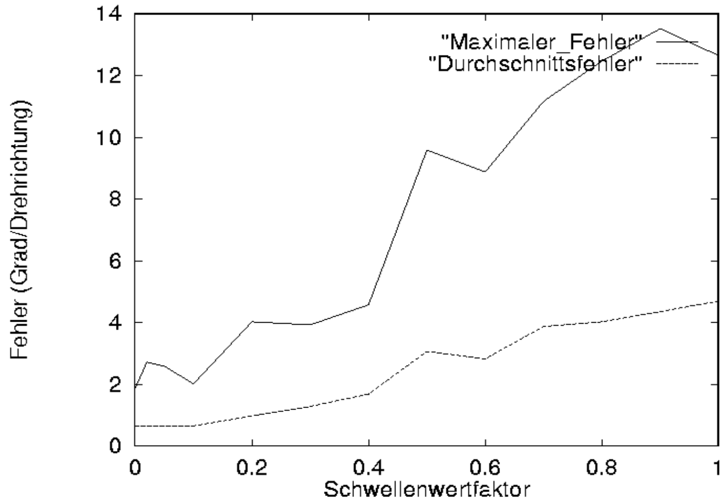


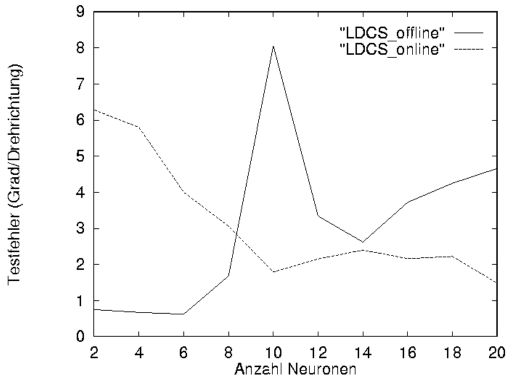
Abbildung 4.3: Der durchschnittliche und der maximale Testfehler eines vollständig trainierten *offline* LSDCS-Netzes in Abhängigkeit von dem Schwellenwertfaktor ϵ der Eigenwertberechnung.

Die Ergebnisse zeigen einen minimalen durchschnittlichen Testfehler (err = 0.64 Grad/ Drehrichtung) bei einer Einstellung von $\epsilon = 0.01$ des Schwellenwertfaktors. Der Testfehler mit dieser Einstellung ist aber nicht viel kleiner als mit $\epsilon = 0$ (err = 0.66 Grad/Drehrichtung). Wird der Schwellenwertfaktor über $\epsilon = 0.1$ erhöht, so kann ein deutlicher Anstieg des durchschnittlichen Testfehlers erkannt werden.
Diese Resultate zeigen, daß das Rauschen durch die Konvertierung der Farbbilder in Grauwertbilder und durch die Faltung mit Gaborfiltern sehr gut herausgefiltert wird.

Anzahl der Neuronen im Netz

Das nächste Experiment soll das Verhältnis zwischen der Anzahl der Neuronen und dem durchschnittlichen Testfehler der Netzausgaben verdeutlichen. Die folgenden Netztypen werden verwendet: Ein *offline* LDCS- und ein *offline* LSDCS-Netz, außerdem *online* LDCS- und LSDCS-Netze, die mit einer unterschiedlichen Anzahl von Neuronen initialisiert werden. Beim *online*-Training werden 500 Trainingsschritte pro Neuron durchgeführt ($t_{max} = 500 \cdot n$, wobei n die Anzahl der Neuronen im Netz bezeichnet).

a)



b)

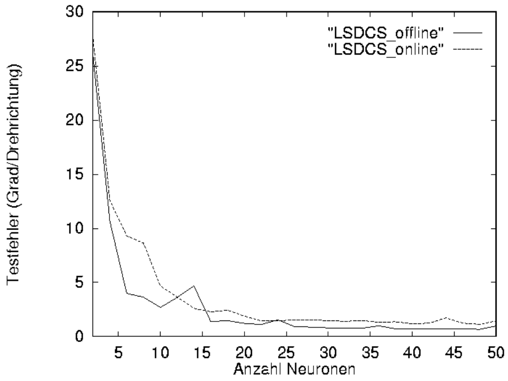


Abbildung 4.4: Der durchschnittliche Testfehler nach vollständigem Training von *online* und *offline* a) LDCS- und b) LSDCS-Netzen in Abhängigkeit von der Anzahl der Neuronen im Netz.

Abbildung 4.4 zeigt die durchschnittlichen Testfehler der Netzausgaben in Abhängigkeit von der Anzahl der Neuronen in den verschiedenen Netzen.
Das *online* LDCS-Netz erreicht mit 20, das *offline* LDCS-Netz mit 6 Neuronen einen minimalen Approximationsfehler. Mit 7 Neuronen hat das *offline* LDCS-Netz die Trainingsdaten auswendiggelernt und der Testfehler steigt an. Ab 8 Neuronen ist die Anzahl der Unbekannten im Gleichungssystem (3.35) zur Berechnung der Ausgaben größer als die Anzahl der vorhandenen Gleichungen. Somit ist das Gleichungssystem unterbestimmt und der Testfehler wird weiter erhöht.
Bei den *offline* und *online* LSDCS-Netzen ist der Fehlerverlauf ähnlich, wobei der minimale Testfehler beim *offline*-Netz niedriger ist. Die beste Approximation wird beim *offline* LSDCS-Netz mit 42, beim *online* LSDCS-Netz mit 40 Neuronen erreicht. Ein Auswendiglernen der Trainingsdaten kommt bis zu einer Anzahl von 50 Neuronen nicht vor.

Die Netztopologie

In den vorigen Versuchen wurden die Testergebnisse der verschiedenen neuronalen Netztypen verglichen. Dieser Abschnitt behandelt die Topologie der Netze, ohne dabei die Genauigkeit der Approximation zu beachten.
Ein *offline* LSDCS-Netz wird trainiert und seine Topologie nach 5, 10, 20 und 40 Lernschritten in Abbildung 4.5 in den ersten zwei Dimensionen des Eingaberaumes (Zeile a) und im Ausgaberaum (Zeile b) dargestellt. In der Zeile c) ist die Topologie von vier *online* LSDCS-Netzen in den ersten zwei Dimensionen des Eingaberaumes abgebildet. Die Netze werden mit 5, 10, 20 und 40 Neuronen initialisiert, um ihre Topologie mit der Topologie des

offline LSDCS-Netzes vergleichen zu können. In der Zeile d) sind die *online* LSDCS-Netze im Ausgaberaum dargestellt. Die Punkte im Hintergrund zeigen die Lage der Trainingsdaten.
Durch die Darstellungen im Ausgaberaum wird veranschaulicht, wie die Topologie der Trainingsdaten durch die neuronalen Netze erfaßt wird: Es ergibt sich eine zweidimensionale Verbindungsstruktur. Die Ausgabevektoren der Neuronen sind im Ausgaberaum gleichmäßig verteilt. Ferner sind nur Neuronen mit ähnlichen Ausgaben miteinander verbunden. Bei den Netzen mit mehr als 10 Neuronen existieren auch keine sich kreuzenden Verbindungen.

Der Rechenaufwand

Bei einer Echtzeit-Anwendung ist außer der Genauigkeit der Approximation durch die neuronalen Netze auch die für das Training benötigte Zeit von großer Bedeutung.
In diesem Abschnitt werden die für das vollständige Training benötigte Rechenzeiten der LDCS- und LSDCS-Netze mit *offline*- und *online*-Training miteinander verglichen. Die Daten sind in der Tabelle 4.9 zusammengefaßt. Die Rechenzeiten geben die für das Training benötigte CPU-Zeit in Sekunden an. Es soll darauf hingewiesen werden, daß diese Zeitangaben vom Rechnertyp und der Ausstattung abhängig und daher relativ sind.

Netztyp	Rechenzeit (Sek.) (CPU-Zeit)
offline LDCS	50
offline LSDCS	475
online LDCS	103
online LSDCS	478

Tabelle 4.9: Für ein vollständiges Training der offline und online LDCS- und LSDCS-Netze benötigte Rechenzeit in Sekunden.

Der Rechenaufwand bei den LDCS-Netzen ist deutlich geringer als bei den LSDCS-Netzen. Zwar sind die Eingabevektoren und damit die Zentren und die Jacobi-Matrizen der Neuronen beim LDCS-Netz hochdimensional, aber die lokale intrinsische Unterräume müssen hier nicht berechnet werden. Bei einer relativ hohen Anzahl von Neuronen ist die Berechnung der Unterräume sehr rechenaufwendig, dadurch wird aber eine genauere Approximation erreicht.
Nach einem abgeschlossenen Training ändert sich die Netzstruktur nicht mehr, d.h. die intrinsischen Unterräume müssen nicht mehr neu berechnet werden. Die Berechnung der Netzausgaben des LSDCS-Netzes erfordert nach dem Training wegen der niedrigdimensionalen Jacobi-Matrizen einen kleineren Rechenaufwand als die Netzausgaben des LDCS-Netzes.

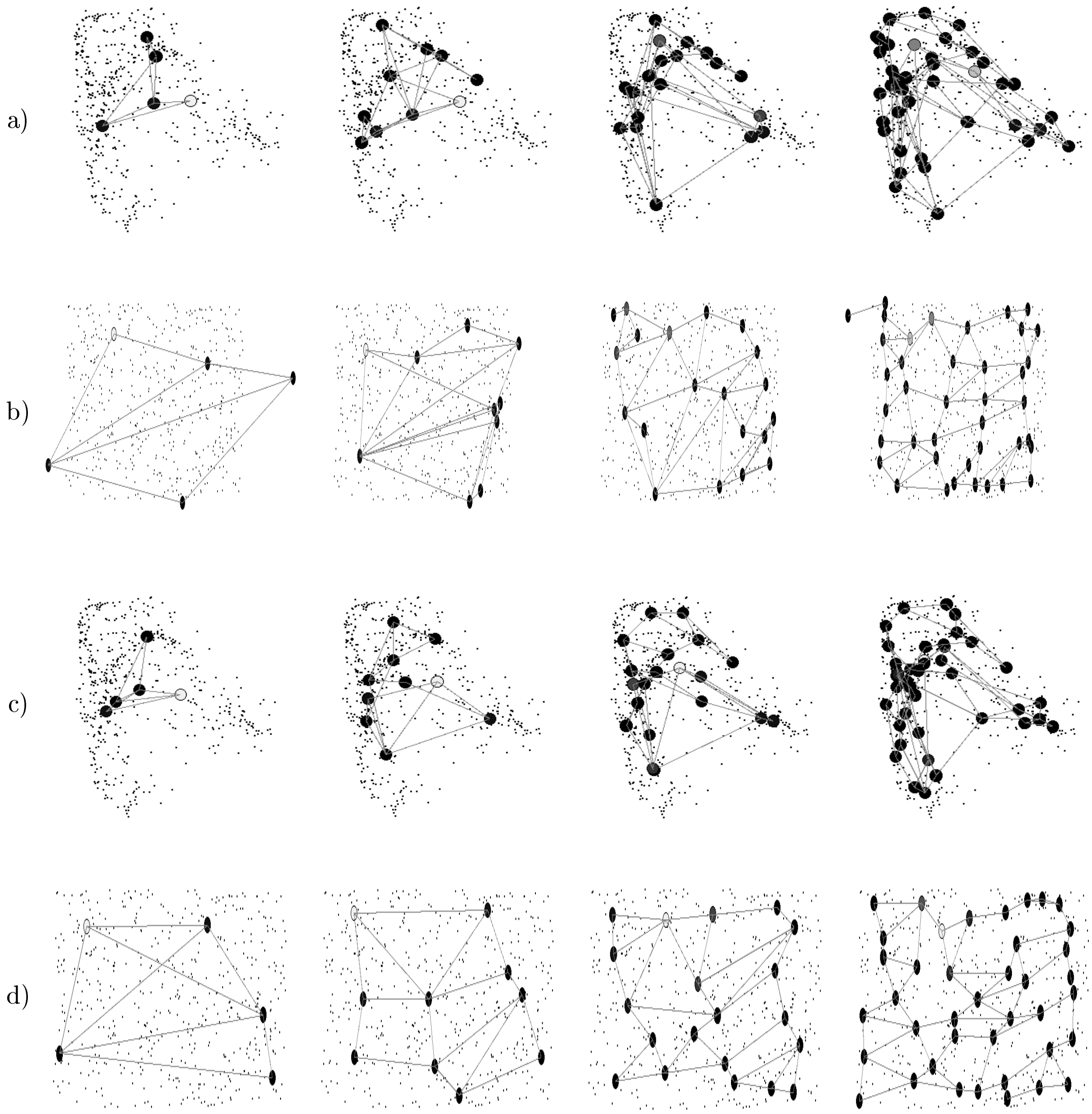


Abbildung 4.5: a) Die Entwicklung der Topologie eines *offline* LSDCS-Netzes nach 5, 10, 20 und 40 Lernschritten (von links nach rechts), dargestellt in den ersten zwei Dimensionen des Eingaberaumes und b) im Ausgaberaum. c) Die Topologie eines *online* LSDCS-Netzes, initialisiert mit 5, 10, 20 und 40 Neuronen (von links nach rechts), dargestellt in den ersten zwei Dimensionen des Eingaberaumes und d) im Ausgaberaum.

Vergleich der verwendeten Netztypen

In den vorigen Versuchen wurden verschiedene Netztypen angewendet und ihre Eigenschaften verglichen.
Neuronale Netze mit linearer Erweiterung haben deutlich bessere Approximationseigenschaften als neuronale Netze mit einfachen Ausgaben. Durch die Berechnung der Jacobi-Matrizen wird der Rechenaufwand erhöht, es wird aber eine um ein Vielfaches verbesserte Approximation erreicht.
Die beiden verschiedenen Aktivierungsfunktionen, die in Kapitel 3 vorgestellt wurden, weisen keine deutlichen Vor- oder Nachteile auf. Bei der Aktivierung der *best matching unit* ist der Rechenaufwand kleiner, dagegen liefert das *online*-Training mit einer auf die Nachbarn erweiterten Aktivierungsfunktion bessere Ergebnisse.
Die Einführung eines Schwellenwertfaktors bei der Eigenwertberechnung in Gl.(3.41) sollte die Wirkung des Rauschens mindern. Da aber die Merkmalsvektoren aus Grauwertbildern durch Faltung mit Gaborfiltern, die selbst rauschfilternd wirken, extrahiert werden, haben die Eingabedaten einen relativ kleinen Rauschanteil. Aus diesem Grund kann ein Schwellenwertfaktor $\epsilon > 0$ in dieser Anwendung keine deutliche Verbesserung der Ergebnisse mit $\epsilon = 0$ bewirken.
Im Verlauf des Trainings nimmt der Testfehler bei allen Netzen im Mittel ab, bis ein minimaler Testfehler erreicht wird. Beim *offline* LSDCS-Netz bleibt der Approximationsfehler beim Weitertrainieren relativ konstant. Beim *offline* LDCS-Netz tritt ein Überlerneffekt auf und der Testfehler erhöht sich. Daß beim *offline* LSDCS-Netz kein Überlerneffekt auftritt, ist auf die Projektion auf die lokalen intrinsischen Unterräume zurückzuführen.
Beim *online*-Training müssen die Netze mit einer minimal nötigen Anzahl von Neuronen initialisiert werden, um eine optimierte Approximation lernen zu können. Wird das Netz mit mehr Neuronen als diese minimal nötige Anzahl initialisiert, so kann zwar die Approximation nicht mehr verbessert werden, sie wird aber auch nicht verschlechtert.
Die Versuchsergebnisse haben gezeigt, daß die *offline*-Netze bessere Approximationseigenschaften besitzen als die Netze mit *online*-Training. Ein Grund dafür ist, daß die *offline*-Netze Zugriff auf die gesamte Trainingsmenge haben, während den Netzen mit *online*-Training zu jedem Zeitpunkt nur ein Trainingsdatum zur Verfügung steht und die Approximation nur lokal in kleinen Schritten gelernt werden kann. Ferner haben die *online*-Netze mehr Lernparameter als die *offline*-Netze. Beim *online*-Training müssen 14 Lernparameter bestimmt werden:

- Anzahl der Neuronen im Netz;
- Anzahl der maximalen Lernschritte;
- Parameter p der Aktivierungsfunktion in Gl.(3.14) bzw. Gl.(3.43);
- Schwellenwert ϵ der Eigenwertberechnung in Gl.(3.41);
- Vergessenskonstante α in Gl.(3.29);

- Schwellenwert θ in Gl.(3.29) und
- die Lernkonstanten $\epsilon_{bmu}^{max}, \epsilon_{bmu}^{min}, \epsilon_{Nh}^{max}, \epsilon_{Nh}^{min}, \epsilon_{out}^{max}, \epsilon_{out}^{min}, \epsilon_{jac}^{max}, \epsilon_{jac}^{min}$ für die Bestimmung der Lernschrittweiten.

Das *offline*-Training hat dagegen nur 2 Lernparameter:

- Parameter p der Aktivierungsfunktion in Gl.(3.14) bzw. Gl.(3.43) und
- Schwellenwert ϵ der Eigenwertberechnung in Gl.(3.41).

Die Bestimmung der optimierten Werte der Lernparameter ist oft schwierig und meistens nur experimentell möglich.
Zwischen den Testergebnissen der *offline* LDCS- und LSDCS-Netze kann kein auffallender Unterschied festgestellt werden. Erwartungsgemäß würde bei stärker verrauschten Eingabedaten das *offline* LSDCS-Netz die besseren Ergebnisse liefern.
Bei den *online*-Netzen wird die Approximation durch die Projektion der Eingabedaten auf die lokalen Unterräume verbessert. Zwar ist der durchschnittliche Testfehler des LSDCS-Netzes i.a. nicht viel geringer als beim LDCS-Netz, aber der maximale Fehler ist niedriger. Ein maximaler Testfehler von ca. 9 Grad/Drehrichtung am Ende des Trainings eines *online* LDCS-Netzes, wie in Tabelle 4.7, kann für eine praktische Anwendung zu groß sein. Deshalb ist beim *online*-Training das LSDCS-Netz geeigneter als das LDCS-Netz.

4.4.3 Versuche mit zwei verschiedenen Objekten

In den vorausgehenden Abschnitten wurde eine optimierte Methode zur Merkmalsextraktion aus den Bildern und eine optimierte Einstellung der neuronalen Netze bestimmt. In diesem Abschnitt wird die bisherige Aufgabenstellung, also die Posenbestimmung eines Gesichtes, auf zwei Gesichter erweitert. Abbildung 4.6 zeigt die beiden Objekte.

a) b)

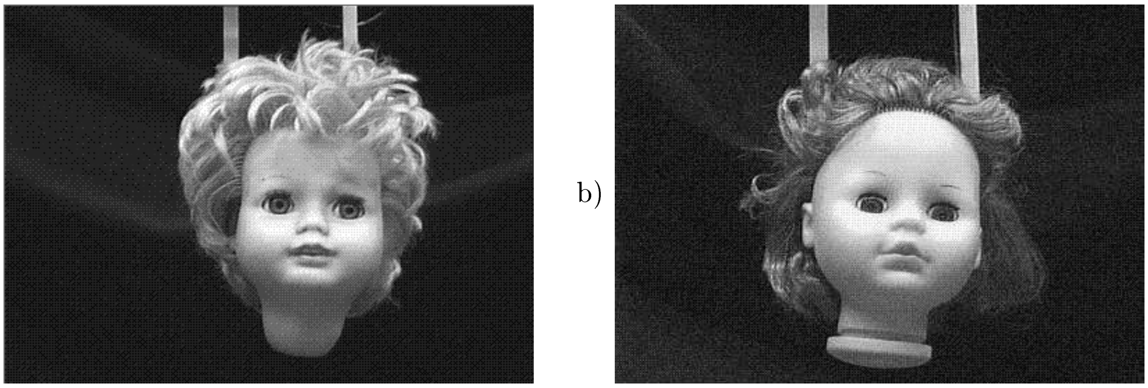


Abbildung 4.6: Die zwei verwendeten Objekte: a) Puppe I b) Puppe II.

Zusätzlich zur Posenbestimmung soll auch eine Identifikation der Gesichter erfolgen. Die

verschiedenen Objekte werden durch Zahlen identifiziert. Der Ausgabevektor jedes Trainingsdatums wird mit der Nummer des Objektes, durch dessen Merkmalsvektor der Eingabevektor des Trainingsdatums definiert ist, erweitert. Die Trainingsmengen bestehen aus gemischten Trainingsdaten der beiden Objekte.
Für die Trainingsmenge stehen 500, für die Testmenge 100 Bilder jedes Objektes mit zufälligen Orientierungen zur Verfügung, wobei sich der horizontale Drehwinkel zwischen $-75°$ und $+75°$ und der vertikale Kippwinkel zwischen $-30°$ und $+30°$ bewegen kann.
Zur Merkmalsextraktion aus den Trainings- und Testbildern wird bei jedem Versuch das im Abschnitt 4.4.1 beschriebene optimierte Verfahren angewendet. Die Schwellenwerte der Farbsegmentierung des zweiten Objektes sind $\mathbf{s_{min}} = (40, 75, 130)$ und $\mathbf{s_{max}} = (255, 128, 170)$.
Es werden LSDCS-Netze *offline* und *online* trainiert. Die Parametereinstellungen der Netze sind wie im Abschnitt 4.4.1 angegeben.

Testergebnisse der Posenerkennung des zweiten Objektes

Um die späteren Testergebnisse der Identifikation und Posenbestimmung der beiden Objekte vergleichen zu können, werden zunächst *offline* und *online* LDCS- und LSDCS-Netze nur mit den Trainingsdaten der ersten und dann nur mit den Trainingsdaten der zweiten Puppe trainiert.

Netztyp	Puppe I		Puppe II	
	mittlerer Testfehler	maximaler Testfehler	mittlerer Testfehler	maximaler Testfehler
	(Grad/Drehrichtung)			
offline LDCS	0.73	2.38	0.62	1.85
offline LSDCS	0.64	1.88	0.75	2.44
online LDCS	1.84	9.21	1.31	4.91
online LSDCS	1.26	4.42	1.24	4.39

Tabelle 4.10: Testergebnisse der Posenerkennung zweier Puppengesichter mit offline und online LDCS- und LSDCS-Netzen.

Die Testergebnisse sind in der Tabelle 4.10 angegeben. Die Fehler in der Posenbestimmung der beiden Objekte weisen keine deutlichen Unterschiede auf.

Testergebnisse der Posenerkennung und Identifikation von zwei Objekten

Bisher wurden die neuronalen Netze mit Trainingsdaten der einzelnen Objekte trainiert. Dadurch wurden die Posen der beiden Gesichter unabhängig voneinander gelernt.
In einem nächsten Versuch werden die neuronalen Netze mit gemischten Trainings- und Testdaten trainiert, die durch das Zusammenfügen der Trainings- und Testmengen der einzelnen Objekte entstehen.

Um eine Identifikation der beiden Gesichter zu ermöglichen, werden die bisherigen Sollvektoren der Trainings- und Testmengen (also die Orientierungen) um eine Identifikationskomponente erweitert: Eine "1" bedeutet, daß es sich um einen Sollvektor des ersten Objektes handelt. Eine "2" deutet entsprechend auf das zweite Objekt hin. Dadurch ergeben sich dreidimensionale Ausgabevektoren.
Die Netzausgabe spaltet sich dementsprechend auch in eine Orientierungsangabe und eine Identifikationskomponente.
Die Netzausgabe ohne Identifikationskomponente gibt die Orientierung des Objektes an. Die Identifikation eines Objektes wird vom Netz durch die Identifikationskomponente der Netzausgabe bestimmt. Wenn für die Identifikationskomponente o_{id}^{net} der Netzausgabe $0.5 \leq o_{id}^{net} < 1.5$ gilt, so wird das erste Objekt identifiziert, im Falle $1.5 \leq o_{id}^{net} < 2.5$ das zweite. Bei $o_{id}^{net} < 0.5$ oder $2.5 \leq o_{id}^{net}$ wird die Identifikation als falsch bewertet.
Der Testfehler ist durch den durchschnittlichen und maximalen Testfehler der Orientierungsbestimmung und durch den Identifikationsfehler auf den Testdaten definiert.
Der durchschnittliche und der maximale Testfehler läßt sich als der durchschnittliche und der maximale Abstand der Netzausgaben und der Sollvektoren der Testdaten ohne Identifikationskomponente angeben.
Der Identifikationsfehler der Testdaten wird in Prozent angegeben und durch

$$err_{id} = \frac{\sum_{(\mathbf{x},\mathbf{y}) \in Test} |y_{id} - [o_{id}^{net}(\mathbf{x}) + 0.5]|}{|Test|} \tag{4.1}$$

berechnet, wobei y_{id} bzw. $o_{id}^{net}(\mathbf{x})$ die Identifikationskomponente des Sollvektors des Testdatums bzw. der Netzausgabe bezeichnet.[2] Er gibt den Prozentsatz der falsch klassifizierten Daten an.
Die *online*-Netze werden mit 80 Neuronen initialisiert. Bei der Initialisierung wird die Hälfte der Neuronen mit Trainingsdaten des ersten, die andere Hälfte mit Trainingsdaten des zweiten Objektes initialisiert. Die Anzahl der Trainingsschritte wird auf 30 000 erhöht. Der Schwellenwert in Gl.(3.29) für das Löschen einer Verbindung zwischen zwei Neuronen wird wegen der größeren Trainingsmenge auf $\theta = 0.03$ eingestellt.
Tabelle 4.11 zeigt die Testergebnisse. Der mittlere und der maximale Testfehler wird in Grad/Drehrichtung, der Identifikationsfehler in Prozent angegeben.
Alle Netztypen konnten die Identifikation der beiden Objekte vollständig lernen. Dies deutet darauf hin, daß sich die Eingaberäume der beiden Objekte nicht überlappen.
Die Testfehler beim gemeinsamen Training der beiden Objekte liegen in der Größenordnung der Testfehler beim Training der einzelnen Objekte. D.h. durch die vereinigte Trainings- und Testmenge werden die Approximationsergebnisse der einzelnen Objekte nicht verschlechtert.
Außer dem *online* LDCS-Netz erweisen sich alle Netztypen mit einem in der Praxis vertretbaren Testfehler als geeignet. Wie in den vorigen Versuchen, sind die Approximationsfehler der beiden *offline*-Netze niedriger als der des *online* LSDCS-Netzes.

[2] Mit $[\cdot]$ wird der ganzzahlige Teil eines reellen Wertes bezeichnet.

Objekt	Testfehler	offline LDCS	offline LSDCS	online LDCS	online LSDCS
Puppe I	mittlerer	0.71	0.82	1.97	1.17
	maximaler	2.16	2.51	16.28	4.06
	Identifikation	0	0	0	0
Puppe II	mittlerer	0.62	0.81	1.33	1.17
	maximaler	1.60	2.48	4.81	3.81
	Identifikation	0	0	0	0
Puppe I+II	mittlerer	0.67	0.81	1.68	1.17
	maximaler	2.16	2.51	16.28	4.06
	Identifikation	0	0	0	0

Tabelle 4.11: Testergebnisse der Posenerkennung und Identifikation zweier Puppengesichter mit offline und online LDCS- und LSDCS-Netzen. Der mittlere und der maximale Testfehler wird in Grad/Drehrichtung, der Identifikationsfehler in Prozent angegeben.

In vielen Anwendungen, wie z.B. bei Steuerungsaufgaben, liegt der Schwerpunkt der Interesse an der Orientierungserkennung unterschiedlicher Gesichter. Eine Identifikation der Gesichter wird nicht gefordert. Das oben beschriebene Verfahren kann in diesem Fall vereinfacht werden, indem die Netze mit Trainingsdaten ohne Identifikationskomponenten trainiert werden. Die Trainingsmenge entsteht dann durch einfaches Zusammenfügen der Trainingsdaten aller Gesichter.
Tabelle 4.12 zeigt die Ergebnisse eines solchen Trainings für zwei Objekte. Die Testfehler der Orientierungsbestimmung ohne die Erweiterung der Trainingsdaten mit Identifikationskomponenten sind mit den vorigen Ergebnissen vergleichbar.

Objekt	Testfehler	offline LDCS	offline LSDCS	online LDCS	online LSDCS
Puppe I+II	mittlerer	0.73	0.81	1.24	1.51
	maximaler	3.07	3.46	5.97	4.82

Tabelle 4.12: Testergebnisse der Posenerkennung zweier Puppengesichter mit offline und online LDCS- und LSDCS-Netzen. Die Netze wurden mit Trainingsdaten der beiden Objekte ohne Identifikationskomponenten trainiert. Der mittlere und der maximale Testfehler wird in Grad/Drehrichtung angegeben.

Ein Nachteil des vorgestellten Trainings mit mehreren Objekten ist, daß das Netz für jedes neue Gesicht neu trainiert werden muß, also nicht für verschiedene Gesichter generalisiert. Eine Lösung dieses Problems könnte in einer abgewandelten Merkmalsextraktion bestehen, die weniger sensitiv auf die individuellen Besonderheiten reagiert. In unseren Versuchen sind die Eingaberäume der beiden Objekte so unterschiedlich, daß ein mit den Trainingsdaten der ersten Puppe trainiertes Netz auf Eingabevektoren der zweiten Puppe unbrauchbare Antworten liefert. Beim *offline*-Training des LSDCS-Netzes ergibt sich z.B. nach dem Training mit der Trainingsmenge der Puppe I ein Durchschnittsfehler von ca.

50° auf der Testmenge der Puppe II.

4.5 Anwendungen

In den vorigen Abschnitten wurden eine optimierte Methode zur Merkmalsextraktion aus Bildern und optimierte Parametereinstellungen der neuronalen Netze entwickelt. Dadurch konnte beim *offline* LSDCS-Netz für zwei Objekte eine fehlerfreie Identifikation und ein durchschnittlicher Testfehler von 0.81 Grad/Drehrichtung erreicht werden. Der maximale Testfehler betrug 2.51 Grad/Drehrichtung.
Diese Genaugkeit der Posenbestimmung ermöglicht die Anwendung in vielen technischen Bereichen. In erster Linie kommen Steuerungsaufgaben mit zwei Freiheitsgraden in Frage. Im folgenden werden aus dem Bereich der Computer- und Robotersteuerung als Beispiele drei Anwendungsmöglichkeiten vorgestellt: Ein Malprogramm, ein Programm zur Maussteuerung und ein Programm zur Steuerung der Orientierung des Manipulators eines Roboterarmes.
In allen Programmen wird ein *offline* LSDCS-Netz angewendet. Die Parametereinstellungen des Netzes sind wie im Abschnitt 4.4.1 angegeben. Zur Merkmalsextraktion dient das im Abschnitt 4.4.1 beschriebene optimierte Verfahren.
Für das Training stehen 600 Trainings- und 180 Testbilder von der Puppe I zur Verfügung. Die Bilder wurden unter konstanten Lichtverhältnissen aufgenommen. Der horizontale Drehwinkel bewegt sich wieder zwischen $-75°$ und $+75°$, der vertikale Kippwinkel zwischen $-30°$ und $+30°$. Durch Merkmalsextraktion entstehen eine Trainings- und eine Testmenge, mit denen das Netz trainiert werden kann.
Den Anwendungen stehen zwei weitere Bildsequenzen zur Verfügung: Die erste Bildsequenz besteht aus 180 Bildern, deren zugehörige Orientierungsvektoren (θ_1, θ_2)[3] im zweidimensionalen Orientierungsraum einen Kreis beschreiben ($-25° \leq \theta_1, \theta_2 \leq +25°$). Die Orientierungen der Bildsequenz sind auf dem Kreis gleichmäßig verteilt. Die Gesichtsorientierungen (θ_1, θ_2) in der Bildsequenz sind in der Abbildung 4.7 a) im Ausgabenraum dargestellt. Die horizontale Achse entspricht dem horizontalen Drehwinkel $-75° \leq \theta_1 \leq +75°$, die vertikale Achse dem vertikalen Kippwinkel $-30° \leq \theta_2 \leq +30°$.
Die zweite Bildsequenz besteht aus 50 Bildern. Die Orientierungen des Gesichtes in den Bildern bilden eine Linie im Ausgaberaum. Für jedes Bild, in dem das Gesicht die Orientierung (θ_1, θ_2) hat, gilt $-40° \leq \theta_1 \leq +10°$, $-25° \leq \theta_2 \leq +25°$ und $\theta_2 = \theta_1 + 15°$. Die Orientierungen der Bildsequenz sind auf der Linie gleichmäßig aufgeteilt. Die Gesichtsorientierungen (θ_1, θ_2) in der Bildsequenz sind in der Abbildung 4.7 b) im Ausgabenraum dargestellt.

[3] Die erste bzw. zweite Komponente des Orientierungsvektors gibt die Dreh- bzw. Kippwinkel des Objektes in Grad an.

4.5.1 Malprogramm

Die erste Anwendung ist ein Malprogramm. Die Kopforientierung in den aufgenommenen Bildern kann sich zwischen $-75°$ und $+75°$ in der horizontalen und zwischen $-30°$ und $+30°$ in der vertikalen Richtung bewegen. Das entspricht $151°$ Bewegungsfreiheit in der horizontalen und $61°$ in der vertikalen Richtung.
Für das Malprogramm sei der Veranschaulichung halber ein Fenster mit der Breite von 151 Pixeln und mit der Höhe von 61 Pixeln definiert. (Natürlich können auch kleinere oder größere Fenster definiert werden, die Pixelkoordinaten müssen nur für die Darstellung dementsprechend transformiert werden.)
Jedes Pixel im Fenster korrespondiert mit einem bestimmten Intervall im Ausgaberaum, d.h. mit einem bestimmten Orientierungsintervall. Das Pixel in der Mitte des Fensters soll die Orientierung $(0, 0)$ darstellen. Rechts von diesem Punkt werden die positiven, links die negativen horizontalen Drehungen dargestellt. Die Orientierungen mit positiven Kippwinkeln werden in der oberen, die mit negativen Kippwinkeln in der unteren Fensterhälfte abgebildet.
Ist der Koordinatenursprung $(0, 0)$ des Fensters in der linken oberen Ecke, so korrespondiert das Pixel (i, j) mit $0 \leq i \leq 150$ und $0 \leq j \leq 60$ mit Netzausgaben (θ_1, θ_2) mit $i - 75 \leq \theta_1 < (i + 1) - 75$ und $j - 30 \leq \theta_2 < (j + 1) - 30$. D.h., ein bestimmter Ausgabevektor (θ_1, θ_2) wird durch das Pixel $([\theta_1] + 75, [\theta_2] + 30)$ dargestellt.[4]
Nachdem das LSDCS-Netz vollständig trainiert worden ist, legen wir als Eingabe den Merkmalsvektor eines Bildes an. Die Netzausgabe ist die zugehörige Kopforientierung (θ_1, θ_2), die im Fenster mit dem Pixel $([\theta_1] + 75, [\theta_2] + 30)$ korrespondiert. Dieses Pixel wird durch Änderung dessen Farbwertes markiert.
Danach wird ein neues Bild eingelesen, der Merkmalsvektor gebildet und an das Netz angelegt. Die Netzausgabe liefert eine neue Position im Fenster. Um beim Malen eine kontinuierliche Linie zu erzeugen, wird zwischen der alten und der neuen Position eine gerade Linie gezogen. Es werden kontinuierlich nach jeder Berechnung neue Bilder eingelesen, bis das Programm angehalten wird.
Wird der Kopf horizontal gedreht, so entsteht eine horizontale Linie im Fenster. Bei vertikaler oder schräger Bewegung folgt die gemalte Linie weiterhin spiegelartig den Orientierungsänderungen.
Wenn die Merkmalsvektoren der ersten Bildsequenz nacheinander anlegt werden, wird ein Kreis gemalt. Bei der zweiten Bildsequenz entsteht eine Linie.
Die Anwendung ist gut geeignet, die Größenordnung der auftretenden Fehler in der Orientierungsbestimmung zu veranschaulichen. Abbildung 4.7 c) und d) zeigen die Ergebnisse des Malprogrammes mit der Eingabe der ersten bzw. der zweiten Bildsequenz.
Die Steuerung durch Kopforientierung kann mit einer Sprachsteuerung ergänzt werden. Dazu wäre ein relativ einfaches Spracherkennungssystem nötig, das einige Wörter, wie z.B. "herunter" oder "hoch" erkennen kann, um den Stift herunterzulassen (um zu Malen) oder hochzuheben (um das Malen an einer neuen Position anfangen zu können).

[4] Mit $[\cdot]$ wird der ganzzahlige Teil eines reellen Wertes bezeichnet.

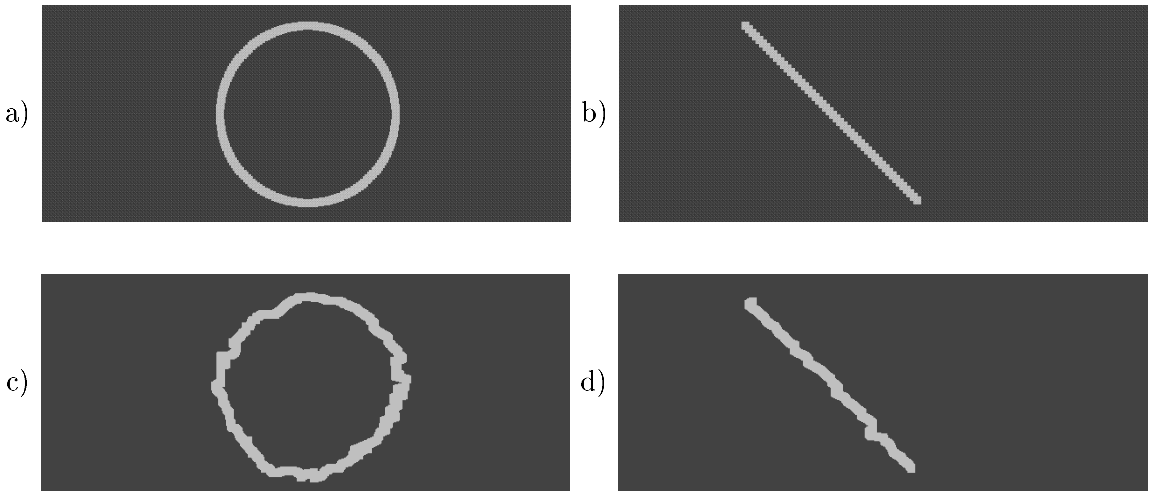


Abbildung 4.7: a) Die Gesichtsorientierungen in der ersten Bildsequenz, dargestellt im Ausgabenraum. b) Die Gesichtsorientierungen in der zweiten Bildsequenz, dargestellt im Ausgabenraum. c) Ergebnis des Malprogrammes mit der ersten Bildsequenz. d) Ergebnis des Malprogrammes mit der zweiten Bildsequenz.

4.5.2 Maussteuerung

Diese Anwendung ist der vorigen sehr ähnlich. Der Unterschied besteht darin, daß immer nur diejenige Fensterposition markiert wird, die mit der aktuellen Kopforientierung korrespondiert. Für jedes neue Bild wird der Pixelwert an der alten Position auf den Hintergrundswert zurückgestellt und der Pixelwert auf der neuen aktuellen Position auf den Vordergrundswert geändert. Dadurch wird es möglich, durch Drehung des Kopfes eine Maus zu steuern.
Wenn die Merkmalsvektoren der ersten Bildsequenz nacheinander anlegt werden, bewegt sich die Maus entlang eines Kreises. Bei der zweiten Bildsequenz bewegt sich die Maus entlang einer Linie.

4.5.3 Robotersteuerung

Die dritte Anwendung ist die Steuerung der Orientierung des Manipulators eines Roboters durch die Kopforientierung.
Ein neuronales Netz wird wieder mit den vorhandenen Trainingsdaten trainiert. Danach erwartet das Programm, wie vorher, Bildsequenzen, aus denen Merkmalsvektoren extrahiert werden. Letztere dienen als Eingabevektoren des neuronalen Netzes. Die Netzausgabe ist die Kopforientierung. In diese Orientierung wird der Manipulator auch gedreht. Somit wird die Orientierung des Manipulators mit der Kopforientierung gekoppelt.

Kapitel 5

Diskussion

In den vorigen Kapiteln wurden die theoretischen Grundlagen und die Realisierung von Erkennung und Posenbestimmung von Gesichtern mittels künstlicher neuronaler Netze beschrieben. Dieses Kapitel widmet sich der Auswertung der Ergebnisse und der Diskussion der Erweiterungsmöglichkeiten der vorgestellten Methoden. In den folgenden Abschnitten 5.1 und 5.2 werden die vorliegenden Ergebnisse ausgewertet und mit anderen Arbeiten verglichen. Der Abschnitt 5.3 schließt das Kapitel und damit die vorliegende Arbeit mit einer Diskussion über Erweiterungsmöglichkeiten des vorgestellten Konzeptes.

5.1 Auswertung der Ergebnisse

In Kapitel 2 wurden verschiedene Methoden der Merkmalsextraktion vorgestellt. Die Lokalisierung eines Gesichtes in einem Farbbild wird mit Hilfe der Farbsegmentierung durchgeführt. Die eigentliche Merkmalsextraktion geschieht durch lokale Faltung der Bilder mit Gaborfiltern. Die Faltungsergebnisse bilden die Eingabevektoren der neuronalen Netze. Durch eine Auswertung der Testergebnisse in Kapitel 4 wurde eine optimierte Methode zur Merkmalsextraktion entwickelt.

Die theoretischen Grundlagen der verwendeten neuronalen Netze wurden in Kapitel 3 beschrieben. Die besten Approximationseigenschaften weisen neuronale Netze mit *offline*-Training auf. Bei einer optimierten Einstellung der Netzparameter erreichen die *offline*-trainierten LDCS- und LSDCS-Netze bei zwei Objekten eine fehlerfreie Identifikation und einen Durchschnittsfehler in der Größenordnung von 0.6 bis 1 Grad/Drehrichtung. Der maximale Testfehler liegt zwischen 2 und 3 Grad/ Drehrichtung.

Der Testfehler des *online*-trainierten LSDCS-Netzes liegt höher als der der *offline*-Netze, ist aber in der Praxis immer noch vertretbar (fehlerfreie Idetifikation, Durchschnittsfehler $\approx$ 1.2 Grad/Drehrichtung, maximaler Fehler $\approx$ 4 Grad/Drehrichtung). Das LDCS-Netz mit *online*-Training erweist sich aufgrund der hohen maximalen Fehler auf der Testmenge als ungeeignet.

Abbildung 5.1 macht deutlich, welchem optischen, d.h. mit den Augen bemerkbaren Unterschied eine Ortientierungsänderung von 2 bzw. 4 Grad in beiden Drehrichtungen entspricht.

Die Orientierung des Objektes im ersten Bild ist $(0, 0)$, im zweiten Bild $(2, 2)$ und im dritten Bild $(4, 4)$.[1] Ein maximaler Testfehler in der Größenordnung von 2 Grad/Drehrichtung (wie bei den *offline*-Netzen) entspricht also dem optischen Unterschied zwischen den ersten beiden Bildern. Ein maximaler Testfehler in der Größenordnung von 4 Grad/Drehrichtung (wie bei dem *online* LSDCS-Netz) entspricht einem ähnlichen optischen Unterschied wie zwischen dem ersten und dem dritten Bild.

Abbildung 5.1: Puppe I mit 3 verschiedenen Orientierungen: a) Kippwinkel=Drehwinkel=0°, b)Kippwinkel=Drehwinkel=2° und c) Kippwinkel=Drehwinkel=4°.

In Kapitel 4 wurde gezeigt, daß das Zufügen von neuen Neuronen in ein Netz einer bestimmten Neuronenanzahl an zu keiner weiteren Verbessung der Testergebnisse führt, d.h. es kann kein beliebig kleiner Testfehler erzielt werden. Der Testfehler eines vollständig trainierten Netzes mit optimierten Parametereinstellungen und Neuronenanzahl resultiert aus mehreren Komponenten.

Eine Komponente ergibt sich aus dem Rauschen in den Originalbildern. Da die Trainings- und Testmengen unterschiedliche Bilder enthalten, ist der Rauschanteil in den Bildern auch unterschiedlich. Werden die Trainingsvektoren zu genau gelernt, so wird der Rauschanteil der Trainingsdaten auch mitgelernt, was wiederum zu einem erhöhten Fehler auf der Testmenge führt. Dieser Effekt tritt z.B. bei dem *offline* LDCS-Netz schon nach wenigen Trainingsschritten (d.h. mit wenigen Neuronen) auf.

Eine wesentliche Störquelle der Eingabedaten wird von der Farbsegmentierung verursacht. Die Gesichtshaut hat zwar einen konstanten Farbwert, der aber in den Bildern helligkeits- und beleuchtungsabhängig dargestellt wird. Weitere Probleme werden durch die Lichtreflexion und Spiegelung verursacht. Somit ist eine exakte Definition der Gesichtsfarbe in den Bildern sehr schwierig. Die Folgen sind Ungenauigkeiten in der Gesichtslokalisierung, die den Testfehler erhöhen.

Die zu lernende Funktion, die den Eingaberaum auf den Ausgaberaum abbildet, ist unbekannt. Nur die durch die Trainingsdaten dargestellten Stützpunkte stehen zur Verfügung. Die lineare Erweiterung der neuronalen Netze ermöglicht eine lineare Approximation. Bei rauschfreien Daten kann dadurch eine linear approximierbare Funktion beliebig genau approximiert werden. Da aber in der Praxis die Daten verrauscht sind, führt eine zu genaue Approximation der Trainingsmenge zu dem vorher erwähnten Überlerneffekt.

[1] Die erste bzw. zweite Komponente des Orientierungsvektors gibt den Dreh- bzw. Kippwinkel des Objektes in Grad an.

5.2 Vergleich der Ergebnisse mit anderen Arbeiten

Es gibt mehrere Arbeiten, die sich die Lokalisierung und die Orientierungsbestimmung von Gesichtern zur Aufgabe stellen.
Einige Arbeiten verwenden ein geometrisches Modell zur Orientierungsbestimmung. Es werden Gesichtsmerkmale wie Augen, Mund oder Nase identifiziert. Von den Merkmalen ausgehend wird die Gesichtsnormale bestimmt, aus der die Orientierung berechnet werden kann ([GC94], [BS95]).
Murase und Nayar [MN95] berechnen für jedes verwendete Objekt einen Eigenraum. Durch Projektion der Bilder eines Objektes auf dessen Eigenraum wird die Orientierung bestimmt (siehe Kapitel 2.3). Die Lokalisierung der Objekte geschieht mittels Farbsegmentierung. Mit dieser Methode wurde die Identifikation und Orientierungsbestimmung von unterschiedlichen Objekten realisiert. Auf Gesichter wurde die Methode aber nicht angewendet. Ferner wurde die Orientierung in nur eine Richtung variiert (horizontales Drehen).
Kubisch und Ritter [KR96] verwenden neuronale Netze sowohl zur Lokalisierung als auch zur Orientierungsbestimmung von Gesichter. Es wurden nur Versuche mit einem Objekt durchgeführt. Die Orientierung hat zwei Freiheitsgrade (horizontales Drehen, vertikales Kippen). Das Gesicht wird mittels Farbsegmentierung mit Hilfe eines *online*-trainierten neuronalen Netzes (LLM-Netz) grob lokalisiert. In einem zweiten Schritt wird durch Verwendung eines zweiten *online*-trainierten LLM-Netzes eine Ellipse an die Gesichtsregion angepaßt. Die Eingabe des neuronalen Netzes ist ein Merkmalsvektor, der aus den Faltungsantworten eines Satzes von Gaborfiltern besteht. Die Ausgabe gibt den Mittelpukt, die Orientierung und die Halbachsen der Ellipse an. Das neuronale Netz zur Ellipsenanpassung wird mit Hilfe von handmarkierten Trainingsdaten trainiert. Zum Training wurde ein Datensatz von 100 Trainingsbildern und 100 Testbildern aufgenommen.
Zur Erkennung der Gesichtsrichtung wird ein drittes *online*-trainiertes LLM-Netz verwendet. Es wird wiederum eine Merkmalsextraktion mit Gaborfiltern durchgeführt. Auf die Eingabe des Merkmalsvektors liefert das Netz die zweidimensionale Kopforientierung als Netzausgabe. Zur Generierung der Trainingsdaten wurde am Kopf einer Versuchsperson ein Laserzeiger befestigt. Mit dem Laserzeiger wurden im Labor verteilte Marker fixiert, die zu bekannten kalibrierten Kopfrichtungen gehörten. Der Abstand zwischen Mensch und Kamera war dabei konstant. Zum Training wurde ein Datensatz von 105 Trainingsbildern und 105 Testbildern aufgenommen. Jedes Bild wird mit der Farbsegmentierung und dem Ellipsennetz vorverarbeitet, um die Gesichtsregion zu ermitteln. Anschließend wird mittels Faltung der Merkmalsvektor bestimmt.
Die besten Ergebnisse der Orientierungsbestimmung wurden mit einem LLM-Netz mit 10 Knoten und 5000 Lernschritten erreicht. Es konnte ein mittlerer Fehler von ca. $\pm 11°$ über beide Winkel erzielt werden. Dieser Fehler ist viel größer als der in dieser Arbeit erreichte Fehler. Dafür gibt es mehrere Gründe. Einerseits wurden bei Kubisch und Ritter die Orientierungen von menschlichen Gesichtern bestimmt. Um Trainings- und Testbilder für das LLM-Netz zur Orientierungsbestimmung aufzunehmen, wurde die Kopforientierung mit Hilfe eines Laserzeigers und eines Fixierpunktes an der Wand eingestellt. Dadurch werden zwei der drei möglichen Freiheitsgrade der Kopforientierung festgelegt. Die dritte Ori-

entierungskomponente (schräges Kippen) kann aber nur nach Gefühl des Versuchspersons möglichst konstant gehalten werden. Auch die zwei durch den Fixierpunkt bestimmten Orientierungskomponenten können wegen Meß- oder Einstellungsfehler ungenau sein. Damit sind schon die Trainings- und Testdaten fehlerhaft. Ferner führt die manuelle Erzeugung der Trainingsdaten des LLM-Netzes zur Ellipsenanpassung ebenso zu Ungenauigkeiten. Durch die Hintereinanderschaltung mehrerer Netzwerke wird der Fehler fortgepflanzt.
Ein weiterer Grund liegt in den verwendeten Netzstrukturen. Die LLM-Netze sind neuronale Netze mit linearer Erweiterung und *online*-Training. In Kapitel 4 wurde gezeigt, daß bei den vorgestellten Netztypen mit *offline*-Training bessere Ergebnisse erzielt werden können als mit *online*-Training. Weiterhin kann die Topologie durch dynamische Netzstrukturen besser erfaßt werden. Die Projektion der Eingabedaten auf die lokalen intrinsischen Unterräume ermöglicht eine weitere Verbesserung der Ergebnisse.

5.3 Ausblick

Die erreichte Genauigkeit der Posenbestimmung und Identifikation mittels neuronaler Netze ist ausreichend für viele praktische Anwendungen. Einige der verwendeten Verfahren haben aber bestimmte Nachteile, die den Anwendungsbereich einschränken. Im folgenden werden diese Beschränkungen erläutert und Verbesserungsmöglichkeiten diskutiert.

5.3.1 Die Schwellenwerte der Farbsegmentierung

Die Lokalisierung eines Gesichtes geschieht mittels Farbsegmentierung. Die Farbsegmentierung basiert auf der relativ konstanten Farbe der Gesichtshaut. Ein Nachteil dieser Methode ist die Helligkeitsabhängigkeit. Wenn sich die Helligkeit ändert, ändern sich auch die Pixelwerte im Gesichtsbereich. Dadurch ist man gezwungen, die Gesichtsfarbe durch ein breiteres Intervall zu definieren als bei konstanter Beleuchtung, was wiederum zu Ungenauigkeiten in der Gesichtslokalisierung führt. Zwar wurde eine Methode für die Helligkeitsinvarianz der Merkmalsextraktion durch die Faltung mit logarithmierten Bilddaten vorgestellt (s. Kapitel 2.3.3), diese löst aber nicht die Probleme der Farbsegmentierung.
Es gibt verschiedene Ansätze, die eine Verminderung der Helligkeitsabhängigkeit der Farbsegmentierung ermöglichen. Diese basieren meistens auf dem Modell der multiplikativen Helligkeit, d.h. der Annahme, daß die Helligkeit eine multiplikative Wirkung auf die Pixelwerte im Bild hat. Diese Annahme ist aber nur annäherungsweise gültig, so daß eine perfekte Helligkeitsinvarianz schwer zu erreichen ist.
Eine einfache Methode ist z.B. die Normierung der Farbkomponenten jedes Bildpixels, basierend auf der Annahme der multiplikativen Wirkung der Helligkeit auf die Pixelfarbwerte. Wird die Helligkeit mit h bezeichnet, so gilt für die Farbe eines Pixels (x, y):

$$\mathbf{g}(x, y) = (R, G, B) = (\tilde{R} \cdot h, \tilde{G} \cdot h, \tilde{B} \cdot h), \tag{5.1}$$

wobei $(\tilde{R}, \tilde{G}, \tilde{B})$ die helligkeitsunabhängige Farbe bezeichnet.
Aus einem in allen 3 Farbkomponenten gleich großen Helligkeitsfaktor ausgehend werden

helligkeitsinvariante Farbwerte definiert:

$$\begin{aligned}(R^*, G^*, B^*) &= \left(\frac{R}{R+G+B}, \frac{G}{R+G+B}, \frac{B}{R+G+B}\right) \\ &= \left(\frac{\tilde{R}\cdot h}{\tilde{R}\cdot h+\tilde{G}\cdot h+\tilde{B}\cdot h}, \frac{\tilde{G}\cdot h}{\tilde{R}\cdot h+\tilde{G}\cdot h+\tilde{B}\cdot h}, \frac{\tilde{B}\cdot h}{\tilde{R}\cdot h+\tilde{G}\cdot h+\tilde{B}\cdot h}\right) \\ &= \left(\frac{\tilde{R}}{\tilde{R}+\tilde{G}+\tilde{B}}, \frac{\tilde{G}}{\tilde{R}+\tilde{G}+\tilde{B}}, \frac{\tilde{B}}{\tilde{R}+\tilde{G}+\tilde{B}}\right).\end{aligned} \tag{5.2}$$

Dabei ist (R^*, G^*, B^*) helligkeitsinvariant und normiert. Durch (R, G, B) und (R^*, G^*, B^*) wird die gleiche Farbe dargestellt, da die Farben allein durch die Verhältnisse der Farbkomponenten definiert sind. Die Farbsegmentierung mit solchen vorverarbeiteten Bildpunkten ist helligkeitsinvariant.
Die helligkeitsinvarianten Farbkomponenten würden dann nur für die Gesichtslokalisierung berechnet und nicht gespeichert. Die Pixelwerte im Bild würden dabei nicht geändert, so daß die Pixel für die Bildfaltung ihre Originalfarben behalten.

Eine weitere Methode, die Farben helligkeitsinvariant darzustellen, ist die Abbildung des RGB-Farbraumes in den Yuv-Farbraum. Y gibt die Helligkeit des RGB-Farbwertes an. Die beiden anderen Komponenten u und v sind die Achsen eines kartesischen Koordinatensystems, in dem Rot, Grün und Blau ein gleichseitiges Dreieck aufspannen, wobei Schwarz im Ursprung liegt (Abbildung 5.2).

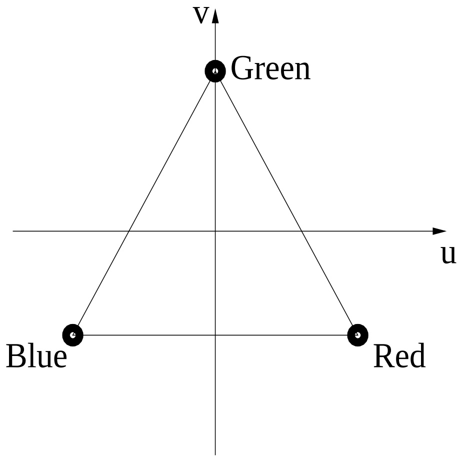


Abbildung 5.2: Die (u, v)-Ebene des Yuv-Modells.

Durch Formulierung des Yuv-Systems in Polarkoordinaten entsteht eine weitere Darstellungsmöglichkeit (HSI-System).
Helligkeitsinvarianz kann auch durch die einfache Umrechnung

$$Q_{RG} = R/(R+G)$$
$$Q_{RB} = R/(R+B)$$

in einen zweidimensionalen Farbraum erreicht werden. [LR95]

Weitere Schwierigkeiten liegen in der Bestimmung der Schwellenwerte der Farbsegmentierung. In der vorliegenden Arbeit geschah dies experimentell. Es gibt einige Methoden, die

die Suche nach optimierten Schwellenwerten erleichtern und in Kapitel 2 erläutert wurden. Die Erfahrung zeigt, daß bei 8-bit RGB-Farbbildern, deren Pixel in allen drei Bändern Werte zwischen 0 und 255 annehmen, eine Veränderung des Schwellenwertes eines einzelnen Farbbandes um einen minimalen Betrag (z.B. um 1 bis 2) zu einem deutlichen Unterschied in der Orientierungsbestimmung führt.
Für eine praktische Anwendung dieser Arbeit ist somit eine automatische Bestimmung der Farbschwellenwerte nötig (z.B. durch neuronale Netze). Eine andere Möglichkeit besteht in der Ergänzung oder Ersetzung der Farbsegmentierung durch eine andere Methode zur Gesichtslokalisierung (z.B. Kantendetektion, Bewegungsverfolgung).

5.3.2 Anwendungen mit menschlichen Gesichtern

Die Versuche in Kapitel 4 wurden mit Puppen durchgeführt. Der Grund dafür liegt darin, daß es bei Menschen sehr schwierig und aufwendig ist, eine vorgegebene Gesichtsorientierung einzustellen, um Trainingsbilder aufnehmen zu können.
Um Experimente mit menschlichen Gesichtern durchzuführen, müßte man in der Lage sein, entweder eine vorgegebene Orientierung des Kopfes einzustellen, also den Kopf in eine bestimmte Pose zu drehen (z.B. durch Befestigung einer Laserquelle am Kopf und eine Skala an der Wand, wie in [KR96]), oder Bilder mit zufälligen Orientierungen aufzunehmen und die Orientierung des Gesichtes in den Trainingsbildern zu berechnen (z.B. durch Berechnung der Gesichtsnormalen)[2]. Eine weitere Möglichkeit besteht darin, um die starr sitzende Person eine Kamera herumfahren zu lassen, die vom Gesicht Bilder aus verschiedenen Perspektiven aufnimmt. Die Kamera kann z.B. auf einem Roboterarm befestigt sein. Leider war die Reichweite des zur Verfügung stehenden Roboterarmes zu klein, um diese Methode realisieren zu können. (Aus Sicherheitsgründen sollte ein relativ großer Abstand zwischen der Kamera und dem Kopf eingehalten werden.)
Ein weiterer Faktor, der die Genauigkeit der Posenbestimmung eines menschlichen Gesichtes beeinflussen kann, ist der sich - im Gegensatz zu einer Puppe - ändernde Gesichtsausdruck. Die Auswirkungen des Gesichtsausdruckes auf die Testergebnisse ist ohne eine praktische Realisierung schwer abzuschätzen.
Das größte Problem bei den Versuchen mit einer Puppe, die von einem Roboterarm gedreht wird, ist die Trennung des Manipulators vom Kopf bei der Farbsegmentierung. Der Manipulator hat eine metallisch glänzende Oberfläche, die - vor allem bei künstlichem Licht - in bestimmten Orientierungen stark reflektierend wirkt. Dadurch ist die Farbe des Manipulators in den Bildern in verschiedenen Orientierungen unterschiedlich und fällt teilweise in den Gesichtsfarbenbereich. Deshalb müssen die Schwellenwerte der Farbsegmentierung sehr genau bestimmt werden. Bei Versuchen mit einem menschlichen Gesicht wird dieses Problem nicht mehr auftreten, so daß - bei einem einfachen Hintergrund - die Farbschwellenwerte nicht so exakt sein müßten.

[2] Auch wenn eine vollständige Methode zur Berechnung der Gesichtsorientierung in einem Bild zur Verfügung stünde, würde die Posenbestimmung durch neuronale Netze wegen des geringeren Rechenaufwandes vorzuziehen sein.

In den Versuchen wurden die Freiheitsgrade der Orientierung auf zwei Drehrichtungen beschränkt. Als Erweiterung kann die dritte Drehrichtung auch betrachtet werden.
In den Versuchen mit Puppen war die Gesicht-Kamera-Entfernung konstant. Bei der Anwendung auf menschliche Gesichter kann die Entfernung nur noch annäherungsweise konstant gehalten werden. Durch die Skalierung des Gesichtes vor der Faltung auf eine vorgegebene konstante Größe werden die Abstandsänderungen jedoch ausgeglichen.
Für einige Anwendungen ist nicht nur die Gesichtsorientierung, sondern auch die Orientierung der Augen wichtig. Durch diese beiden Komponenten (Gesichts- und Augenorientierung) wird die Richtung des Blickes festgelegt. Eine Erweiterung der vorgestellten Methoden mit der Orientierungsbestimmung der Augen würde neue Anwendungsmöglichkeiten eröffnen.

5.3.3 Anwendungen mit mehreren Objekten

Es kann natürlich nicht ausgeschlossen werden, daß die Identifikation durch neuronale Netze bei mehr als zwei Objekten fehlerhaft ist. In diesem Fall sollte das Lernen der Klassifizierung zusätzlich unterstützt werden. Ein Indiz für falsche Klassifizierungen sind Verbindungen zwischen Neuronen, die verschiedene Objekte repräsentieren. Eine Möglichkeit zur Verhinderung solcher Verbindungen wäre, von den Objekten nicht einzelne Bilder, sondern Bildsequenzen (z.B. immer 5 Bilder in einer Sequenz) aufzunehmen, wobei sich die Orientierung des Gesichtes in den Bildern kontinuierlich in kleinen Schritten ändert. Im Training sollten dann (statt der *best matching unit* und der *second best matching unit*) immer nur die *best matching unit*s für die Merkmalsvektoren zweier in einer Bildsequenz nacheinander vorkommenden Bilder miteinander verbunden werden. Hierdurch würde die Kontinuität in der Bewegung zu einer Verbesserung des Topologielernens genutzt.
Ein Nachteil des vorgestellten Verfahrens ist, daß die neuronalen Netze für jedes neues Gesicht neu trainiert werden müssen. Wünschenswert wäre hier ein Ansatz, der zwischen unterschiedlichen Gesichtern generalisiert.

Literaturverzeichnis

[BS95] P. Ballard, G. C. Stockman. Controlling a Computer via Facial Aspect. *IEEE Transaction on Systems, Man, and Cybernetics*, 25(4):669-676, 1995.

[BS95] J. Bruske, G. Sommer. Dynamic Cell Structure learns Perfectly Topology Preserving Map. *Neural Computation*, 7(4):845-865, 1995.

[BS97] J. Bruske, G. Sommer. *Intrinsic dimensionality estimation with optimally topology preserving maps.* Technical report 9703, Computer Science Institute, Christian-Albrechts-University Kiel, 1997.

[GC94] A. Gee, R. Cipolla. Determing the gaze of faces in images. *Image and Vision Computing*, 12(10):639-647, 1994.

[HH93] D. R. Hush, B. G. Horne. Progress in Supervised Neural Networks: What's New Since Lippmann. *IEEE Signal Processing Magazine*, 10:8-39, Jan. 1993.

[Jäh93] B. Jähne. *Digitale Bildverarbeitung.* Springer Verlag, 1993.

[Koh84] T. Kohonen. *Self-Organisation and Assoziative Memory.* Springer Verlag, Berlin, 1984.

[KS94] A. Koschan, K. Schlüns. *Grundlagen und Voraussetzungen für die Digitale Farbbildverarbeitung.* Technischer Bericht 94-14, Technische Universität Berlin, 1994.

[KR96] R. Kubisch, H. Ritter. Erkennung menschlicher Kopfhaltung mittels neuronaler Netze. In B. Jähne, P. Geißler, H. Haußecker, F. Hering, Hrsg., *Mustererkennung 1996*, DAGM, pp. 109-117, Springer Verlag, Berlin, 1996.

[LR95] E. Littmann, H. Ritter. Neural and Statistical Methods for Adaptive Color Segmentation - A Comparison. In G. Sagerer, S. Posch, F. Kummert, Hrsg., *Mustererkennung 1995*, DAGM, pp. 84-93, Springer Verlag, Berlin, New York, 1995.

[MN95] H. Murase, S. K. Nayar. Visual Learning and Recognition of 3-D Objects from Appearance. *International Journal of Computer Vision*, 14(1):5-24, 1995.

[Pre92] W. H. Press. *Numerical Recipies in C.* Cambridge University Press, 1992.

[RMS91] H. Ritter, M. Martinetz, K. Schulten. *Neuronale Netze - Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke.* Addison-Wesley Verlag, Bonn, 1991.

[TP91] M. Turk, A. Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71-86, 1991.

Danksagung

Ich möchte jedem danken, der mich während meines Studiums mit Rat und Tat unterstützt hat. Mein besonderer Dank gilt meiner Familie. Ohne die liebevolle und tatkräftige Unterstützung meines Mannes Oliver wäre mein Studium undenkbar gewesen. Meine Tochter Judith hat meine Arbeit sicherlich nicht immer erleichtert. Ihre Liebe und Nähe möchte ich trotzdem nie missen. Ich bin sehr dankbar für die Liebe und Hilfsbereitschaft meiner Schwiegereltern Margit und Jürgen, die immer für uns da waren. Mein Dank gilt auch meinen Eltern, die durch meine Erziehung die Grundsteine meines Studiums gelegt haben.

Ich danke meinem Betreuer Jörg Bruske, der bei Problemen immer Zeit für mich hatte. Durch seine kreative Kritik, Hilfsbereitschaft und Geduld hat er sehr viel zu der Entstehung dieser Arbeit beigetragen.

Schließlich möchte ich allen Freunden und Bekannten danken, die versucht haben, meine unzähligen grammatikalischen Fehler in der Diplomarbeit zu beseitigen: Frank, Christiane, Michael, Kirstin, Charles und Dieter.

Erklärung:

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und ausschließlich die angegebenen Hilfsmittel und Quellen verwendet habe.

Kiel, den