Visuell basiertes Greifen gestützt auf neuronale Netze

vorgelegt von

Jan Tröster

Kiel, im September 1997

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 und Dr. Josef Pauli

Zusammenfassung

Gegenstand dieser Arbeit ist die Entwicklung eines *visuell basierten Greifsystems*, welches, rein auf Bildinformationen gestützt, gelernte Objekte erkennen und greifen kann. Im Vordergrund steht dabei die *Objekterkennung und Posenbestimmung*, welche auf Grundlage der *photometrischen Eigenschaften* der Objekte in einem Computerbild realisiert werden. Hierfür geschieht eine Faltung der Bilddaten an festen Ortspunkten mit *orientierungsselektiven Bandpaßfilteroperatoren*, um so *Merkmalsvektoren* aus den Ansichten zu extrahieren. Die Filteroperatoren sind dabei so gewählt, daß verschiedene Merkmalsvektoren des selben Objektes eine *Mannigfaltigkeit* im Merkmalsraum beschreiben, während Merkmalsvektoren von verschiedenen Objekten dagegen auf unterschiedlichen Mannigfaltigkeiten liegen.
Die Merkmalsvektoren sind die Eingabevektoren für neuronale Netze, die auf die Posenbestimmung der einzelnen Objekte trainiert werden. Das neuronale Netz bedient sich bei der Approximation einer *topologischen Karte*, welche die Topologie der jeweiligen Mannigfaltigkeit beschreibt. Damit ist es möglich, die Approximation direkt in dem niederdimensionalen Unterraum durchzuführen, wodurch einerseits die Berechnung effizienter ist, andererseits die Approximationsgüte verbessert wird, indem unter anderem Rauschen in den Daten erkannt und herausgefiltert werden kann.
Die *Objektlokalisation* bedient sich ebenfalls allein der Bilddaten, indem die Translation der Position des Objektes zwischen einer Modell- und der Szenenansicht im Bild bestimmt und diese in Weltdaten umgerechnet wird.
Das entwickelte Greifsystem ist in der Lage, die Klassifizierung, Posenbestimmung und *Lokalisation* für gelernte Objekte in einer Genauigkeit durchzuführen, die ein sicheres Greifen des jeweiligen Objektes zuläßt.
In dieser Arbeit werden schrittweise die einzelnen Komponenten des Systems vorgestellt, ihre Komposition zu einem Gesamtsystem erläutert und die Leistungsfähigkeit des Gesamtsystems an einigen Versuchsreihen evaluiert.

Inhaltsverzeichnis

Kapitel 1

Einführung

Die Automatisierung von Produktionsvorgängen in der Industrie hält in der heutigen Zeit immer mehr Einzug. Roboter zu konstruieren, die in der Lage sind, selbständig Gegenstände von einem Ort zu anderen zu transportieren, exakte Bohrungen an Werkstücken vorzunehmen oder Formen aus Werkstoffen herauszuschneiden, stellt kein Problem mehr dar. Dabei sind den Systemen aber die für den jeweiligen Arbeitsvorgang anzufahrenden Positionen und die einzunehmenden Greifstellungen a priori exakt vorgegeben. Im Forschungsbereich der kognitiven Systeme hingegen sind Systeme interessant, die in der Lage sind, diese Greifaufgabe autonom durchzuführen, das heißt zum Beispiel, das System soll „sehen, was es tut". Man stelle sich ein Robotersystem vor, welches in seinem Arbeitsbereich beliebig positionierte Objekte greifen kann. Es soll mit einem Kamerasystem ausgestattet sein, welches Bildinformationen von einem Objekt in der Szene liefert. Das System muß aus den Bildinformationen die Geometrie des Objektes rekonstrieren und anhand der Geometrie eine stabile Greifposition am Objekt finden. Blake [1] stellt in seiner Arbeit ein Verfahren vor, welches Greifpositionen an flachen Objekten berechnet. Eine Kamera betrachtet die Objekte von oberhalb, so daß die Rekonstruktion der Geometrie sich auf ein zweidimensionales Problem reduziert. Es wird eine geschlossene Kontur des Objektes approximiert und anhand dieser eine stabile Greifstellung berechnet. Für die dreidimensionale Rekonstruktion eines Objektes reicht jedoch eine Ansicht nicht aus, man benötigt Ansichten aus verschiedenen Positionen der Kamera. Bei Taylors Arbeit [19] ist die Kamera am Manipulator fixiert, und die Rekonstruktion sowie die Suche einer geeigneten Greifstellung geschieht über die Bewegung der Kamera in verschiedene Betrachtungspositionen zum Objekt. Das Finden von Greifstellungen an unbekannten Objekten, allein basierend auf visuellen Daten, ist generell jedoch ein schwieriges Problem, denn man erhält keinerlei Informationen über die materielle Beschaffenheit eines Objektes, die auf die Stabilität einer Greifstellung ebenfalls einen großen Einfluß hat.
Sind die zu greifenden Objekte von vornherein bekannt, so können mögliche Greifstellungen an den Objekten anhand eines Modells a priori vorgegeben werden. Das System hat dann die Aufgabe, aus einer Szenenansicht das aufgenommene Objekt als eines der bekannten Objekte zu erkennen und seine momentane Lage zu bestimmen und damit auch die

Lage der Greifposition. Eine solche *modellbasierte Objekterkennung und Posenbestimmung* unterteilt sich in zwei Phasen. In der *Trainingsphase* werden aus verschiedenen Ansichten des jeweiligen Objektes Merkmale extrahiert, die geeignet sind, das Objekt in jeglicher Lage zu erkennen und seine Orientierung zu bestimmen. Diese Merkmale werden in einer *Modelldatenbank* gespeichert. Die Phase der *Objekterkennung* ist dann der Vergleich von Merkmalen der Modelldatenbank mit Merkmalen, die aus einer aktuellen Szenenansicht extrahiert wurden. Als Merkmale verwendet man häufig geometrische Eigenschaften des Objektes wie Ecken, Kanten, Winkel usw. Dieses Verfahren bietet sich besonders an, wenn ein geometrisches Modell der Objekte z.B. in Form von CAD-Daten existiert. Der Nachteil der Verwendung von geometrischen Eigenschaften ist, daß viele weitere Informationen, die sich in einer Ansicht befinden, verloren gehen.
Bei der Wahl der *photometrischen* Eigenschaften wird dagegen die gesamte Erscheinung eines Objektes im Bild betrachtet. Murase und Nayar [14] erzeugen als Grundlage für die Objekterkennung und Posenbestimmung eine Menge von Ansichten eines Objektes in verschiedenen Orientierungen. Sie sprechen dabei nicht von Bildung eines Modells, sondern führen den Begriff des *Eigenraums* ein. Jede Ansicht wird als Vektor aufgefaßt, dessen Komponenten die Pixelwerte des Bildes sind. Auf diesen Vektoren wird eine Hauptachsenanalyse durchgeführt, deren Ergebnis eine Menge von Eigenvektoren ist. Die Eigenvektoren, die mit den größten Eigenwerten korrespondieren, spannen den Eigenraum auf. Der Pixelvektor jeder Ansicht wird nun in den Eigenraum projiziert, so daß der projizierte Punkt die Ansicht im Eigenraum repräsentiert. Die Eigenschaft des Eigenraums ist zum einen, daß die Projektionspunkte von Ansichten, die das Objekt in ähnlichen Orientierungen zeigen, im Eigenraum dicht beieinander liegen. Zum anderen bilden die Projektionen aller Ansichten einen Unterraum im Eigenraum, eine sogenannte *Mannigfaltigkeit*. Ansichten von unterschiedlichen Objekten wiederum erzeugen unterschiedliche Mannigfaltigkeiten. Die Mannigfaltigkeiten werden über eine Splineinterpolation approximiert. Um nun für ein Objekt in der Szene die Objekterkennung und Posenbestimmung durchzuführen, wird die Ansicht ebenfalls in den Eigenraum projiziert. Die Mannigfaltigkeit, auf die die Ansicht abgebildet wird, klassifiziert das Objekt, die Position in der Mannigfaltigkeit gibt die gesuchte Orientierung an.

1.1 Beitrag der Arbeit

In dieser Arbeit wird ein System entwickelt, das in der Lage ist, flache Objekte, die das System vorher gelernt hat, zu greifen. Man stelle sich vor, daß das System an einem Fließband installiert ist, von dem es die Objekte mittels einer Roboterhand herunternimmt und sortiert. Das Fließband stoppt, wenn sich ein Objekt in einem bestimmten Bereich befindet. Auf diesen Bereich schaut eine Kamera von oben herab. Das Objekt kann sich an einem beliebigen Ort in dem Bereich befinden und besitzt in seiner Orientierung einen Freiheitsgrad, nämlich eine Rotation um eine vertikale Achse.
Für die Objekterkennung und Posenbestimmung orientiert sich das gewählte Vorgehen

an dem Verfahren Murases und Nayars. Es wird jedoch nicht ein Eigenraum verwendet, sondern ein anderer, spezieller Raum, der *Merkmalsraum*. Die Projektion in den Merkmalsraum geschieht durch Faltung der Ansichten an festgelegten Ortspunkten mit verschiedenen Faltungsoperatoren. Als Operatoren dienen hierfür *Gaborfilter*. Sie gehören in die Klasse der Bandpaßfilter, wodurch sie sich zur Objekterkennung eignen, da unterschiedliche Objekte unterschiedliche Frequenzen im Bild erzeugen. Weiterhin besitzen sie Orientierungsselektivität, womit sie einen Beitrag zur Posenbestimmung leisten. Die Faltungsantworten der Operatoren bilden den Merkmalsvektor im Merkmalsraum.
Die Berechnung der Orientierungen geschieht durch *neuronale Netze*. Die Netze erhalten als Eingabe die Merkmalsvektoren. Dabei wird ein neuronaler Netztyp entwickelt, der sich einer *topologischen Karte* bei der Approximation bedient. Die topologische Karte beschreibt die durch die projizierten Ansichten erzeugte Mannigfaltigkeit und damit die Topologie der Eingabedaten des neuronalen Netzes. Es wird sich zeigen, daß die Verwendung einer topologischen Karte der Eingabedaten Vorteile im Hinblick auf den Approximationsvorgang des neuronalen Netzes besitzt. So geschieht die Approximation innerhalb des niederdimensionalen Unterraums, wodurch die Berechnung effizienter wird. Zudem läßt sich die Approximationsgüte verbessern, weil unter anderem Rauschen in den Daten erkannt werden kann. So liegen verrauschte Daten außerhalb der Mannigfaltigkeit, aber durch Rückprojektion der Daten in den Unterraum kann das Rauschen herausgefiltert werden.
Die Posenbestimmung geschieht für jedes Objekt über ein eigenes neuronales Netz, wobei die Netze später zu einem großen Netzkonstrukt zusammengefügt werden.

1.2 Aufbau der Arbeit

Im zweiten Kapitel folgt nach einer kleinen Einführung in die Theorie der neuronalen Netze eine Beschreibung, wie die Topologie einer Punktemenge bestimmt werden kann und für den Approximationsvorgang eines neuronalen Netzes genutzt wird. Schritt für Schritt wird das Zusammenspiel von topologischer Karte und neuronalem Netz dabei verbessert. Kapitel drei stellt die verwendeten bildverarbeitenden Mittel vor, die die Aufgabe haben, die Merkmalsvektoren aus den Bilddaten zu berechnen. Im vierten Kapitel wird das Gesamtsystem zur Lösung der Greifaufgabe vorgestellt. Dort wird auch das Verfahren zur Lokalisation der Objekte in der Szene beschrieben. Das fünfte Kapitel liefert dann die Resultate des Systems in einigen Versuchsreihen. Kapitel sechs widmet sich schließlich einer Zusammenfassung der erzielten Ergebnisse, einer Diskussion und dem Ausblick auf die Integration dieser Arbeit.

Kapitel 2

Neuronale Architekturen

2.1 Einführung

Der Begriff „*künstliche neuronale Netze*“ stammt aus dem Bereich der Neuroinformatik. Künstliche neuronale Netze ahmen biologische neuronale Netze als informationsverarbeitende Systeme nach. Sie liefern eine alternative mathematische Berechnungsmöglichkeit für Funktionen. Ein neuronales Netz ist also eine Art „Abbildungsmaschine“, die eine Funktion F modelliert [17]. Die Fähigkeit, eine bestimmte Funktion zu berechnen, ist dem Netz jedoch nicht a priori vorgegeben, sondern es lernt die Funktion über Beispiele. Beispiele sind im mathematischen Sinne Paare $(\boldsymbol{x}, \boldsymbol{y})$, bei denen für eine Eingabe $\boldsymbol{x}$ die Ausgabe $\boldsymbol{y}$ bekannt ist, also $\boldsymbol{y} = \mathrm{F}(\boldsymbol{x})$ gilt. Es sind sogenannte *Abtastpunkte*.
Grundlage eines jeden neuronalen Netzes sind die Neuronen, oder auch Knoten genannt. Jedes *Neuron* ist eine Art Teilfunktion des Gesamtnetzes. Eine Eingabe in das Netz, auch *Stimulus* genannt, erzeugt eine Aktivierung an den Neuronen, woraus das jeweilige Neuron eine Ausgabe berechnet. Aus den Ausgaben der Neuronen berechnet sich schließlich die eigentliche Gesamtausgabe des Netzes. Das Einfügen von Neuronen und die schrittweise Adaption der Teilfunktionen durch Abtastpunkte ist der Lernvorgang des Netzes. Geschieht der Lernvorgang völlig autonom, so spricht man von *selbstorganisierendem Lernen des Netzes.*
Man unterscheidet weiterhin zwischen statischen und rekurrenten Netzwerken. Bei den statischen Netzwerken berechnet sich die Ausgabe allein aus der aktuellen Eingabe, während ein rekurrentes Netz auch vorangegangene Ein- und Ausgaben in die aktuelle Berechnung miteinbezieht [7]. Ein klassischer Vertreter der statischen Netze sind die Radialen Basisfunktionen (RBF), sie liefern die Grundstruktur für alle in dieser Arbeit verwendeten neuronalen Netztypen.

2.2 Radiale Basisfunktionen (RBF)

Radiale Basisfunktionen sind neuronale Netze, die aus zwei aufeinanderfolgenden Schichten bestehen. Die erste Schicht, die sogenannte *verdeckte Schicht*, beinhaltet die Neuronen des neuronalen Netzes. Jedes Neuron ist über eine *Basisfunktion* definiert. Im Fall des RBF-Netzes ist dies eine radialsymmetrische Funktion im Eingaberaum mit einem *Zentrum* $\boldsymbol{c_i}$, dem Ortsvektor der Funktion. Die Funktion liefert zu einem Stimulusvektor $\boldsymbol{x}$ einen Ausgabewert a_l, die sogenannte *Aktivierung* des Knotens bezüglich des Stimulus'. Diese Aktivierung ist größer je dichter der Eingabevektor an dem betreffenden Knoten liegt. Als Funktionen, die diese Aktivierung berechnen, werden meist Gaußfunktionen verwendet. Sei also $\boldsymbol{x} \in \mathbb{R}^n$ der genannte Stimulus oder Eingabevektor in das Netz, dann berechnet sich die Aktivierung a_i von Knoten i durch:

$$a_i = \exp\left(-\frac{\| \boldsymbol{x} - \boldsymbol{c_i} \|^2}{\sigma_i^2}\right), \tag{2.1}$$

wobei σ_i der sogenannte *Normalisierungsparameter* ist. Die Aktivierung nimmt Werte zwischen 0 und 1 an, wobei der Wert 1 bedeutet, daß Zentrum und Eingabevektor koinzidieren. Mit zunehmendem Abstand zwischen Zentrum und Eingabevektor verringert sich die Aktivierung. Zwei Eingabevektoren, die denselben Abstand zum Zentrum haben, erzeugen dieselbe Aktivierung, daher stammt der Name „Radiale Basisfunktion“. Jedem Neuron ist zusätzlich ein *Ausgabevektor* $\boldsymbol{o_i} \in \mathbb{R}^m$ zugeordnet. Die Ausgabevektoren symbolisieren die zweite Schicht, die *Ausgabeschicht*. Hier wird nun die Ausgabe $\boldsymbol{y_i} \in \mathbb{R}^m$ jedes Knotens, abhängig von dessen Aktivierung, nach

$$\boldsymbol{y_i} = a_i \boldsymbol{o_i} \tag{2.2}$$

berechnet. Die eigentliche Ausgabe des gesamten Netzes $\boldsymbol{y} \sim F(\boldsymbol{x})$ berechnet sich dann aus der Summe der Ausgaben der einzelnen Knoten mit

$$\boldsymbol{y} = \sum_{i=1}^{N} \boldsymbol{y_i}, \tag{2.3}$$

wobei N die Anzahl der Knoten im Netz ist.

Die Konfiguration eines solchen Netzes wird also durch verschiedene Parameter beschrieben zum einen durch die Anzahl der Neuronen, zum anderen durch die Zentren $\boldsymbol{c_i}$, die Ausgabevektoren $\boldsymbol{o_i}$ und die Normierungsparameter σ_i der Neuronen. Die Parameter können nun entweder a priori vorgegeben oder in der sogenannten Trainingsphase oder Lernphase berechnet werden.

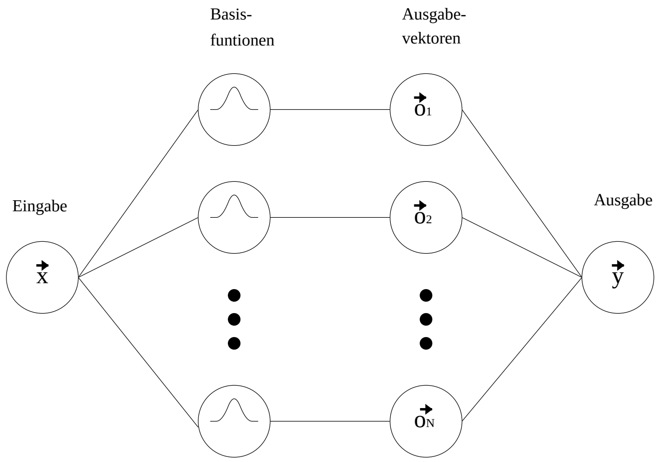

Abbildung 2.1: Schema eines RBF-Netzes

2.3 Trainieren von neuronalen Netzen

Die *Lernphase* von neuronalen Netzen stützt sich auf die T Abtastpunkte $(\boldsymbol{x_t}, \boldsymbol{y_t}) \in \mathbb{R}^n \times \mathbb{R}^m$ einer Funktion $F : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, für die gelten:

$$\boldsymbol{y_t} = F(\boldsymbol{x_t}) \qquad t = 1, \ldots, T \tag{2.4}$$

Die Menge der Abtastpunkte wird auch als *Trainingsmenge* bezeichnet, wobei $\boldsymbol{x_t}$ den Eingabevektor und $\boldsymbol{y_t}$ den Ausgabevektor eines *Trainingsdatums* bezeichnet. Abtastpunkte geben also Vektoren im Eingaberaum an, für die der Funktionswert a priori bekannt ist. Die Berechnung der Parameter des Netzes geschieht sukzessive in 2 Phasen. In der ersten Phase werden die Zentren der Knoten und in manchen Fällen auch die Normierungsparameter bestimmt, in der zweiten Phase dann die Ausgabevektoren der einzelnen Knoten in Abhängigkeit von den in der ersten Phase bestimmten Parametern.

2.3.1 Trainieren der ersten Schicht

Es gibt verschiedene Verfahren zur Bestimmung der Neuronenzentren. Geht man von einer festgewählten Anzahl k von Neuronen aus, so wird in der Literatur häufig der *k-means*-Algorithmus zur Bestimmung der Knotenzentren verwendet. Das Verfahren teilt die Abtastpunkte durch Klassifizierung ihrer Eingabevektoren in k Cluster ein. Die Schwerpunkte der Cluster wählt man dann als Zentren der Knoten. Man kann jedoch das Netz auch schrittweise aufbauen, indem man iterativ solange Knoten neu in das Netz einfügt und es anschließend konfiguriert, bis eine geforderte Approximationsgüte des Netzes erreicht ist. Man spricht von einem *wachsenden neuronalen Netz* [5]. Die Approximationsgüte ist dabei über den durchschnittlichen Approximationsfehler re in den Abtastpunkten definiert, das heißt über die durchschnittliche Differenz zwischen berechneter Ausgabe $\tilde{F}(\boldsymbol{x_i})$ und gewünschter Ausgabe $y_i = F(\boldsymbol{x_i})$. Es ist

$$re = \frac{\sum_{i=1}^{M} \parallel \tilde{F}(\boldsymbol{x_i}) - F(\boldsymbol{x_i}) \parallel^2}{M} = \frac{\sum_{i=1}^{M} \parallel \tilde{F}(\boldsymbol{x_i}) - \boldsymbol{y_i} \parallel^2}{M}. \tag{2.5}$$

Je kleiner der durchschnittliche Fehler re ist, desto besser ist die Approximationsgüte des Netzes bezüglich der Abtastpunkte. In dieser Arbeit werden die wachsenden Netzwerke verwendet. Es sollen dabei die Positionen der schon eingefügten Neuronen in jedem Schritt nicht völlig neu berechnet werden, sondern lediglich eine günstige Position für das neu einzufügende Neuron gefunden werden. Für die Suche nach der Position gibt es zwei Strategien. Beide Strategien orientieren sich dabei an den Abtastpunkten. Die erste Strategie ist, das neue Neuron in einem Bereich einfügen, in dem sich im Verhältnis zur Anzahl von Abtastpunkten nur wenige Neuronen befinden. Diese Strategie verteilt die Neuronen gleichmäßig über die Bereiche, in denen sich Abtastpunkte befinden. Die zweite Strategie ist dagegen, ein neues Neuron in dem Bereich einzufügen, in dem der Approximationsfehler bezüglich der Abtastpunkte am größten ist. Um beide Strategien zu konkretisieren, soll zunächst der Begriff des „naheliegendsten Knotens“ eingeführt werden, der im weiteren Verlauf der Arbeit immer wieder seine Anwendung findet.

Definition 1 *Sei N die Menge der Knoten in einem Netz G, $S = \{\boldsymbol{c_i} \in I\!R^n \mid i = 1, \ldots, \text{Anzahl der Knoten}\}$ die Menge der Zentren zu den Knoten in N, $\boldsymbol{x} \in I\!R^n$. Dann heißt*
$n_i \in N$ naheliegendster Knoten *zu* $\boldsymbol{x} \quad \Leftrightarrow \quad \parallel \boldsymbol{x} - \boldsymbol{c_i} \parallel \leq \parallel \boldsymbol{x} - \boldsymbol{c_j} \parallel \forall \boldsymbol{c_j} \in S.$
Der naheliegendste Knoten wird im weiteren auch als bmu *(best matching unit) bezeichnet.*

Mit Hilfe der Definition ist jetzt eine Zuordnung der einzelnen Abtastpunkte zu den Neuronen des Netzes möglich. Es lassen sich dann die beiden Strategien zur Bestimmung der Position des einzufügenden Neurons folgendermaßen formalisieren:

1. Quantisierung der Entfernungsdifferenzen im Eingaberaum: Man summiere für jedes Neuron die Entfernungen zwischen Zentrum und Eingabevektor der ihm zugeordneten Abtastpunkte auf und bestimme anschließend das Neuron mit dem größten Quantisierungsfehler. In dem lokalen Bereich dieses Neurons wird das neue Neuron eingefügt.

2. Quantisierung der Approximationsfehler: Man summiere für jedes Neuron die Approximationsfehler der ihm zugeordneten Abtastpunkte und bestimme anschließend das Neuron mit dem größten Quantisierungsfehler. In dem lokalen Bereich dieses Neurons wird das neue Neuron eingefügt.

Die Erwartung war es, daß sich letztere Strategie als generell sinnvoller erweisen würde, also ein neues Neuron in dem Bereich einzufügen ist, in dem die Funktion am schlechtesten approximiert wird. Diese Annahme hat sich in den Versuchen zu dieser Arbeit

nicht bestätigt. Die Begründung liegt darin, daß sich die Verwendung einer topologischen Karte für die Approximation eines Netzes als sehr nützlich erwiesen hat, die Topologie einer Punktemenge sich jedoch exakter berechnen läßt, wenn die Neuronen des Netzes gleichmäßig über die Bereiche, in denen die Punkte liegen, verteilt sind. Dies wird bei der ersten Strategie gewährleistet, nicht jedoch bei der zweiten. Daher wird in dieser Arbeit durchgehend die erste Strategie verwendet.
Der folgende Algorithmus gibt nun ein Verfahren an, durch welches die gesuchte Position nach der ersten Strategie exakt bestimmt wird [1].

Algorithmus (neuer Knoten)

Vorbedingung: G konfiguriertes Netz mit Knotenmenge N, $T \subseteq (\mathbb{R}^n \times \mathbb{R}^m)$ Trainingsmenge, $E = \{\boldsymbol{x_i} \mid (\boldsymbol{x_i}, \boldsymbol{y_i}) \in T\}$ Menge der Eingabevektoren der Trainingsdaten.

for all $n_i \in N$ **do**
  max_error[n_i] = 0, total_error[n_i] = 0;
for all $\boldsymbol{x_i} \in E$ **do**
  bestimme bmu von $\boldsymbol{x_i}$
  total_error[n_{bmu}] += $\| \boldsymbol{x_i} - \boldsymbol{c_{bmu}} \|$;
  if max_error[n_{bmu}] $\leq \| \boldsymbol{x_i} - \boldsymbol{c_{bmu}} \|$;
    max_error[n_{bmu}] = $\| \boldsymbol{x_i} - \boldsymbol{c_{bmu}} \|$;
    max_error_index[n_{bmu}] = i;
  end if;
end for;
wähle Knoten n_{max} mit max_error[n_{max}] $\geq$ max_error[n_i] $\forall n_i \in N$;
wähle $\boldsymbol{x_{max\_error\_index[n_{max}]}}$ als Zentrum des neuen Knotens;

Der Algorithmus summiert für jeden Knoten die Abstände zu den ihm zugeordneten Trainingsdaten auf und ermittelt den Knoten mit der größten Summe der Abstände. Für jeden Knoten wird zusätzlich das Trainingsdatum gespeichert, welches den größten Abstand zum jeweiligen Knoten besitzt. Der Eingabevektor für dieses Trainingsdatum wird als Zentrum des neu einzufügenden Knotens gewählt. Um alle Trainingsdaten bei Adaption der Zentren miteinzubeziehen, kann anschließend eine Variante des k-means Algorithmus verwendet werden. Der folgende Algorithmus *cluster-nearest-neighbour* bestimmt für jedes Trainingsdatum den naheliegendsten Knoten und verschiebt diesen Knoten in Richtung des Trainingsdatums. Man versucht damit, die Knoten in den Verlauf der Trainingsdaten zu ziehen [5].

[1] Ein Algorithmus nach der zweiten Strategie ist jedoch nahezu identisch, es müßte lediglich $\| \boldsymbol{x_i} - \boldsymbol{c_{bmu}} \|$ durch $\| \tilde{F}(\boldsymbol{x_i}) - (\boldsymbol{y_i}) \|$ ersetzt werden.

Algorithmus (cluster-nearest-neighbour)

Vorbedingung: G konfiguriertes Netz mit Knotenmenge N, $T \subseteq (\mathbb{R}^n, \mathbb{R}^m)$ Trainingsmenge, $E = \{\boldsymbol{x_i} \mid (\boldsymbol{x_i}, \boldsymbol{y_i}) \in T\}$ Menge der Eingabevektoren der Trainingsdaten.

repeat
  for all $n_i \in N$
    $\boldsymbol{new\_c_i} = 0$
    $count_i = 0$
  end for;
  for all $\boldsymbol{x_i} \in E$
    bestimme bmu von $\boldsymbol{x_i}$;
    $\boldsymbol{new\_c_{bmu}} + = \boldsymbol{x_i}$;
    $count_{bmu} + +$;
  end for;
  for all $n_i \in N$
    $c_i = \frac{\boldsymbol{new\_c_i}}{count_i}$;
until keine weiteren Änderungen in den Zentren;

Normierungsparameter

Der Normierungsparameter σ bestimmt den Radius der Radialen Basisfunktion eines Knotens. Bei der Gaußfunktion entspricht der Normierungsparameter der Varianz. Je größer σ gewählt ist, desto kleiner wird die Aktivierung, die ein Stimulus bei gleichbleibender Entfernung zu einem Knoten erzeugt. Mit den Normierungsparametern regelt man die Überlagerung der Basisfunktionen der Knoten. In einigen Arbeiten geht man von festen Normierungsparamtern aus. Im Verlauf dieser Arbeit werden Verfahren vorgestellt, die auch diesen Parameter adaptieren und somit für die jeweilige Konfiguration anpassen.

2.3.2 Trainieren der zweiten Schicht

Die Berechnung der Parameter der zweiten Schicht realisiert nun die eigentliche Approximation der Funktion. Die Parameter der ersten Schicht liefern dabei ein Grundgerüst, auf das sich die Approximation stützt. Grundlage für diese Phase ist wiederum die Menge der Trainingsdaten. Man unterscheidet zwischen sequentiellen und Offline-Verfahren. In den *sequentiellen Verfahren* wirkt jedes Trainingsdatum sukzessive auf die Parameter ein, man spricht von schrittweisem Lernen. Ein typisches Verfahren hierfür ist der *least-mean-square*-Algorithmus (LMS)[7]. In einem *Offline-Verfahren* gehen alle Trainingsdaten gleichzeitig in die Berechnung ein, die Berechnung geschieht damit also in einem Schritt. Letztere Variante wird in dieser Arbeit verwendet.

Offline-Berechnung der Parameter

Gegeben ist die Menge $T = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \mid i = 1, \ldots, t\}$ von Trainingsdaten. G ist ein Netz mit Knoten $n_i = (\boldsymbol{c_i}, \sigma_i)$, $i = 1, \ldots, k$, welches die erste Trainingsphase durchlaufen hat. $A \in \mathbb{R}^{t,k}$ ist die sogenannte *Aktivierungsmatrix*, d.h. die Komponente a_{ij} bezeichnet die Aktivierung von Knoten j bei Eingabe von $\boldsymbol{x_i}$ in das Netz. Da für jeden Abtastpunkt $\boldsymbol{x_i}$ die Ausgabe $\boldsymbol{y_i}$ bekannt ist, muß für jedes Trainingsdatum die Gleichung

$$\boldsymbol{y_i} = \sum_{j=1}^{k} a_{ij} \boldsymbol{o_j} \tag{2.6}$$

erfüllt sein. Faßt man die t Gleichungen zu einem Gleichungssystem zusammen, erhält man folgendes lineares Ausgleichsproblem zur Bestimmung der Ausgabevektoren $\boldsymbol{o_i}$ bei komponentenweiser Betrachtung der Vektoren:

$$\begin{pmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & & \vdots \\ a_{t1} & \cdots & a_{tk} \end{pmatrix} \begin{pmatrix} o_{11} & \cdots & o_{1m} \\ \vdots & & \vdots \\ o_{k1} & \cdots & o_{km} \end{pmatrix} = \begin{pmatrix} y_{11} & \cdots & y_{1m} \\ \vdots & & \vdots \\ y_{t1} & \cdots & y_{tm} \end{pmatrix} \tag{2.7}$$

Es ist o_{ij} bzw. y_{ij} die j-te Komponente des i-ten Ausgabevektors.
Ist die Anzahl der Trainingsdaten kleiner als die Anzahl der Knoten, so ist das Gleichungssystem unterbestimmt, es existiert dann keine eindeutige Lösung des Gleichungssysyems 2.7, sondern ein Lösungsraum. Für das Netz bedeutet dies, daß es die Funktion in den Abtastpunkten zwar exakt berechnen kann, aber aufgrund der Vielzahl von Lösungen nicht zwangsläufig eine Konfiguration erreicht, die auch auf Eingaben, die zu den Trainingsdaten verschieden sind, ein vernünftige Approximation liefert. Man spricht von der Situation, daß das Netz aufgrund der zu hohen Freiheitsgrade im Verhältnis zu der Anzahl von Bedingungen die Trainingsmenge „auswendig lernt". Hier ergibt sich also eine obere Schranke für die Größe des Netzes in Abhängigkeit von der Trainingsmenge. In der Regel ist das Gleichungssystem 2.7 jedoch überbestimmt, d.h. man erhält ein lineares Ausgleichsproblem. Die beste Approximation ergibt sich dann aus der Lösung mit den kleinsten Fehlerquadraten (least squares solution). Die Güte der Approximation hängt dabei von der Kondition der Aktivierungsmatrix ab. Um unabhängig von der Kondition zu sein, kann die Matrix A mittels Singulärwertzerlegung (Singular Value Decomposition SVD) in das Produkt von Matrizen $U * W * V$ zerlegt werden, wodurch das Problem der schlechten Konditionierung gelöst wird [15].

2.4 Topologiebetrachtungen

Im bisherigen Vorgehen wurde immer der gesamte Eingaberaum $\mathbb{R}^n$ betrachtet. Die Parameter des Netzes wurden über die Abtastpunkte bestimmt, aber für die Eingabe in das Netz wurde ein beliebiger Vektor des Eingaberaums zugelassen. In vielen Anwendungen verhält es sich aber so, daß die Eingabevektoren nicht den gesamten Eingaberaum aufspannen, sondern einen Unterraum bilden mit einer Dimension $s < n$. Wissen über die Topologie und die daraus abgeleitete intrinsische Dimensionalität des Unterraums ist von großem Nutzen für die Approximationsgüte neuronaler Netze [4]. Man kann die Approximation, anstatt den gesamten Eingaberaum als Definitionsbereich zuzulassen, auf den niederdimensionalen Unterraum beschränken, indem die Neuronen nur innerhalb der Unterraums plaziert werden. Die Abstandsberechnung zwischen Neuronen und Eingabedaten, aus der sich die Aktivierung der Neuronen ergibt, geschieht dann ebenfalls nur innerhalb des Unterraums, das heißt, Abstandskomponenten außerhalb der Unterraums werden als Rauschen in den Eingabedaten gewertet und herausgefiltert. Die vorliegende Arbeit wird zudem in einem Verfahren zeigen, daß es möglich ist, die gesuchte Funktion innerhalb des Unterraums linear zu approximieren.
Die Nachbarschaftsbeziehungen zwischen den Neuronen, die sich aus der berechneten Topologie ergeben, können wiederum ausgenutzt werden, in dem nicht mehr alle Neuronen zur Berechnung einer Ausgabe betrachtet werden, sondern nur diejenigen in der regionalen Nachbarschaft eines Stimulus. Es wird damit der Rechenaufwand bei der Approximation vermindert.
Es gibt zwei wesentliche Vorgehensweisen bei der Suche nach der intrinsischen Dimensionalität. Bei der *globalen Vorgehensweise* versucht man, ein globales Maß durch Einbeziehen aller Daten zu minimieren. Dagegen wird bei der *lokalen Vorgehensweise* versucht, die intrinsische Dimensionalität direkt über Nachbarschaftsbeziehungen zwischen den Daten in lokalen Bereichen zu bestimmen, ohne die Daten in den Unterraum zu projizieren[4]. In dieser Arbeit wird letzteres Vorgehen gewählt. Um den Begriff der Nachbarschaftsbeziehungen näher zu beschreiben, wird der Begriff der Optimal Topologie erhaltenden Karte (Optimal Topology Preserving Map OTPM) eingeführt.

2.4.1 Optimal Topologie erhaltende Karte (OTPM)

Die *Optimal Topologie erhaltende Karte* leitet sich von Martinetz' *Perfekt die Topologie erhaltende Karte (PTPM)* [12] ab und beschreibt die Nachbarschaftsbeziehungen einer Knotenmenge bezüglich einer gegebenen Punktemenge. Der einzige Grund für die Einführung eines neuen Begriffs liegt darin, daß Martinetz davon ausgeht, daß die Vektoren im betrachteten Raum „dicht" liegen [12]. Diese Bedingung kann bei den Trainingsdaten im allgemeinen nicht eingehalten werden.
Das neuronale Netz wird von nun an in einen Graphen eingebettet, wobei die Knoten des Graphens die bisher betrachteten Neuronen des Netzes repräsentieren, zusätzlich aber auch Kanten zwischen den Knoten verlaufen können.

Definition 2 *Seien* $T = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \mid \boldsymbol{x_i} \in I\!R^n, \boldsymbol{y_i} \in I\!R^m i = 1, \ldots, t\}$ *die Menge der Trainingsdaten,* $T_e = \{\boldsymbol{x} \mid (\boldsymbol{x}, \boldsymbol{y}) \in T\}$ *die Eingabevektoren der Trainingsdaten, sei* $S = \{\boldsymbol{c_i} \in I\!R^n \mid i = 1, \ldots, N\}$ *eine Menge von Vektoren im Eingaberaum.*
Man bezeichnet den ungerichteten Graphen $G = (S, E)$ *als eine Optimal Topologie erhaltende Karte von* S *bezogen auf die Trainingsmenge* T*, wenn gilt:*
$(i,j) \in E \Leftrightarrow \exists \boldsymbol{x} \in T_e \forall 1 \leq k \leq N, k \notin \{i,j\} : \| \boldsymbol{c_k} - \boldsymbol{x} \| \geq max(\| \boldsymbol{c_i} - \boldsymbol{x} \|, \| \boldsymbol{c_j} - \boldsymbol{x} \|)$
Verläuft in G *eine Kante zwischen Knoten i und j, so spricht man davon, daß i und j* benachbart *sind.*

Der Algorithmus zur Bestimmung der OTPM eines Netzes bezüglich einer Trainingsmenge T ist konstruktiv. Man bestimmt für jedes $\boldsymbol{x} \in T_e$ den naheliegendsten (bmu) und zweitnaheliegendsten Knoten (zweite bmu) und verbindet diese beiden Knoten. So erhält man schrittweise die OTPM.
Die OTPM besitzt zwei wichtige Eigenschaften. Erstens hängt sie ausschließlich von der *intrinsischen Dimensionalität* von T_e ab und nicht vom Eingaberaum, das heißt, bettet man T_e in einen höherdimensionalen Raum ein, so ändert sich der Graph der OTPM nicht. Zweitens ist die OTPM skalierungs- und transformationsinvariant.

2.4.2 Intrinsische Dimensionalitätsbestimmung mit Hilfe der Hauptachsenanalyse

Die in 2.4.1 bestimmten Nachbarschaftsbeziehungen sind nun Grundlage der Bestimmung der intrinsischen Dimensionalität. Es wird davon ausgegangen, daß die Eingabevektoren nicht aus dem gesamten Eingaberaum mit der Dimension r, sondern aus einem Teil- oder Unterraum mit einer Dimension $l < r$ stammen. Man betrachte daher die Eingabevektoren $\boldsymbol{x} \in T_e$ als Ergebnis einer vektorwertigen Funktion $f : \mathbb{R}^l \to \mathbb{R}^r$,

$$\boldsymbol{x} = f(\boldsymbol{k}) + \boldsymbol{\eta}, \tag{2.8}$$

wobei $\boldsymbol{k} = [k_1, \ldots, k_l]^T$ ein k-dimensionaler Parametervektor ist und $\boldsymbol{\eta}$ das Rauschen symbolisiert, welches in den Trainingsdaten gerade unter realistischen Bedingungen auftreten kann. f kann dann in einem kleinen Bereich $\triangle \boldsymbol{k}$ um $\boldsymbol{k_0}$ mit Hilfe der Taylorentwicklung unter Vernachlässigung Terme höherer Ordnung durch eine lineare Funktion approximiert werden:

$$\triangle f = f(\boldsymbol{k_0} + \triangle \boldsymbol{k}) - f(\boldsymbol{k_0}) = \triangle \boldsymbol{k}^T \Psi(\boldsymbol{k_0}) \quad mit \quad [\Psi(\boldsymbol{k_0})]_{ij} = \frac{\partial f_i(\boldsymbol{k_0})}{\partial k_j}. \tag{2.9}$$

Man berechne nun die „Quasi-Kovarianzmatrix“ $\boldsymbol{C}$ der Vektoren $\boldsymbol{x}$ von f in der Nachbarschaft des „Quasi-Mittelwerts“ [2] $\boldsymbol{x_0} = f(\boldsymbol{k_0})$

$$\boldsymbol{C} = \boldsymbol{E}\{\triangle f \triangle f^T\} = \boldsymbol{E}\{(\boldsymbol{x} - \boldsymbol{x_0})(\boldsymbol{x} - \boldsymbol{x_0})^T\}. \tag{2.10}$$

[2] Man spricht hier von „Quasi-Kovarianz“ und „Quasi-Mittelwert“, da $\boldsymbol{x_0}$ nicht der Mittelwertvektor sondern ein fest gewählter Vektor ist. Somit ist $\boldsymbol{C}$ die Korrelationsmatrix zu $\boldsymbol{x_0}$.

Durch eine Hauptachsenanalyse läßt sich die "Quasi-Kovarianzmatrix" $\boldsymbol{C}$ in das Hauptachsensystem oder Eigensystem drehen, so daß man die p Eigenwerte μ_i und die orthonormalen Eigenvektoren $\boldsymbol{u_i}$ von $\boldsymbol{C}$ erhält mit $\boldsymbol{C}\boldsymbol{u_i} = \mu_i \boldsymbol{C} \quad i = 1, \ldots, p$ [8].
Diese Eigenvektoren bilden eine alternative Basis für die Approximation von f, also:

$$\triangle f = \triangle \boldsymbol{h}^T \Theta \quad mit \quad \Theta = [\boldsymbol{u_1}, \ldots, \boldsymbol{u_k}]. \tag{2.11}$$

Dabei koinzidieren $\triangle \boldsymbol{k}$ und Ψ mit $\triangle \boldsymbol{h}$ und Θ durch eine lineare aber unbekannte Transformation.
$\triangle f$ wird durch maximal l linear unabhängige Vektoren aufgespannt. Im Fall von unverrauschten Daten erhält man maximal l Eigenwerte. Bei verrauschten Daten können aber mehr als l Eigenwerte auftreten. Daher ist eine Selektion der Eigenwerte notwendig. Ein großer Eigenwert deutet darauf hin, daß die Varianz in Richtung des zugehörigen Eigenwerts sehr groß ist, dagegen bedeutet ein kleiner Eigenwert nur eine kleine Varianz in Richtung des Eigenvektors. Demzufolge führt man einen Schwellwert ein. Liegt ein Eigenwert unter diesem Schwellwert, geht man davon aus, daß diese Varianz durch Rauschen entstanden ist und der zugehörige Eigenvektor vernachlässigt werden kann. So ist das System in der Lage, auch verrauschte Daten zu verarbeiten. Die Wahl der Schranke muß dabei ausgewogen sein. Wählt man eine zu niedrige Schranke, ist das System sehr rauschanfällig, wählt man die Schranke zu hoch, können relevante Vektoren wegfallen, so daß ebenfalls eine falsche intrinsische Dimensionalität berechnet wird.

2.4.3 Experimente

In diesem Abschnitt soll an zwei Beispielen gezeigt werden, wie die in 2.4.1 und 2.4.2 gezeigten Zusammenhänge in einem konkreten Fall zur Anwendung kommen. Es soll also die intrinsische Dimensionalität in den Testdaten berechnet werden, indem eine Art neuronales Netz entlang der Trainingsdaten aufgebaut und Topologiebetrachtungen angestellt werden, um so auf die Dimensionalität schließen zu können. Die Experimente beziehen sich auf eine Menge von dreidimensionalen Punkten. Die Punkte beschreiben in der x-y-Ebene geometrische Figuren, die z-Koordinate soll konstant sein, ist aber durch einen gleichmäßigen Rauschanteil gestört. Das heißt, die intrinsische Dimensionalität der Punktemenge beträgt maximal zwei, bei einigen geometrischen Figuren auch nur eins. Ein neuronales Netz soll diese intrinsische Dimensionalität erkennen. Dabei wird das Netz ähnlich wie in 2.3.1 sukzessive durch Einfügen von Knoten aufgebaut. Dies ist jedoch kein neuronales Netz im eigentlichen Sinne, da das Netz keine Ausgabe erzeugt, sondern sich nur der Topologie anpassen soll. Es werden also schrittweise Neuronen entlang der Punktemenge in das neuronale Netz eingefügt und die OTPM approximiert. Anschließend kann für jeden Knoten die Kovarianzmatrix bezüglich seiner Nachbarn nach 2.10 berechnet und eine Hauptachsenanalyse durchgeführt werden, mit der die intrinsische Dimensionalität in jedem Knoten bestimmt wird. Das Vorgehen in den Experimenten wird noch einmal durch folgenden Algorithmus verdeutlicht:

Algorithmus (Topologiebestimmung)

repeat
  Bestimme Position des neueinzufügenden Knotens und füge Knoten dort ein;
  Verschiebe Knoten mit Nearest-Cluster-Algorithmus;
  Berechne OTPM;
  Bestimme Korrelationsmatrix C für jeden Knoten bezüglich seiner Nachbarn;
  Führe Hauptachsenanalyse für jeden Knoten durch und streiche kleine Eigenwerte und zugehörige Eigenvektoren nach dem Schwellwert-Prinzip;
until Abbruchkriterium erfüllt;

Im eigentlichen Sinne existiert für diesen Algorithmus kein Abbruchkriterium. Der Algorithmus soll nur deutlich machen, wie sich die Topologieberechnung mit steigender Neuronenzahl verhält.

Erstes Experiment

Beim ersten Experiment (vergl. auch [3]) besteht die Punktemenge aus 5000 zufällig gewählten Punkten, die aus Bereichen der x-y-Ebene entnommen werden. Diese Bereiche sind ein Quadrat, der Rand eines Kreises und eine Linie (siehe Abbildung 2.2). Das Quadrat hat eine Größe von 6×6 Einheiten, der Kreis einen Durchmesser von 6 Einheiten und die Linie eine Länge von 6 Einheiten. Die z-Koordinate soll konstant null sein für alle Daten, trägt aber einen Rauschanteil von maximal ± 0.5 Einheiten, also eine Varianz von 1/12.

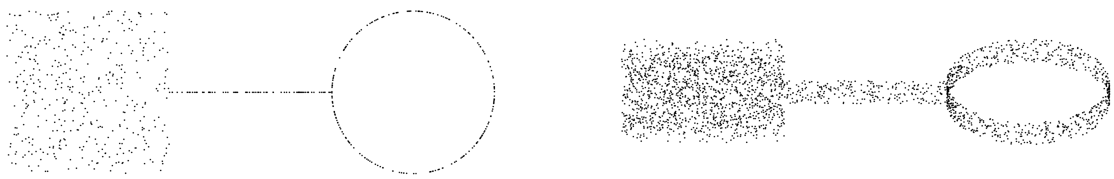

Abbildung 2.2: 2 Ansichten der Datenpunkte: Links: Ansicht auf die x-y-Ebene parallel zur z-Achse. Rechts: Ansicht auf die x-y-Ebene in einem Winkel von 50° relativ zur x-Achse.

Auf diese Daten wird nun der oben aufgeführte Algorithmus angewendet, um die intrinsische Dimensionalität in den verschiedenen Bereichen Quadrat, Kreisrand und Linie zu bestimmen. Als Schwellwert für die Eliminierung kleiner Eigenwerte und zugehöriger Eigenvektoren wurde 1/10 des größten Eigenwertes eines Neurons gewählt, der Schwellwert ist also für jeden Knoten verschieden. Im Weiteren spricht man dabei vom 10 Prozent *Niveau*. Abbildung 2.3 zeigt das neuronale Netz für 10, 20, 30, 40, 50 und 80 Knoten. Sie zeigt die eingefügten Neuronen und die zwischen den Neuronen ausgeprägten Nachbarschaftbeziehungen. Ein weißes Neuron steht dabei für eine berechnete intrinsische Dimension von drei in dem Knoten, ein graues Neuron für die Dimension zwei und ein schwarzes für die Dimension eins. Die Abbildung zeigt, wie mit zunehmender Anzahl von Neuronen die erwarteten intrinsischen Dimensionen erreicht werden, also die Dimension zwei für das Quadrat und die Dimension eins für den Kreisrand und die Linie. Es zeigt sich aber auch, daß fehlerhafte Knoten auftreten können. So befinden sich auf dem Kreisrand noch Knoten, die auf eine Dimension von zwei hindeuten (n = 10 bis 20), es sind also noch nicht genug Neuronen auf dem Rand, um die Krümmung eindimensional approximieren zu können. Bei n = 40 ermitteln alle Neuronen die richtige Dimensionalität. Mit weiter zunehmender Anzahl von Neuronen liefern immer mehr Neuronen wieder zu hohe Dimensionsangaben, genauer gesagt, einen um eine Dimension zu hohen Wert. Das liegt daran, daß das Rauschen in der dritten Komponente auch als intrinsische Dimensionalität gewertet wird (ab n = 50). Dies geschieht dann, wenn der Abstand eines Knoten zu seinen Nachbarn zu klein wird und damit auch die Varianz zwischen den Knoten, so daß nicht mehr zwischen für die intrinsische Dimensionalität wichtiger Varianz und redundantem Rauschen unterschieden werden kann.

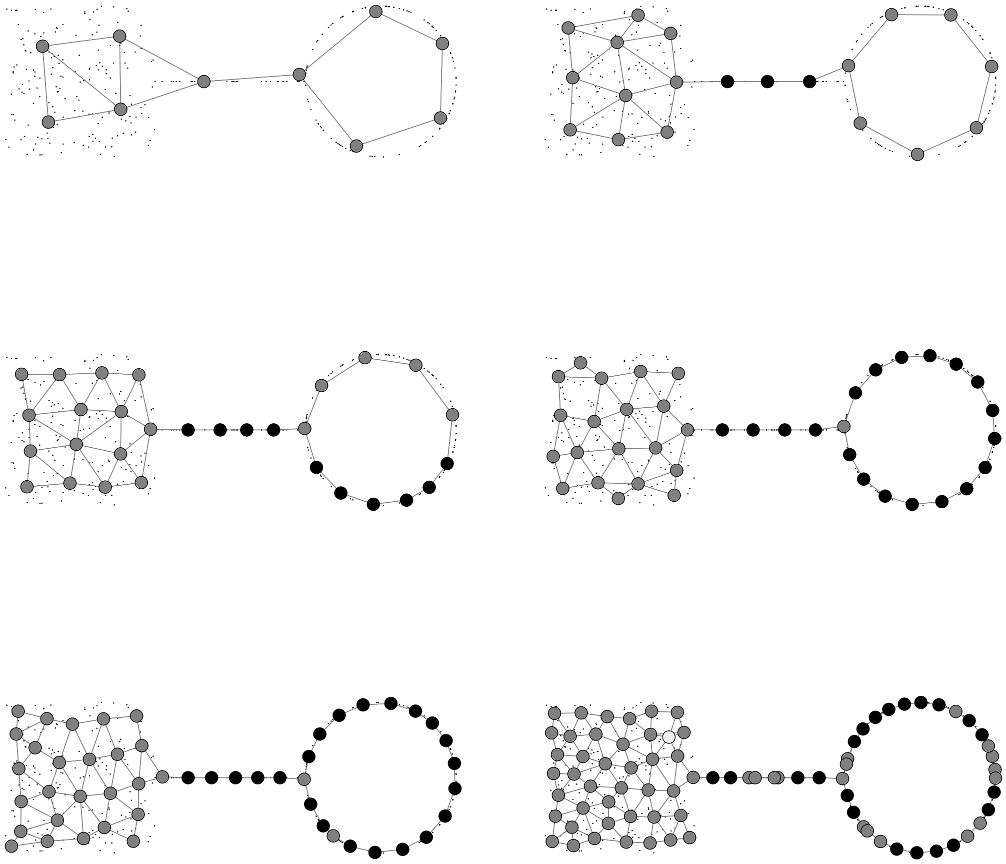

Abbildung 2.3: Die Konfiguration des neuronalen Netzes von links nach rechts, von oben nach unten bei 10, 20, 30, 40, 50, 80 Knoten. Ein weißer Knoten steht für intrinsische Dimensionalität 3, ein grauer für 2 und ein schwarzer für 1

Zweites Experiment

Im zweiten Experiment handelt es sich wieder um eine dreidimensionale Punktmenge von 1000 zufällig gewählten Punkten. Die Punkte beschreiben in der x-y-Ebene eine s-förmige Figur (siehe Abbildung 2.4). Die z-Komponente soll konstant null sein, trägt aber ein gleichmäßiges Rauschen. Die Figur besteht aus zwei Halbkreisen mit einem Radius R von 4 Einheiten, das Rauschen auf der dritten Komponente hat eine Amplitude von
$\pm$ 0.5 Einheiten, also eine Varianz von 1/12. Die Schwellwertbestimmung für die Eigenwerte geschieht auf 10 Prozent Niveau.

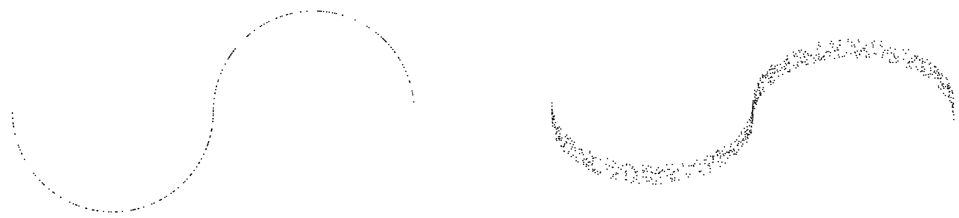

Abbildung 2.4: 2 Ansichten der Datenpunkte: Links: Ansicht auf die x-y-Ebene parallel zur z-Achse. Rechts: Ansicht auf die x-y-Ebene in einem Winkel von 50° relativ zur x-Achse.

Die Figur läßt sich durch eine Funktion $f : [0, 2\pi] \longrightarrow \mathbb{R}^2$

$$\boldsymbol{x} = \begin{cases} \begin{pmatrix} R * \cos(\phi) \\ R * \sin(\phi) \end{pmatrix} & \phi \leq \pi \\ \begin{pmatrix} 2 * R - R * \cos(\phi) \\ R * \sin(\phi) \end{pmatrix} & \phi > \pi \end{cases} \tag{2.12}$$

beschreiben, die Ausgabevektoren der Funktion liefern also die ersten beiden Komponenten der dreidimensionalen Punkte. Die intrinsische Dimensionalität der Punkte beträgt damit eins. Die intrinsische Dimensionalität wird mit Hilfe des schon im ersten Experiment verwandten Algorithmus' untersucht. Abbildung 2.5 zeigt die Konfigurationen des Netzes bei 5, 10, 15, 21, 25 und 30 Neuronen. Dabei zeigt sich wieder, wie zunächst mit zunehmender Neuronenzahl immer mehr Neuronen die richtige Dimensionalität liefern, bis bei n = 21 alle Neuronen die Dimension eins berechnen. Erhöht man die Zahl der Knoten weiter, so liefern immer mehr Neuronen eine um eins zu hohe Dimension, das heißt, derselbe Effekt wie im ersten Experiment tritt ein, das Rauschen in der dritten Komponente wird als intrinsische Dimension bewertet, da die Abstände zwischen den Neuronen zu klein werden. Die Abbildung 2.6 zeigt, wie sich die mittlere Anzahl von Eigenwerten pro Neuron bei steigender Neuronenzahl verhält.

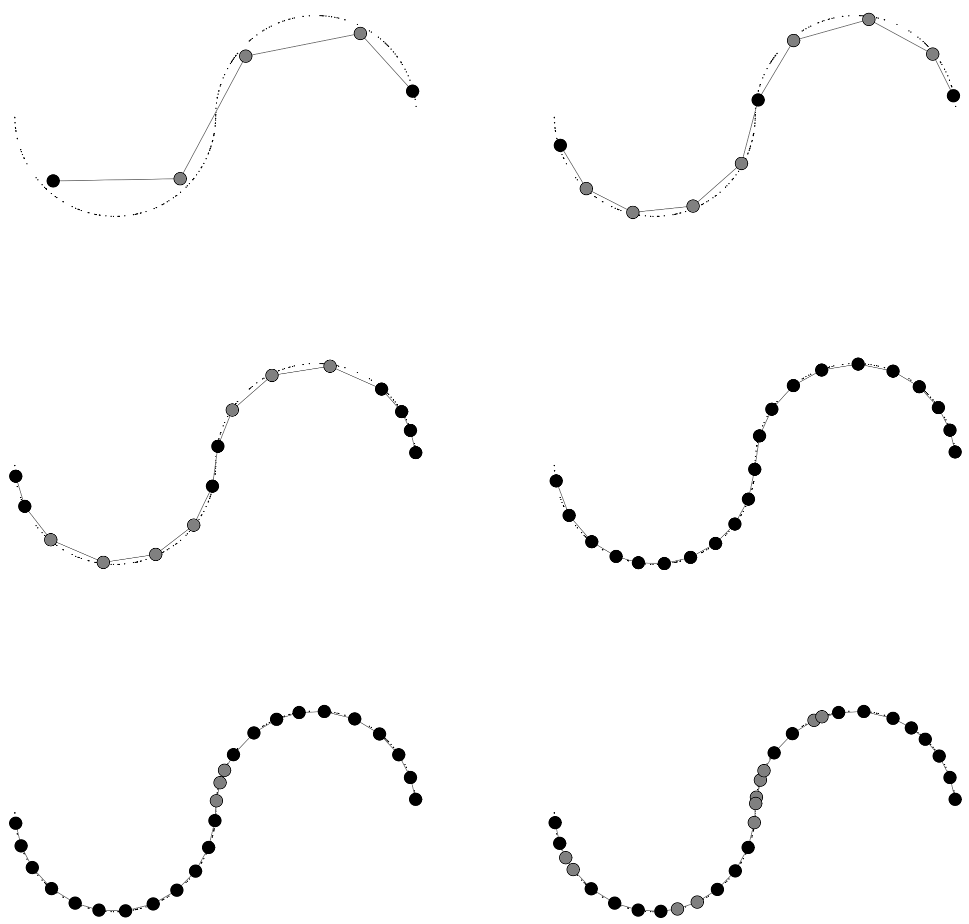

Abbildung 2.5: Die Konfiguration der neuronalen Netzes von links nach rechts von oben nach unten bei 5, 10, 15, 21, 25, 30 Knoten. Ein weißer Knoten steht für intrinsische Dimensionalität 3, ein grauer für 2 und ein schwarzer für 1.

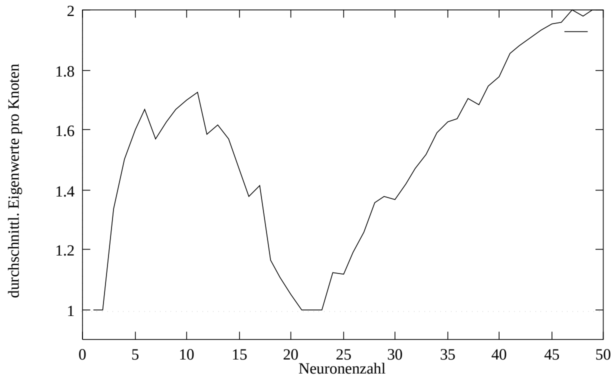

Abbildung 2.6: Kurvenverlauf für die durschnittliche Anzahl von Eigenwerten pro Knoten

2.5 Das DCS-Netz

In den folgenden Abschnitten werden nun verschiedene neuronale Netze vorgestellt. Alle diese Netze bauen auf dem Prinzip der Radialen Basisfunktions Netze (RBF) auf, nutzen jedoch die in Abschnitt 2.4 beschriebenen Topologiebetrachtungen aus, wodurch die Leistung eines neuronalen Netzes als funktionsapproximierendes Instrument verbessert werden kann. Die Abschnitte geben dabei für jeden Netztyp ein Verfahren für Aufbau und Training des Netzes an. Am Schluß jedes Abschnitts folgt dann ein Vergleich der verschiedenen Netztypen durch einige Experimente, welcher die kontinuierliche Verbesserung der Netze deutlich machen soll.
Das erste hier vorzustellende Netz ist das *DCS-Netz* von Bruske, wobei DCS für "Dynamic Cell Structures" steht. Die Theorie des DCS-Netzes hat ihr Vorbild in den *Growing Cell Structures (GCS)* von Fritzke [5]. Die Wachsenden Zell-Strukturen beschreiben eine Verbindungsstruktur zwischen den Knoten im Eingaberaum in Form von *Hypertetraedern*. Dabei ist die Anzahl von Nachbarn, die ein Knoten hat, fest vorgegeben. Bruske trennt sich von dieser Konvention und wählt die Anzahl von Verbindungen eines Knotens dynamisch, was zu der schon in 2.4.1 beschriebenen optimal die Topologie erhaltenen Karte führt. Das DCS-Netz ist dem RBF-Netz sehr ähnlich. Der Unterschied liegt darin, daß bei der Interpolation oder Approximation der Ausgabe im Gegensatz zum RBF-Netz nicht alle Neuronen betrachtet werden, sondern der zum Eingabevektor naheliegendste Knoten ermittelt wird und nur dieser Knoten und seine direkten Nachbarn zur Approximation der Ausgabe einen Beitrag leisten. Das DCS-Netz ist also ein RBF-Netz unter Verwendung einer topologischen Karte.

2.5.1 Aufbau des Netzes

Ein DCS-Netz wird durch einen Graphen $G = (V, E)$ repräsentiert. V ist die Menge der Neuronen. Jedes Neuron ist über eine Radiale Basisfunktion mit einem Zentrumsvektor

$\boldsymbol{c_i}$ und einem Normierungsparameter σ_i definiert. Desweiteren besitzt jedes Neuron einen Ausgabevektor $\boldsymbol{o_i}$. E ist eine Menge von Kanten, die Nachbarschaftsbeziehungen zwischen den Neuronen repräsentieren nach dem Prinzip der OTPM. Die Position eines neu einzufügenden Knotens kann wahlweise nach einer der beiden vorgestellten Strategien berechnet werden (siehe Algorithmus "Neuer Knoten" aus 2.3.1). Anschließend werden die Zentren der Knoten per cluster-nearest-neighbour-Algorithmus in Richtung der Trainingsdaten adaptiert.

2.5.2 Trainieren des Netzes

Trainieren der ersten Phase

Das Trainieren des Netzes geschieht auch hier getrennt in den zwei Schichten oder Phasen des Netzes. Die Bestimmung der Neuronenzentren ist schon im Aufbau des Netzes beschrieben worden. Daher soll hier gezeigt werden, wie in dieser ersten Phase auch die Adaption des Normierungsparameters σ_i jedes Neurons geschehen kann. σ_i bestimmt den Radius der radialen Basisfunktion. Der Radius wiederum gibt an, in welchem Bereich des Eingaberaums der betreffende Knoten einen relevanten Einfluß auf das Interpolationsergebnis hat. Dieser Bereich begrenzt sich durch die Positionen der direkten Nachbarn des Knotens. Daher wählt man den Normierungsparameter als die mittlere Entfernung eines Knoten zu seinen Nachbarn,

$$\sigma_i^2 = \frac{\sum_{j \in NB_i} \parallel \boldsymbol{c_i} - \boldsymbol{c_j} \parallel^2}{|NB_i|^2}. \tag{2.13}$$

$NB_i = \{j \in V \mid (i,j) \in E \vee (j,i) \in E\}$ ist dabei die Menge der Nachbarn eines Neurons.

Trainieren der zweiten Phase

In der zweiten Phase wird die Gesamtausgabe des Netzes aus der der gewichtete Summe der einzelnen Ausgabevektoren $\boldsymbol{o_i}$ der Neuronen berechnet. Es sind also diese Ausgabevektoren $\boldsymbol{o_i}$ in der zweiten Trainingsphase zu bestimmen.
Die Approximation stützt sich auf die Aktivierungen der Knoten durch die einzelnen Eingabevektoren $\boldsymbol{x_i}$ der Trainingsdaten T. Es werden, wie erwähnt, nur die Aktivierungen des zu $\boldsymbol{x_i}$ naheliegendsten Knotens (bmu) und seiner Nachbarn betrachtet, also

$$a_{i,j} = \begin{cases} \exp\left(\frac{(\boldsymbol{x_i}-\boldsymbol{c_j})^T * (\boldsymbol{x_i}-\boldsymbol{c_j})}{\sigma_j^2}\right) & j \in \{bmu \vee NB_{bmu}\} \\ 0 & sonst \end{cases} . \tag{2.14}$$

Die Summe der Aktivierungen durch einen Eingabevektor $\boldsymbol{x_i}$ ist dann

$$SumAct_i = a_{i,bmu} + \sum_{j \in NB_{bmu}} a_{i,j}. \tag{2.15}$$

Die Gesamtaktivierung des Netzes durch einen Stimulus steigt unter anderem mit der Anzahl von Nachbarn eines Neurons. Dies kann zu negativen Effekten bei der Funktionsapproximation führen, besonders bei großen Unterschieden in der Anzahl von Nachbarn der Neuronen. Es ist sinnvoll, eine konstante Gesamtaktivierung $SumAct_i = 1$ für jeden Stimulus zu erzielen. Daher werden die einzelnen Aktivierungen mit

$$a_{ij} = \frac{a_{i,j}}{SumAct_i} \quad i = 1, \dots, |T| \quad j = 1, \dots, |V| \tag{2.16}$$

normiert. Man erhält somit die Aktivierungsmatrix $\boldsymbol{A} \in \mathbb{R}^{|T|} \times \mathbb{R}^{|V|}$ und damit das aus 2.2 bekannte Approximierungsproblem zu einer Trainingsmenge $T = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \mid i = 1, \dots, t\}$:

$$\begin{pmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & & \vdots \\ a_{t1} & \cdots & a_{tk} \end{pmatrix} \begin{pmatrix} o_{11} & \cdots & o_{1m} \\ \vdots & & \vdots \\ o_{k1} & \cdots & o_{km} \end{pmatrix} = \begin{pmatrix} y_{11} & \cdots & y_{1m} \\ \vdots & & \vdots \\ y_{t1} & \cdots & y_{tm} \end{pmatrix}, \tag{2.17}$$

wobei o_{ij} bzw. y_{ij} die j-te Komponente des i-ten Ausgabevektors ist. Man beachte, daß im Gegensatz zu 2.2 viele Einträge a_{ij} der Matrix $\boldsymbol{A}$ gleich null sind.

2.5.3 Vergleich

In diesem Abschnitt soll nun untersucht werden, welche allgemeinen und speziellen Vorteile die Erweiterung eines neuronalen Netzes durch eine topologische Karte, im Speziellen durch eine OTPM erbringt. Es wird daher im Folgenden ein Vergleich mit dem klassischen RBF-Netz durchgeführt und die Unterschiede unter verschiedenen Kriterien betrachtet.

Effizienz

Effizienz bedeutet hier den Rechenaufwand und damit die Rechengeschwindigkeit des Netzes. Es ist leicht zu sehen, daß ein Netz mit topologischer Karte effizienter ist. So werden beim RBF-Netz bei der Berechnung der Ausgabe zu einem Stimulus alle N Knoten betrachtet, der Rechenaufwand hängt also von N ab. Beim DCS-Netz werden für die Suche nach dem naheliegendsten Neuron zwar zunächst auch alle Knoten betrachtet, in die Berechnung der Ausgabe des Netzes gehen jedoch nur die Ausgabevektoren von K Knoten ein (bmu und seine Nachbarn), wobei in der Regel $K << N$ ist. Dieser Vorteil macht sich auch oder gerade in der Ausgleichsrechnung zur Bestimmung der Ausgabevektoren während der Trainingsphase bemerkbar. Dieses trifft besonders bei großen Netzen zu, das heißt bei Netzen mit vielen Neuronen.

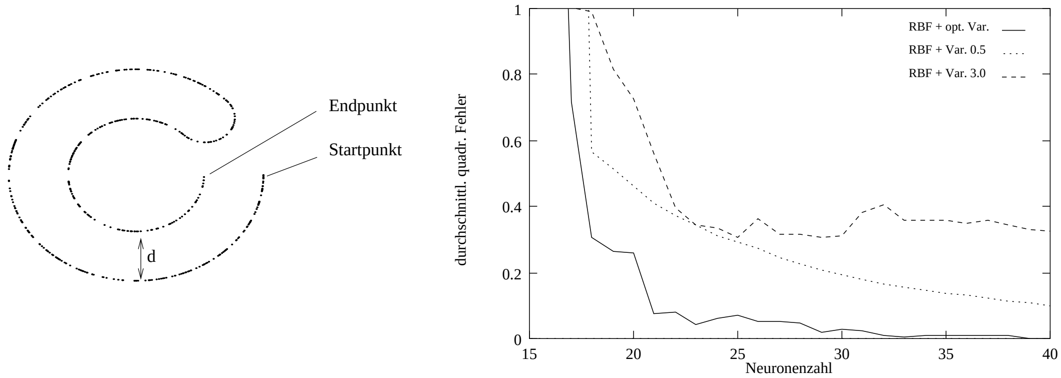

Abbildung 2.7: Links: Die Strecke, auf der sich alle Trainingspunkte befinden mit Anfangs- und Endpunkt. Rechts: Mittlerer quadratischer Fehler der Ausgaben für die drei RBF-Netze mit konstanten bzw. adaptierten Varianzen.

Adaption der Varianzen

Es wurde gezeigt, daß über die topologische Karte eine geeignete, autonome Adaption der Normierungparameter oder Varianzen der Neuronen möglich ist. Wie wichtig eine optimale Adaption für die Approximationsgüte eines RBF-Netzes ist, soll folgendes Experiment zeigen. Die Daten- oder Trainingsmenge besteht aus 1000 Daten. Die Eingabevektoren sind zweidimensionale Punkte, die auf einer Strecke mit einem Start- und einem Endpunkt verlaufen. Diese Strecke besteht aus zwei 7/8 Kreisen, die durch einen weiteren Halbkreis miteinander verbunden sind (siehe Abbildung 2.7). Der Radius des äußeren Kreises beträgt 5 Einheiten, der des inneren Kreises 3 Einheiten, so muß der Verbindungshalbkreis einen Durchmesser d von zwei Einheiten besitzen. Die Gesamtstreckenlänge der Figur beträgt damit $15\pi \sim 47$ Einheiten. Die Ausgabe eines Trainingsdatums ist die Länge der Strecke vom Startpunkt bis zum Eingabevektor des Trainingsdatums. Die Aufgabe besteht also in der Approximation einer Funktion $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$. Die Funktion ist so gewählt, daß viele Daten auftreten, bei denen die Eingabevektoren dicht beieinander liegen, deren Ausgabe jedoch sehr unterschiedlich ist, so daß die Voraussetzung, auf die sich RBF-Netze stützen, daß ähnliche Eingaben ähnliche Ausgaben erzeugen, verletzt wird. Es wird sich zeigen, daß gerade bei dieser komplizierten Struktur die optimale Adaption der Varianzen eine große Rolle bei der Approximationsgüte eines RBF-Netzes spielt. Es werden drei RBF-Netze mit unterschiedlicher Konfiguration auf dieser Datenmenge trainiert. Ein Netz adaptiert die Varianzen über die topologische Karte, während den anderen beiden die Varianzen über bestimmte Standardabweichungen a priori vorgegeben werden, also konstant sind. Die Wahl der vorzugebenen Standardabweichungen orientiert sich an der Differenz d der beiden großen Kreise. Dem einen Netz wird mit einer Standardabweichung von $0.2 << d$ Einheiten ein sehr kleiner Wert vorgegeben, während die Standardabweichung des anderen Netzes mit $3.0 > d$ Einheiten sehr groß gewählt ist. In Abbildung 2.7 (rechts) zeigt sich das Approximationsergebnis im durchschnittlichen quadratischen Approximationsfehler auf den Trainingsdaten. Wie erwartet erzielt das RBF-Netz mit adaptierten Varianzen

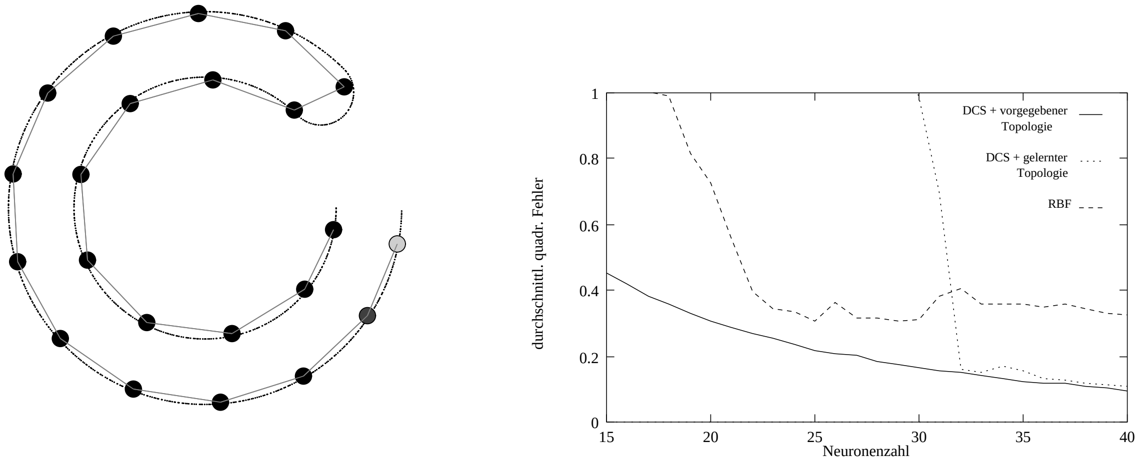

Abbildung 2.8: Links: Ein neuronales Netz mit topologischer Karte, welche die vorherrschende Topologie richtig wiedergibt. Rechts: Mittlerer, quadratischer Fehler zweier DCS-Netze mit konstanten Varianzen, einmal mit gelernter, einmal mit vorgegebener Topologie. Zum Vergleich der Fehler eines RBF-Netzes unter den selben Bedingungen.

das beste Ergebnis. Auch das Netz mit den sehr kleinen Varianzen erzielt mit steigender Neuronenzahl noch ein akzeptables Ergebnis. Das schlechteste Ergebnis liefert das Netz mit zu groß gewählten Varianzen. Die Aktivierungsbereiche der einzelnen Neuronen überlappen sich stark, so daß z.B. der Aktivierungsbereich eines Neurons auf dem inneren Kreis sehr stark in den Bereich des äußeren Kreises hineinragt. Damit erzeugt ein Stimulus, der sich auf dem äußeren Kreis befindet, eine relativ hohe Aktivierung von Neuronen auf dem inneren Kreis und vice versa, was zu schlechten Approximierungsergebnissen führt. Es zeigt sich also, daß Adaption der Varianzen über eine topologische Karte einen großen Beitrag zur Steigerung der Approximationsgüte eines RBF-Netzes leisten kann.

Es soll aber auch untersucht werden, wie sich ein DCS-Netz bei konstanten Varianzen verhält. Es werden ebenfalls zwei DCS-Netze mit konstanten Varianzen von 3.0 Einheiten trainiert. Dem einen Netz werden die richtigen Nachbarschaftsbeziehungen a priori vorgegeben, das andere soll sie autonom lernen. Die Ergebnisse der beiden Netze und als Vergleich das schon bekannte Ergebnis eines RBF-Netzes unter den selben Bedingungen sind in Abbildung 2.8 (rechts) zu sehen. Das DCS mit vorgegebener Topologie liefert ein besseres Ergebnis als das RBF-Netz. Das Problem der zu großen Aktivierungsbereiche der Neuronen wird durch die Nachbarschaftsbeziehungen gelöst. Es wird damit verhindert, daß ein Stimulus, welcher auf dem äußeren Kreis liegt, eine Aktivierung von Neuronen auf dem inneren Kreis erzeugt. Betrachtet man dagegen das Ergebnis des DCS-Netzes mit lernender Topologie, so stellt man einen sehr hohen Approximationsfehler bis zum Zeitpunkt $n = 30$ fest. Zu diesem Zeitpunkt hat das Netz die richtigen Nachbarschaftsbeziehungen gelernt. In den Stadien vorher bilden sich noch fehlerhafte Verbindungen zwischen den Neuronen. Gerade falsche Verbindungen zwischen Neuronen auf dem inneren und äußeren Kreis führen dabei zu großen Defekten bei der Approximation.

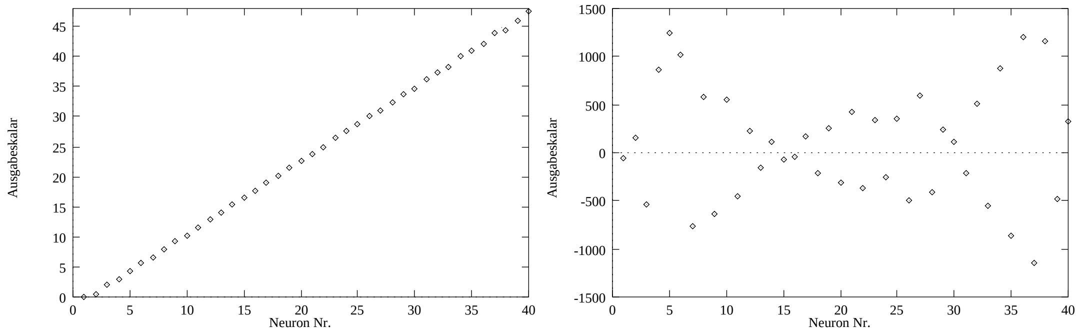

Abbildung 2.9: Ausgabeskalare der Neuronen zweier trainierter Netze mit jeweils 40 Neuronen, wobei die Neuronen nach der Reihenfolge ihres Auftretens auf der Figur geordnet sind. Links: Graph für ein DCS-Netz. Rechts: Graph für ein RBF-Netz.

Es hat sich also gezeigt, daß die Verwendung einer topologischen Karte die Approximationsgüte eines neuronales Netzes verbessern kann. Die Topologie liefert eine Berechnungsgrundlage für die Adaption der Varianzen. Geht man jedoch von konstanten Varianzen aus, so können Probleme, die daraus entstehen, wie zum Beispiel Überlappungen der Aktivierungsbereiche, ebenfalls über die topologische Karte gelöst werden.

Regularisierung

Ein weiterer Vorteil der Verwendung einer topologischen Karte soll jedoch nicht außer acht gelassen werden, es ist ihre *regularisierende Wirkung* auf das Netz. Das allgemeine RBF-Netz arbeitet nach dem "black-box"-Prinzip. Die Position der Neuronen und die Radien der Basisfunktionen ergeben sich aus der vorliegenden Topologie, der Ausgabevektor eines Neurons läßt im Allgemeinen aber keine Interpretation zu. Bei dem DCS-Netz ist dies anders. Betrachtet man im vorangegangenen Beispiel die Struktur des Netzes, so zeigt sich, daß Neuronen, die im Streckenverlauf dichter am Startpunkt liegen, einen kleineren Ausgabewert haben als Neuronen, die im Streckenverlauf weiter entfernt liegen. Das bedeutet, daß sich die Topologiebetrachtung auch in der Ausgabe jedes einzelnen Neurons widerspiegelt. Abbildung 2.9 macht diesen Effekt sehr deutlich. Die Werte der Ausgabevektoren (in diesem Fall Skalare) der einzelnen Neuronen sind in der Reihenfolge, wie sie vom Startpunkt an auf der Figur auftreten, in einen Graphen eingetragen. Beim DCS-Netz liegen sie wie erwartet auf einer Geraden. Im Fall des RBF-Netzes dagegen lassen die Daten keinerlei Struktur erkennen, auch der Wertebereich ist völlig verschieden von dem der eigentlich zu berechnenden Ausgabe.

Basis für erweiterte Lernregeln

Die Lernverfahren von neuronalen Netzen basieren auf Lernregeln. Es wurde hier das parallele Lernverfahren über eine Ausgleichsrechnung und ein schrittweises Lernverfahren mittels des *least-mean-square*-Algorithmus' vorgestellt. Es gibt jedoch auch Verfahren, die eine topologische Karte benötigen. So bauen z.B. die *Kohonen Regeln* von T. Kohonen auf Nachbarschaftsbeziehungen zwischen den Knoten auf [10]. DCS-Netze können diese Lernverfahren in abgewandelter Form nutzen (siehe dazu z.B. [2]).

2.6 Das SDCS-Netz

In diesem Abschnitt soll nun eine erste Erweiterung des DCS-Netzes vorgestellt werden, es nennt sich *SDCS-Netz*. SDCS steht für "Subspace Dynamic Cell Structures". Beim DCS-Netz wurden Nachbarschaftsbeziehungen zwischen den Neuronen betrachtet, die sich aus der Mannigfaltigkeit der Daten herleiten ließen, die Approximation geschah aber im gesamten Eingaberaum.
Das SDCS-Netz dagegen nutzt die Mannigfaltigkeitsberechnungen von 2.4.2 aus und bewegt sich bei der Approximation nur auf dem Unterraum. Es wird sich zeigen, daß ein SDCS-Netz Vorteile in der Approximationsgüte gegenüber den vorangegangenen Netztypen hat. Sie liegen darin, daß ein SDCS-Netz durch Projektion auf die Eingabemannigfaltigkeit relativ robust ist gegenüber verrauschten Daten.

2.6.1 Aufbau des Netzes

Das SDCS-Netz wird, genau wie das DCS-Netz, durch einen Graphen $G = (V, E)$ repräsentiert mit einer Menge V von Neuronen und einer Kantenmenge E, die die Nachbarschaftsbeziehungen angibt. Jedes Neuron hat einen Zentrumsvektor $\boldsymbol{c_i}$, einen Normierungsparameter σ_i und einen Ausgabevektor $\boldsymbol{o_i}$. Die Erweiterung zum DCS-Netz liegt in der Matrix $\boldsymbol{E}_i$ der normierten *Eigenvektoren* und dem Vektor $\boldsymbol{e}_i$ der zugehörigen *Eigenwerte*, die den Verlauf der Unterraums, in dem sich die Eingabedaten befinden, und dessen lokale intrinsische Dimensionalität im lokalen Bereich des Neurons angeben. Man erhält die Eigenvektoren und Eigenwerte über eine Hauptkomponentenanalyse der Korrelationmatrix des jeweiligen Neurons. Die Eigenvektoren spannen im lokalen Bereich des Neurons den Raum der gesuchten Mannigfaltigkeit auf und ermöglichen damit die Approximation in dem Unterraum. Die Idee dabei ist, daß, sobald man die Mannigfaltigkeit bestimmt hat, sich alle unverrauschten Eingabevektoren innerhalb der Mannigfaltigkeit befinden werden. Befindet sich ein Eingabevektor jedoch außerhalb der Mannigfaltigkeit, so geht man davon aus, daß es sich um ein verrauschtes Datum handelt und der Rauschanteil durch eine Rückprojektion in die Mannigfaltigkeit herausgefiltert werden kann.

2.6.2 Trainieren des Netzes

Hauptkomponentenanalyse

Bei der Erläuterung dieser Trainingsphase wird davon ausgegangen, daß das Netz aus einer Menge von Knoten besteht, deren Zentren und Verbindungen in Bezug auf eine Trainigsmenge bestimmt worden sind. Die dazu nötigen Verfahren sind ausreichend erläutert worden. Dieser Abschnitt konzentriert sich auf die Berechnung der neuen Komponenten, also die Bestimmung der Eigenwerte und Eigenvektoren.
Die Matrix $\boldsymbol{E_i}$ der Eigenvektoren eines Neurons berechnet sich über die Korrelationsmatrix $\boldsymbol{C_i}$ zu seinen Nachbarn. Das heißt, ist $NB_i = \{j \in V \mid (i,j) \in E \vee (j,i) \in E\}$ die Menge der Nachbarn von Neuron i, dann ist

$$\boldsymbol{C}_i = \frac{1}{|NB_i|}\boldsymbol{A^T A} \quad mit \quad \boldsymbol{A^T} = [\boldsymbol{c_{1_i}} - \boldsymbol{c_i}, \ldots, \boldsymbol{c_{|NB_i|_i}} - \boldsymbol{c_i}] \quad \boldsymbol{c_{j_i}} \in NB_i, \tag{2.18}$$

wobei $(\boldsymbol{c_{j_i}} - \boldsymbol{c_i})$ die Differenzvektoren zwischen dem Zentrum des betrachteten Neurons und den Zentren seiner Nachbarn sind. Ist n die Dimension der Eingabevektoren, dann ist $\boldsymbol{C_i}$ also eine $n \times n$ Matrix. Die Eigenwerte und zugehörigen Eigenvektoren erhält man durch eine Hauptachsenanalyse der Matrix $\boldsymbol{C_i}$. Ein Verfahren dafür ist die Singulärwertzerlegung, die die Eigenwerte und die orthogonalen Eigenvektoren liefert [15]. Der Aufwand beträgt $O(n^3)$, das Verfahren ist also für hochdimensionale Eingaben sehr rechenaufwendig. Der folgende Satz [14] zeigt eine Umformung des Problems in ein anderes, welches in einem Aufwand von $O(|NB_i|^3 + |NB_i|^2 n)$ berechnet werden kann. Da die Dimension des Eingaberaums in der Regel erheblich größer ist als die Zahl der Nachbarn eines Knotens, erbringt diese Umformung einen erheblichen Effizienzgewinn.

Satz 1 *Sei $S = \{\boldsymbol{c_i} \in I\!R \mid i = 1, \ldots, N\}$ eine Menge von Punkten und $\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{A}^T\boldsymbol{A}$ die Kovarianzmatrix zu S mit $\boldsymbol{A}^T = [\boldsymbol{c_1} - \boldsymbol{c_{mid}}, \ldots, \boldsymbol{c_N} - \boldsymbol{c_{mid}}]$, wobei $\boldsymbol{c_{mid}}$ der Mittelwert von S ist. Sei $\tilde{\boldsymbol{\Sigma}} = \frac{1}{N}\boldsymbol{A}\boldsymbol{A}^T$.*
Dann haben $\boldsymbol{\Sigma}$ und $\tilde{\boldsymbol{\Sigma}}$ dieselben Eigenwerte $\mu_1, \ldots, \mu_N$, und die orthonormalen Eigenvektoren $\boldsymbol{v_1}, \ldots, \boldsymbol{v_N}$ von $\boldsymbol{\Sigma}$ berechnen sich aus den orthonormalen Eigenvektoren $\tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_N$ von $\tilde{\boldsymbol{\Sigma}}$ durch $\boldsymbol{v_i} = \frac{A^T\tilde{\boldsymbol{v}}_i}{\sqrt{\mu_i N}}$.

Beweis: Siehe Anhang A.

Die Matrix $\tilde{\boldsymbol{\Sigma}} = \frac{1}{N}\boldsymbol{A}\boldsymbol{A}^T$ hat die Dimension $|NB_i| \times |NB_i|$. Eine Singulärwertzerlegung von $\boldsymbol{AA^T}$ hat den Aufwand $O(|NB_i|^3)$, die Umformung geschieht in Zeit $O(|NB_i|^2 n)$, der Gesamtaufwand der Hauptachsenanalyse liegt damit bei $O(|NB_i|^3 + |NB_i|^2 n)$.
Somit erhält man für jedes Neuron i die Matrix der Eigenvektoren

$$\boldsymbol{E_i} = (\boldsymbol{v_1}, \ldots, \boldsymbol{v_{|NB_i|}})^T \tag{2.19}$$

und den Vektor der Eigenwerte

$$\boldsymbol{e_i} = [\mu_1, \ldots, \mu_{|NB_i|}]^T. \tag{2.20}$$

Wie schon in 2.4.2 erläutert, soll weiterhin eine Auswahl der Eigenwerte und damit der Eigenvektoren geschehen. Sehr kleine Eigenwerte bedeuten geringe Varianzen in Richtung des zugehörigen Eigenvektors, die durch Rechenungenauigkeiten oder Rauschen in den Daten verursacht worden sein können und damit eliminiert werden können. Als Schwellwert s_i zur Eliminierung wählt man wieder einen Bruchteil des größten Eigenwertes, also

$$s_i = \epsilon * \max\{\mu_j \mid j \in NB_i\}, \tag{2.21}$$

wobei ϵ das übliche Prozent Niveau angibt. Für den Vektor $\boldsymbol{e_i}$ der Eigenwerte und die Matrix $\boldsymbol{E_i}$ ergibt sich dann

$$\boldsymbol{e_i} = [\mu_1, \ldots, \mu_{l_i}]^T \quad mit \quad \mu_j \geq s_i, \quad l_i \leq |NB_i| \tag{2.22}$$

und

$$\boldsymbol{E_i} = (\boldsymbol{v_1}, \ldots, \boldsymbol{v_{l_i}})^T. \tag{2.23}$$

Dabei ist die j-te Zeile der Matrix $\boldsymbol{E_i}$ der normierte Eigenvektor zum j-ten Eigenwert, der in der j-ten Komponente des Vektors $\boldsymbol{e_i}$ steht. Die intrinsische Dimension im lokalen Bereich eines Neurons wird also durch die Dimension des Vektors $\boldsymbol{e_i}$ und der lokale Unterraum über die Eigenvektoren in der Matrix $\boldsymbol{E_i}$ definiert.

Normierungsparameter

Der Normierungsparameter σ_i eines Neurons berechnet sich aus der gemittelten Summe der Abstände zu seinen Nachbarknoten. Dabei werden die Abstände in dem durch die Matrix $\boldsymbol{E_i}$ beschriebenen Unterraum betrachtet, Abstandskomponenten, die orthogonal zum Unterraum sind, sollen eliminiert werden, um auch hier gegen Rauscheinflüsse robust zu sein. Man erhält:

$$\sigma_i^2 = \sum_{\tilde{j} \in NB_i} \frac{[\boldsymbol{E}_i(\boldsymbol{c}_{\tilde{j}} - \boldsymbol{c}_i)]^T \boldsymbol{E}_i(\boldsymbol{c}_{\tilde{j}} - \boldsymbol{c}_i)}{|NB_i|^2}. \tag{2.24}$$

Aktivierungen

Der Eingabevektor $\boldsymbol{x_i}$ eines Trainingsdatums aktiviert wie im Fall des DCS-Netzes den naheliegendsten Knoten bmu und seine Nachbarn. Der Abstand zwischen $\boldsymbol{x_i}$ und den Zentren der Knoten bestimmt die Stärke der Aktivierung, wobei der Abstand wiederum nur innerhalb des Unterraums gemessen wird. Man erhält dadurch eine neue Berechnung der Aktivierung durch einen Eingabevektor $\boldsymbol{x}_i$:

$$a_{i,j} = \begin{cases} \exp\left(\frac{[\boldsymbol{E}_i(\boldsymbol{x}_i - \boldsymbol{c}_j)]^T \boldsymbol{E}_i(\boldsymbol{x}_i - \boldsymbol{c}_j)}{\sigma_j^2}\right) & j \in \{bmu \vee NB_{bmu}\} \\ 0 & sonst \end{cases} . \tag{2.25}$$

Die Aktivierungen $a_{i,j}$ werden gemäß (2.15) und (2.16) auf eine Gesamtaktivierung von 1 normiert.

Bestimmung der Ausgangsvektoren

Die Bestimmung der Ausgangsvektoren führt auf das schon bekannte Ausgleichsproblem 2.17, welches nach dem bekannten Verfahren gelöst wird. Damit ist das Trainieren des Netzes abgeschlossen.

2.6.3 Vergleich

Auch hier soll wieder ein Vergleich mit einem RBF-Netz angestellt werden, um Verbesserungen durch diese neue Variante zu verdeutlichen. Der Vergleich soll veranschaulichen, wie das neue Verfahren sich bezüglich der Approximationsgüte verhält. Der Effizienzgewinn ist dabei schon angesprochen worden. Die Berechnungen werden nicht mehr im gesamten Eingaberaum durchgeführt, sondern im niederdimensionalen Unterraum. Die eigentliche Intention für die Erweiterung des Netzes durch die Hauptkomponentenanalyse war es aber, einen Netztyp zu erhalten, der relativ unanfällig gegen Rauschen ist. Die folgenden Versuche zeigen, daß das SDCS-Netz dieses bewerkstelligen kann.

Versuche

Als Versuchsbeispiel dient hier wieder die zweidimensionale, s-förmige Figur aus Abbildung 2.4. Dabei wird die Figur in einen beliebig dimensionalen Raum eingebettet, um zu zeigen, daß ein SDCS-Netz relativ konstante Ergebnisse bei verschieden dimensionalen Eingaberäumen liefert, damit also nahezu unabhängig von der Dimension des Eingaberaums ist. Die Eingabevektoren der Trainingsmenge berechnen sich in Anlehnung an Gleichung 2.12 durch eine Vektorfunktion $f_e : ([0, 2\pi], \mathbb{R}) \longrightarrow R^2$

$$\boldsymbol{f}(\phi, \eta) = \begin{cases} \begin{pmatrix} (R + \eta_0) * \cos(\phi) \\ (R + \eta_0) * \sin(\phi) \end{pmatrix} & \phi \leq \pi \\ \begin{pmatrix} 2 * R - (R + \eta_0) * \cos(\phi) \\ (R + \eta_0) * \sin(\phi) \end{pmatrix} & \phi > \pi \end{cases} \tag{2.26}$$

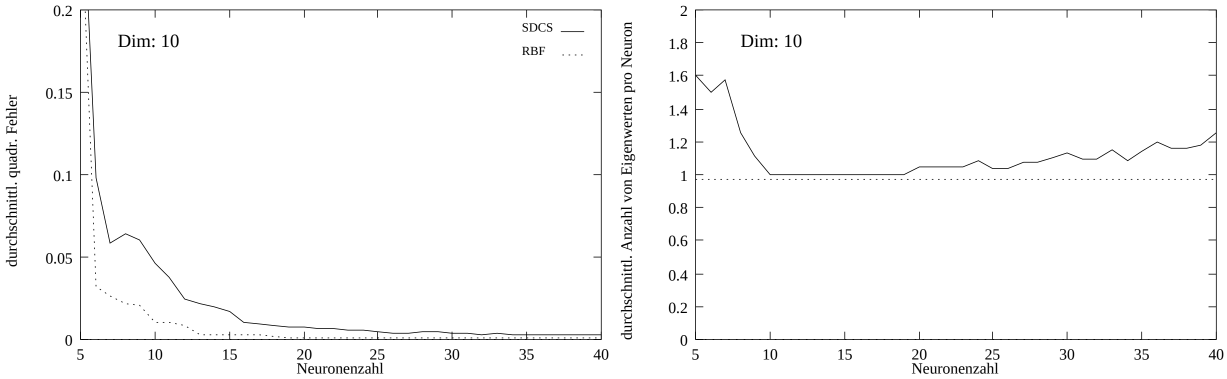

Abbildung 2.10: Links: Der mittlere Fehler eines RBF- und eines SDCS-Netzes auf den verrauschten Trainingsdaten. Rechts: Mittlerer Anzahl von Eigenwerten pro Neuron im SDCS-Netz

Dabei soll die intrinsische Dimension der Eingabevektoren eins sein und η ein gleichmäßiges Verrauschen der Daten repräsentieren, orthogonal zur Tangente in dem unverrauschten Punkt $f_e(\phi, 0)$. Der Eingabevektor wird um d Komponenten erweitert, die jeweils einen konstanten Wert (der Einfachheit halber den Wert 0) haben, jedoch durch ein gleichmäßiges Rauschen $\eta_l, l = 1, \ldots, d$ gestört sind. Die Eingabevektoren x_i erhalten also eine Dimension von $d + 2$. Die zu lernende Funktion $f_a : \mathbb{R}^{d+2} \longrightarrow \mathbb{R}^2$ ist dann wie folgt definiert:

$$f_a(f_e(\phi, \eta_0), \eta_1, \ldots, \eta_d) = f_e(\phi, 0). \tag{2.27}$$

Das heißt also, die Funktion f_a bildet einen Vektor auf seine ersten beiden Komponenten ab, wobei das Rauschen abgeschnitten wird. Diese Funktion eignet sich sehr gut, um zu zeigen, wie vorteilhaft sich ein SDCS-Netz bei verrauschten Daten verhält.
Der erste Versuch testet ein SDCS-Netz und ein RBF-Netz mit optimal angepaßten Varianzen auf einer Trainingsmenge mit 5000 Punkten mit 10-dimensionalen Eingabevektoren. Der Radius der beiden Halbachsen der Figur beträgt 4 Einheiten und die Amplitude des Rauschens auf den einzelnen Komponenten $\pm$ 0.4 Einheiten. Es wird zunächst der mittlere quadratische Fehler zwischen Ausgabe des Trainingsdatums und berechneter Ausgabe des jeweiligen Netzes betrachtet. Es zeigt sich (siehe Abb. 2.10 links), daß beide Verfahren ein ähnliches Ergebnis liefern, wobei sich das RBF-Netzes als etwas besser erweist. Das RBF-Netz lernt das Rauschen mit, während das SDCS-Netz versucht, die richtige Struktur der Eingabedaten zu bestimmen, um Rauschanteile herauszufiltern. Grafik 2.10 rechts gibt die durchschnittliche Anzahl von Eigenwerten pro Neuron im SDCS-Netz an. Bei 10 Knoten im Netz erreicht die Topologiebestimmung den erwarteten Wert von einem Eigenwert pro Neuron. Mit zunehmender Zahl von Neuronen steigt dieser Wert, da dann das Rauschen in den Komponenten in die Dimensionalitätsbestimmung miteingeht.
Das RBF-Netz liefert zwar auf den Trainingsdaten eine besseres Ergebnis, aber der Effekt des Mitlernens des Rauschens wirkt sich negativ aus, wenn das RBF-Netz die Approximation auf einer Datenmenge durchführen soll, die sich von der Trainingsmenge unterscheidet. Das RBF-Netz wird hier größere Fehler in der Approximation erzeugen, während das SDCS-Netz aufgrund seiner Fähigkeit zur Filterung des Rauschens ähnliche Ergebnisse wie

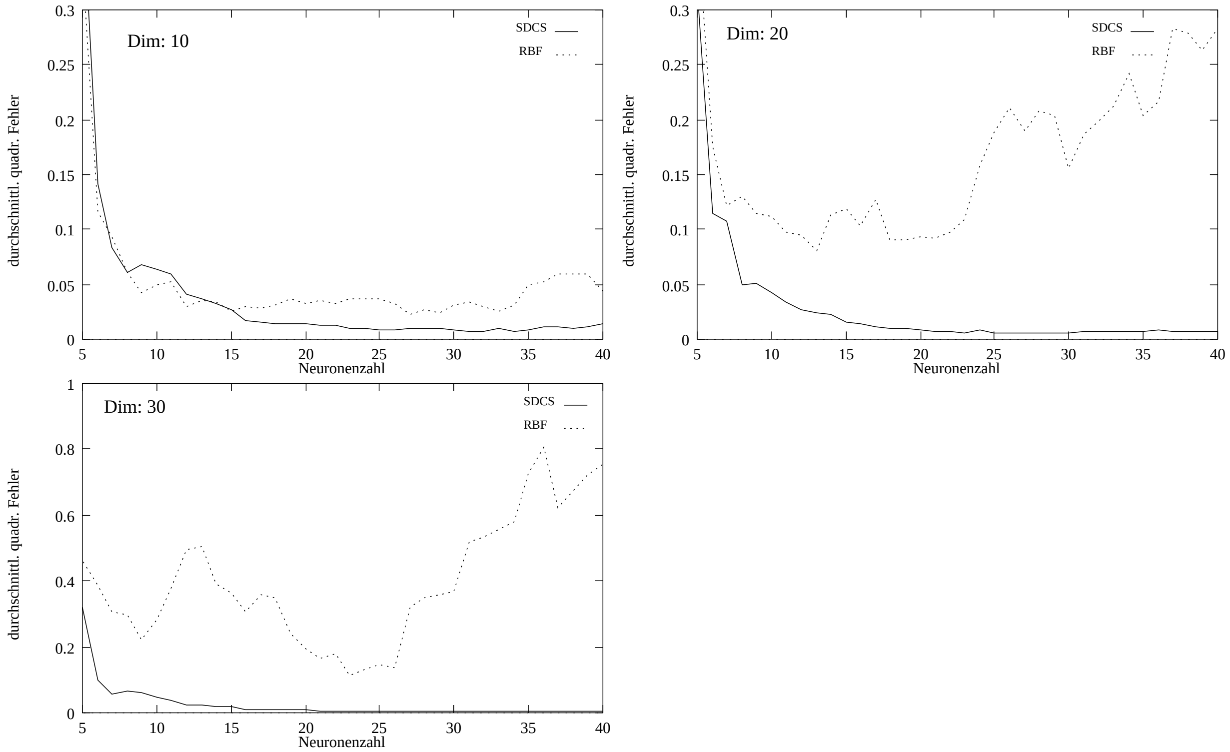

Abbildung 2.11: Von oben nach unten, von links nach rechts: Der mittlere Fehler eines RBF- und eines SDCS-Netzes auf den Testdaten in einem 10-, 20- und 30-dimensionalen Eingaberaum.

auf der Trainingsmenge liefern wird. Der folgende Versuch verdeutlicht dies.
In einem zweiten Versuch werden zwei verschiedene Datenmengen verwendet. Die Abtastpunkte der zu approximierenden Funktion sollen nur durch ein gemäßigtes Rauschen gestört sein. Diese Daten bilden die übliche Trainingsmenge T_{train}. Beide Netze werden mit dieser Trainingsmenge trainiert. In einer zweiten Datenmenge befinden sich stärker verrauschte Daten. Es soll mit diesen Daten die Toleranz der Netze gegenüber stärker verrauschten Daten getestet werden. Sie bilden also die Testmenge T_{test}. Man beachte, daß die Elemente der Testmenge keinerlei Einfluß auf die Konfiguration der Netze haben. Die Trainingsmengen bestehen aus je 5000 Datenpunkten, wobei die Eingabevektoren in Räume unterschiedlicher Dimension eingebettet sind, nämlich der Dimensionen 10, 20 und 30. Die Rauschamplitude beträgt $\pm$ 0.4 Einheiten. Es handelt sich also im Fall von 10-dimensionalen Eingabevektoren um dieselbe Trainingsmenge wie im ersten Versuch. In den Testmengen befinden sich jeweils 5000 stärker verrauschte Daten mit einer Rauschamplitude von $\pm$ 1.0 Einheiten.

In den Ergebnisdiagrammen von Abb. 2.11 wird der mittlere quadratische Fehler zwischen der Ausgabe des jeweiligen Netzes und dem Ausgabevektor der Testdaten betrachtet. Der Fehler, den das RBF-Netz erzeugt, ist deutlich größer und nimmt mit steigender Dimension der Eingabevektoren zu, während die Fehlerkurve des SDCS-Netzes auch in höherdimensionalen Eingaberäumen nahezu denselben Verlauf hat. Dieser Versuch unterstreicht, daß ein SDCS-Netz bei der Approximation einer Funktion nahezu unabhängig von der Dimension des Eingaberaums ist, da es sich in dem niederdimensionalen Unterraum bewegt und so auch gegen verstärktes Rauschen robust bleibt.

2.7 SDCS-Netz mit linearer Erweiterung

In diesem Abschnitt soll nun eine letzte Variante eines neuronalen Netzes entwickelt werden. Es handelt sich um eine Erweiterung des SDCS-Netzes. Beim SDCS-Netz wurde gezeigt, wie es das Problem von verrauschten Daten löst, indem es die Daten in den gefundenen Unterraum zurückprojiziert. Die Informationen über den Unterraum können aber noch weiter ausgenutzt werden, indem man versucht, sich bei einem Approximationsvorgang entlang des detektierten Unterraums von einem Neuron zu einem Trainings- oder Testdatum zu bewegen. Man geht dabei davon aus, daß das Netz so dicht mit Neuronen besetzt ist, daß die Funktion zwischen einem Trainigsdatum und dem naheliegendsten Neuron linear approximiert werden kann. Dabei soll die Strategie des "winner takes all" verfolgt werden, das heißt, es wird nur noch der naheliegendste Knoten aktiviert und nicht mehr auch seine Nachbarn.

2.7.1 Aufbau des Netzes

Es handelt es sich bei diesem Netz um eine Erweiterung des SDCS-Netzes durch eine lineare Approximation. Grundlage ist also ein SDCS-Netz mit all seinen in 2.6.1 beschriebenen Komponenten. Die Erweiterung besteht in der Matrix $\boldsymbol{J_i}$, der sogenannten *Jacobi-Matrix*. Sie gibt die partiellen Ableitungen der Teilfunktion f_i von Knoten i an, die im lokalen Bereich als lineare Funktion betrachtet wird. Damit ist $\boldsymbol{J_i}$ eine $m \times n$ Matrix, wobei n die Dimension der Eingabevektoren und m die der Ausgabevektoren ist.

Approximationsfunktion

Nach den vorausgegangenen Verfahren berechnet sich die Ausgabe $\boldsymbol{y}$ eines der hier vorgestellten neuronalen Netze auf einen Eingabevektor $\boldsymbol{x}$ mit

$$\boldsymbol{y} = \sum_{i \in V} a_i \boldsymbol{o_i}, \tag{2.28}$$

wobei $a_j, j = 1, \ldots, |V|$ die normierten Aktivierungen der Neuronen durch den Eingabevektor sind. Ritter, Martinetz und Schulten jedoch approximieren die zu bestimmende Funktion durch lokale lineare Funktionen in den Voronoizellen der Neuronen [16]. Damit erhält man anstatt 2.28

$$\boldsymbol{y} = \sum_{i \in V} a_i(\boldsymbol{o_i} + \boldsymbol{J_i}(\boldsymbol{x} - \boldsymbol{c_i})), \tag{2.29}$$

wobei J_i die oben beschriebene Jacob-Matrix ist. Da nach der "winner takes all"-Strategie nur die Aktivierung des naheliegendsten Neurons bmu betrachtet werden soll, erhält man durch Normierung der Aktivierung der bmu auf eins die vereinfachte Gleichung

$$\boldsymbol{y} = \boldsymbol{o_{bmu}} + \boldsymbol{J_{bmu}}(\boldsymbol{x} - \boldsymbol{c_{bmu}}). \tag{2.30}$$

Der Differenzvektor $\boldsymbol{x} - \boldsymbol{c_{bmu}}$ beschreibt die Translation zwischen Eingabevektor und Zentrum des naheliegendsten Knotens. Da die Berechnung jedoch innerhalb des Unterraumes geschehen soll, um Rauschen zu unterdrücken, wird der Distanzvektor wieder mit der Matrix der Eigenvektoren $\boldsymbol{E_{bmu}}$ multipliziert, und man erhält:

$$\boldsymbol{y} = \boldsymbol{o_{bmu}} + \tilde{\boldsymbol{J}}_{\boldsymbol{bmu}}[\boldsymbol{E_{bmu}}(\boldsymbol{x} - \boldsymbol{c_{bmu}})]. \tag{2.31}$$

Man beachte, daß die Matrix $\tilde{\boldsymbol{J}}_{\boldsymbol{bmu}}$ eine geringere Spaltenzahl hat, nämlich die Anzahl der Eigenwerte des betreffenden Neurons. Es werden also nur die partiellen Ableitungen innerhalb der Unterraums betrachtet. Im weiteren Verlauf ist mit $\boldsymbol{J_i}$ die auf den Unterraum projizierte Jacobi-Matrix $\tilde{\boldsymbol{J}}_{\boldsymbol{i}}$ zu verstehen.

2.7.2 Trainieren des Netzes

Das Trainieren der ersten Phase, also die Bestimmung der Neuronenzentren, Normierungsparameter und der Matrix der Eigenvektoren bzw. des Vektors der Eigenwerte geschieht analog zum Verfahren eines SDCS-Netzes ohne lineare Erweiterung. In der zweiten Phase gilt es dann, für jedes Neuron i die Jacobi-Matrix $\boldsymbol{J_i}$ und den Ausgabevektor $\boldsymbol{o_i}$ zu bestimmen. Grundlage für die Bestimmung ist die Trainingsmenge T, so daß für jedes Trainingsdatum $(\boldsymbol{x^{(t)}}, \boldsymbol{y^{(t)}}) \in T$ die Gleichung 2.31 erfüllt wird. Man erhält somit $|T|$ Gleichungen

$$\begin{pmatrix} y_1^{(t)} \\ \vdots \\ y_m^{(t)} \end{pmatrix} = \begin{pmatrix} o_1^{(bmu)} \\ \vdots \\ o_m^{(bmu)} \end{pmatrix} + \begin{pmatrix} j_{1,1}^{(bmu)} & \cdots & j_{1,l_i}^{(bmu)} \\ \vdots & & \vdots \\ j_{m,1}^{(bmu)} & \cdots & j_{m,l_i}^{(bmu)} \end{pmatrix} \begin{pmatrix} \Delta_{bmu} x_1^{(t)} \\ \vdots \\ \Delta_{bmu} x_{li}^{(t)} \end{pmatrix}. \tag{2.32}$$

Es sind dabei $[\Delta_i x_1^{(t)}, \ldots, \Delta_i x_{l_i}^{(t)}]^T = \boldsymbol{E_i}(\boldsymbol{x^{(t)}} - \boldsymbol{c_i})$ der Distanzvektor von Neuron i zum Trainingsdatum t im Unterraum, $j_{kl}^{(i)}$ die Matrixeinträge von $\boldsymbol{J_i}$, und $o_k^{(i)}$ die Komponenten des Ausgabevektors $\boldsymbol{o_i}$. Faßt man alle diese Gleichungen zusammen und strukturiert sie zu neuen Gleichungssystemen komponentenweise nach den Komponenten der Ausgabe, so erhält man damit für jedes Neuron i insgesamt m neue Gleichungssysteme

$$\begin{pmatrix} y_k^{(1)} \\ \vdots \\ y_k^{(|T|)} \end{pmatrix} = \begin{pmatrix} ([\Delta_i x_1^{(1)}, \ldots, \Delta_i x_{l_i}^{(1)}], 1) \\ \vdots \\ ([\Delta_i x_1^{(|T|)}, \ldots, \Delta_i x_{l_i}^{(|T|)}], 1) \end{pmatrix} \begin{pmatrix} j_{k,1}^{(i)} \\ \vdots \\ j_{k,l_i}^{(i)} \\ o_k^{(i)} \end{pmatrix} \tag{2.33}$$

bezüglich jeder Komponente k der Ausgabe. Faßt man die Gleichungen aller Knoten für eine Komponente k zusammen, erhält man

$$\begin{pmatrix} y_k^{(1)} \\ \vdots \\ y_k^{(|T|)} \end{pmatrix} = \begin{pmatrix} a_{1,1}([\boldsymbol{\Delta_1 x^{(1)}}, 1) & \dots & a_{1,|V|}([\boldsymbol{\Delta_{|V|} x^{(1)}}, 1) \\ \vdots & & \vdots \\ a_{|T|,1}([\boldsymbol{\Delta_1 x^{(|T|)}}, 1) & \dots & a_{|T|,|V|}([\boldsymbol{\Delta_{|V|} x^{(|T|)}}, 1) \end{pmatrix} \begin{pmatrix} j_{k,1}^{(1)} \\ \vdots \\ j_{k,l_1}^{(1)} \\ o_k^{(1)} \\ \vdots \\ j_{k,1}^{(|V|)} \\ \vdots \\ j_{k,l_{|V|}}^{(|V|)} \\ o_k^{(|V|)} \end{pmatrix}, \tag{2.34}$$

wobei $a_{i,j}$ die Einträge in der Aktivierungsmatrix $\boldsymbol{A}$ mit

$$a_{i,j} = \begin{cases} 1 & \text{Neuron } i \text{ ist naheliegendster Knoten zu Eingabe } \boldsymbol{x}^{(j)} \\ \\ 0 & \text{sonst} \end{cases} \tag{2.35}$$

sind. Gleichung 2.34 ist damit das wohl bekannte lineare Ausgleichsproblem. Man erhält m solcher Ausgleichsprobleme und bestimmt so die Einträge der Jacobi-Matrizen und der Ausgabevektoren stückweise nach den Komponenten der Ausgabe und setzt sie anschließend zu den Matrizen bzw. Vektoren zusammen. Der Vorteil an Gleichung 2.34 liegt darin, daß damit nicht nur die "winner takes all"- Strategie realisiert werden kann, sondern auch andere Strategien wie z.B. die Vorgängerstrategie "bmu und Nachbarn". In jeder Zeile der Gleichungen tauchen die Aktivierungen aller Knoten auf. Im Fall der "winner takes all"-Strategie ist in jeder Zeile genau eine Aktivierung gleich eins und alle anderen null. Der allgemeinere Fall, daß mehrere Neuronen von einem Trainingsdatum aktiviert werden, also mehrere Aktivierungen in einer Zeile ungleich null sind, stellt bei dieser Darstellung des Ausgleichsproblems kein Problem dar.

2.7.3 Versuche

Auch in diesem letzten Abschnitt soll der weiterentwickelte Netztyp an einem künstlichen Experiment getestet werden. In diesem Experiment sollen noch einmal die beiden Hauptprobleme auftreten, die sich in den vorausgegangen Experimenten hervorgetan haben: Rauschen in den Daten und ein komplizierter Verlauf der Daten im Eingaberaum.

Als Versuchsbeispiel dient daher wieder das Streckenlängenproblem aus Abschnitt 2.5.3 mit den dort verwendeten Radien. Die Figur, die die Datenpunkte in einer zweidimensionalen Ebene beschreibt, wird in einen höherdimensionalen Raum eingebettet, wie es schon im vorherigen Kapitel getan wurde. Die Daten werden so in dem Experiment zu 10-dimensionalen Eingabevektoren erweitert. Die Ausgabe ist dann wieder die Streckenlänge von Startpunkt der Figur bis zum Eingabevektor. Die Daten erhalten einen Rauschanteil in den Komponenten orthogonal zur Figurebene und auch in der Figurebene, jedoch dort mit einem Rauschen senkrecht zur Tangente des jeweils zu verrauschenden Punktes. Es wird wiederum von zwei Datenmengen ausgegangen, einer Trainings- und einer Testmenge. Die Eingabevektoren der Trainingsmenge sind mit einer Rauschamplitude von $\pm$ 0.4 Einheiten verrauscht, die der Testmenge mit einer Amplitude von $\pm$ 0.8 Einheiten. Der Versuch soll das SDCS-Netz mit linearer Erweiterung mit einem RBF-Netz und mit einem SDCS-Netz ohne Erweiterung vergleichen. Es werden die drei Netztypen mit der Trainingsmenge trainiert, einmal unter a priori Vorgabe der richtigen Nachbarschaftsbeziehungen, einmal ohne Vorgabe, so daß die Netze diese lernen müssen. Weiterhin wird ein RBF-Netz mit adaptierten Varianzen verwendet, so daß auch für diesen Netztyp optimale Bedingungen gewährleistet sind. Anschließend folgt ein Test der Netze auf den Testdaten. Die Ergebnisse der Versuche stellen sich in den Graphen der Abbildungen 2.12 und 2.13 dar. Es zeigt sich, daß das SDCS-Netz mit linearer Erweiterung den anderen beiden Netztypen deutlich überlegen ist, sowohl im Fall der vorgegebenen Topologie als auch der gelernten. Gerade das gute Ergebnis im zweiten Fall ist eine interessante Erkenntnis, denn obwohl die SDCS-Netze die Topologie erst nach 35 Schritten exakt gelernt haben, kann das SDCS-Netz mit linearer Erweiterung schon vorher die Funktion sehr genau approximieren. Das liegt daran, daß, wenn eine falsche Verbindung erzeugt wird, nur wenige Trainingsdaten in Richtung der falschen Verbindung liegen und somit die falsche Richtung nur einen geringen Einfluß auf die betreffenden Jacobi-Matrizen hat. Die partielle Ableitung in diese Richtung beträgt damit näherungsweise null. Der Negativeffekt einer falschen Verbindung wird damit stark abgeschwächt. Betrachtet man dagegen das Ergebnis des SDCS-Netzes ohne Erweiterung, erkennt man, daß eine ungenau gelernte Topologie dort zu großen Fehlern in der Approximation führt. Der Fehler erreicht erst einen akzeptablen Wert, wenn die Topologie richtig erlernt wurde.

Es ist somit ein Verfahren entwickelt worden, welches in der Lage ist, den gefunden Unterraum besser auszunutzen, indem es versucht, die gesuchte Funktion entlang dieses Unterraumes lokal linear zu approximieren. Die Versuchsergebnisse an künstlichen Daten haben gezeigt, daß dieses Verfahren unter unterschiedlichen Bedingungen eine gute Approximationsgüte besitzt, da es sowohl gegen Rauscheinflüsse in den Daten als auch gegen Defekte in der gelernten Topologie robust ist. Die Ergebnisse motivieren den Einsatz dieses Netztyps zur Objekt- und Posenbestimmung aus Realdaten in den folgenden Kapiteln.

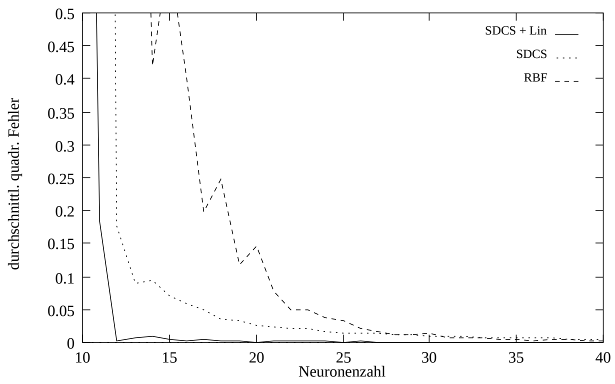

Abbildung 2.12: Approximationsfehler der drei Netztypen bei a priori vorgegebener Topologie. Links: Fehler auf den Trainingsdaten. Rechts: Fehler auf den Testdaten.

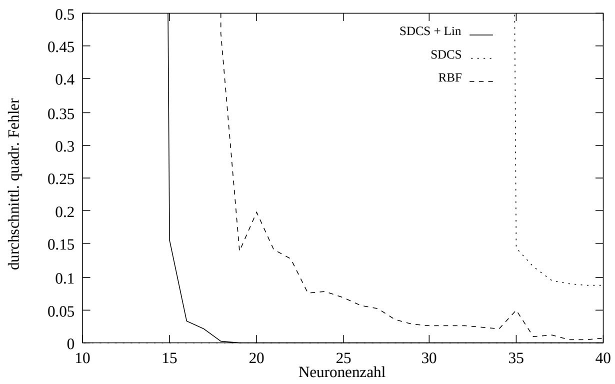

Abbildung 2.13: Approximationsfehler der drei Netztypen bei autonom gelernter Topologie. Links: Fehler auf den Trainingsdaten. Rechts: Fehler auf den Testdaten.

Kapitel 3

Objekterkennung und Posenbestimmung

3.1 Einführung

Die Grundproblemstellung dieser Arbeit ist es, eine Roboterhand Objekte greifen zu lassen, wobei der Greifvorgang visuell gesteuert werden soll. Damit stellen sich drei Grundanforderungen an die Bildauswertung: Die *Objekterkennung, -lokalisation und Posenbestimmung.* Jedes Problem für sich ist ein großes Forschungsgebiet der künstlichen Intelligenz und wurde schon in unzähligen Arbeiten behandelt. Dieses Kapitel beschränkt sich auf die Objekterkennung und Posenbestimmung; ein Verfahren zur Objektlokalisation, welche für den Greifvorgang natürlich auch zwingend notwendig ist, wird in Kapitel 5 vorgestellt.
Die Erscheinung eines Objektes in einem Bild hängt zum einen von der Form des Objektes und seiner Lichtreflektierungseigenschaften ab, Eigenschaften, die konstant und unabhängig von der Szene für ein Objekt sind. Zum anderen wird die Erscheinung durch die Lage des Objektes im Raum bestimmt, also ein Parameter, der von Szene zu Szene variieren kann. Man benötigt daher für ein Objekt eine Menge von repräsentativen Ansichten, die für ein Objekt eine vollständige Beschreibung aller vorkommenden Posen ermöglicht. Die weiterführende Aufgabe ist es, aus diesen Ansichten oder Bildern Merkmale zu extrahieren, die prägnant genug sind, um das Objekt in einer Menge von anderen Objekten zu unterscheiden und auch eine Zuordnung zu einer bestimmten Pose oder Orientierung im Raum zuzulassen. Man wendet dieses Prinzip für verschiedene Objekte an und erhält so eine Objektdatenbank. Die eigentliche Objekterkennung und Posenbestimmung ist dann der Vergleich von Szenedaten mit Daten der Objektdatenbank.

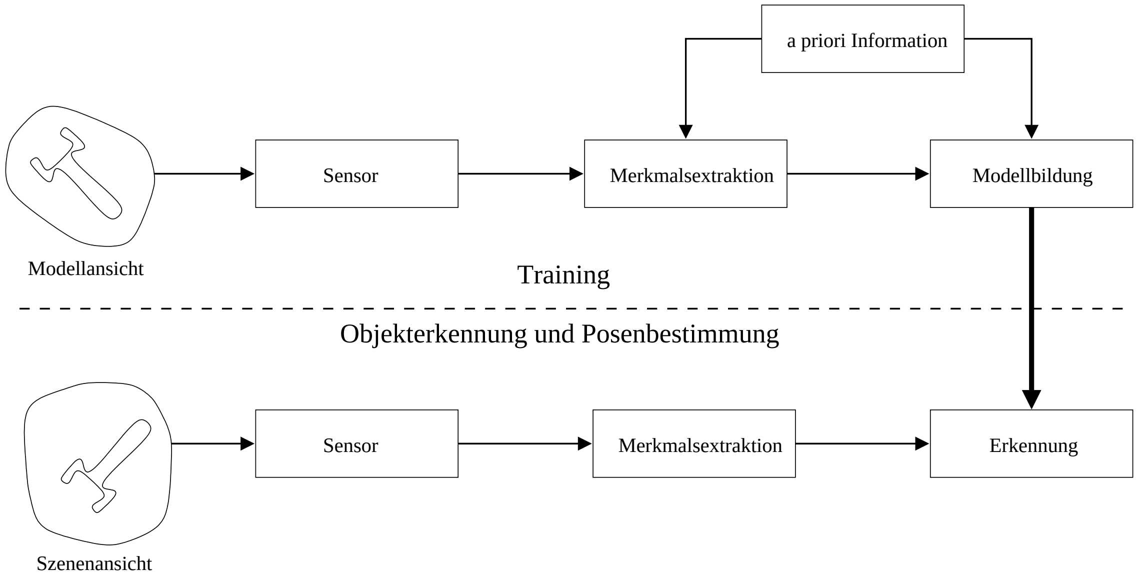

Abbildung 3.1: Die zwei Phasen einer Objekterkennung und Posenbestimmung

3.1.1 Klassische Verfahren

Die Aufgabe der Objekterkennung und Posenbestimmung ist es, ein gelerntes Objekt in der Szene wiederzuerkennen und dessen Lage zu beschreiben. Der Vorgang spaltet sich in zwei Phasen. In der Trainingsphase werden Merkmale bestimmt und extrahiert, die eine hinreichende Beschreibung des zu lernenden Objektes in verschiedenen Posen liefern. Man erstellt so eine *Objekt-* oder *Modelldatenbank* für eine Reihe von Objekten. In der zweiten Phase, der eigentlichen Objekterkennung, geschieht ein Vergleich zwischen Merkmalen, die aus der Szene extrahiert wurden und Merkmalen der Modelldatenbank. Man spricht daher auch von *modellbasierter* Objektbeschreibung und -erkennung [20].
Die Literatur unterscheidet zwischen zwei generellen Arten von Objekterkennung, der *Ansichten basierten Erkennung* und der Erkennung durch *symbolische Beschreibung*. Die symbolische Beschreibung stützt sich auf geometrische Eigenschaften des Objektes wie Flächen, Kanten, Winkeln zwischen Kanten u.s.w.. Diese Verfahren eignen sich, wenn ein geometrisches Modell des Objektes erstellt werden kann oder sogar schon existiert z.B in Form von CAD-Daten [6]. Dabei hat dieses Vorgehen zwei entscheidende Nachteile. Zum einen geschieht die Repräsentation des Objektes in einer Datenbank nicht autonom, sondern die Merkmalswahl wird dem System von außen durch den Benutzer vorgegeben. Es müssen also a priori Informationen über das Objekt vorhanden sein. Zum anderen wird durch die Konzentration auf die geometrischen Gegebenheiten die Erscheinung des Objektes in ihrer Gesamtheit vernachläßigt, und somit geht Information verloren.
Die Ansichten basierte Objektbeschreibung stützt sich auf die *photometrischen Eigenschaften* eines Bildes, das heißt die Abbildung des Objektes auf ein Computerbild, wobei die Pixelwerte (bei einen Grauwertbild Werte zwischen 0 (schwarz) bis 255 (weiß)) das Bild beschreiben. Ein einfaches Verfahren zur Objekterkennung und Posenbestimmung ist dann

der pixelweise Vergleich eines Szenebildes mit Bildern einer Objektdatenbank, ein sogenanntes *Matching*. Für ein gutes Match-Ergebnis benötigt man dabei zum einen nahezu identische Rahmenbedingungen in der Szene und der Modellbildung, zum anderen ein große Anzahl von Ansichten. Auch der Vergleich selbst ist sehr aufwendig. Geht man z.B. von einem Grauwertbild der Größe 128×128 Pixel aus, so hätte der zugehörige Merkmalsvektor 16384 Komponenten.
Ein effizienteres Verfahren ist die Anwendung einer *attentiven Strategie*. Die Methode basiert auf psychologischen Untersuchungen, die beweisen, daß das visuelle System beim Menschen durch Bewegungsstrategien z.B. bei der Objektsuche nur die visuell auffälligen Bildbereiche betrachtet [9]. Die Auffälligkeiten können Helligkeiten, Kontraste, Kanten und ähnliches sein. Es wird also nicht mehr das gesamte Bild betrachtet, sondern nur bestimmte Bereiche.
Murase und Nayar [14] beschreiben eine neue Möglichkeit der Objekterkennung und Posenbestimmung, die von der klassischen Idee der modellbasierten Erkennung abweicht. Sie gehen zwar ebenfalls von einer Menge von Ansichten eines Objektes aus, führen jedoch eine Hauptachsenanalyse auf der gesamten Bildmenge durch. Sie erhalten so eine Menge von Eigenvektoren. Die prägnantesten Eigenvektoren bilden den sogenannten *Eigenraum*. Die einzelnen Ansichten werden in diesen Eigenraum projiziert und bilden dort Punkte auf einer Mannigfaltigkeit, die in dem Fall durch die Parameter Orientierung und Beleuchtungsrichtung bestimmt wird. Die Mannigfaltigkeit wird anschließend über eine kubische Splineinterpolation approximiert. Die Posenbestimmung geschieht dann so, daß eine beliebige Szenenansicht ebenfalls in den Eigenraum projiziert wird und die Position bezüglich der Mannigfaltigkeit, an der sich der im Eigenraum erzeugte Punkt befindet, die Orientierung angibt.

3.1.2 Das in der Arbeit verwendete Verfahren: Projektion der Bilddaten in einen Unterraum durch Bildfaltung

Das Vorgehen in dieser Arbeit folgt dem von Murase und Nayar, so daß die Bilddaten ebenfalls in einen speziellen Raum projiziert werden, er wird als *Merkmalsraum* bezeichnet. Die Orientierung der Objekte in den Ansichten besitzt nur einen Freiheitsgrad, nämlich eine beliebige Rotation des Objektes um eine vertikale Achse. Es soll dabei die Eigenschaft erreicht werden, daß sich die Bilddaten durch eine geeignete Projektion auf einer Mannigfaltigkeit im Merkmalsraum abbilden, welche nur durch einen Parameter, eine Rotation oder Orientierung des Objektes um eine vertikale Achse, bestimmt wird. Bildet man die Ansichtenmengen von verschiedenen Objekten in den Merkmalsraum ab, so soll jedes Objekt eine eigene Mannigfaltigkeit erzeugen. Die Approximation der Mannigfaltigkeiten und die darauf basierende Posenbestimmung geschieht jedoch nicht über Spline-Interpolation, sondern durch neuronale Netzwerke. Kapitel 2 hat gezeigt, daß, wenn die Eingabedaten eines neuronalen Netzes sich auf einer Mannigfaltigkeit befinden und diese Mannigfaltigkeit mittels einer topologischen Karte bestimmt werden kann, die Ausnutzung dieser Information die Approximationsgüte eines Netzes verbessert. Daher bietet es sich an, ein SDCS-Netz

mit linearer Erweiterung für die Posenbestimmung zu verwenden.
Ein einfaches Verfahren wäre es, den Bildraum gleichzeitig als Merkmalsraum zu wählen, also jedes Pixel als eine Komponente des Projektionsvektors zu betrachten. Nicht die extrem hohe Dimension des Raumes allein ist der Grund, sich in dieser Arbeit nicht für diese einfache Variante zu entscheiden, denn Kapitel 2 hat gezeigt, daß ein SDCS-Netz auch in hochdimensionalen Eingaberäumen effizient und effektiv funktioniert, wenn die intrinsische Dimension wesentlich geringer ist. Der Grund ist ein anderer: Der Merkmalsraum soll unter anderem die Eigenschaft besitzen, daß, wenn zwei ähnliche Ansichten in den Merkmalsraum projiziert werden, die erzeugten Vektoren dort dicht beieinander liegen. Betrachtet man die Bilddaten zweier Objektansichten, zwischen denen sich die Orientierung des Objektes nur geringfügig geändert hat, so zeigt sich beim Vergleich der individuellen Pixel, daß dort größere Änderungen in den Werten auftreten konnen. Wählt man den Bilddraum gleichsam als Merkmalsraum, also jedes Pixel als eine Komponente der Merkmalsvektors, so besitzen ähnliche Ansichten damit im allgemeinen sehr unterschiedliche Merkmalsvektoren. Die gewünschte Eigenschaft ist somit nicht erfüllt, da die Unterschiede der individuellen Pixelwerte zu groß sind.
Um dieses Problem zu beheben, kann man geglättete Bilddaten verwenden. Ein typisches Verfahren zur Bildglättung ist die Faltung mit Glättungs- oder Tiefpaßfiltern. Solche Filter werden eingesetzt, um Rauschen oder feine Details zu unterdrücken. Rechteck-, Binomial- und Gaußfilter sind Vertreter dieser Klasse. Der hier verwendete Faltungsoperator soll aber zwei andere Eigenschaften besitzen. Als erstes muß er einen Beitrag zur Objekterkennung leisten, denn es soll erreicht werden, daß die Mannigfaltigkeiten von verschiedenen Objekten im Merkmalsraum weit voneinander entfernt liegen. Objekte von unterschiedlicher Struktur verursachen verschiedene Wellenzahlbereiche im Bild. Es wird also ein Filter benötigt, das einen bestimmten Wellenzahlbereich selektiert, ein sogenanntes Bandpaßfilter. Als zweites muß das Filter für die Posenbestimmung orientierungsselektiv sein. Ein *Gaborfilterpaar* hat diese Eigenschaften, es gehört in die Klasse der orientierungsselektiven Bandpaßfilter. Um einen nicht zu hochdimensionalen Merkmalsraum zu erhalten, geschieht die Faltung nicht wie normalerweise üblich an jeder Position des Bildes, sondern es werden rasterförmig *Ortspunkte* im Bild bestimmt, an denen die Faltungoperatoren aufgesetzt werden. Es werden dabei Gaborfilterpaare mit verschiedenen Orientierungen und Durchlaßbereichen verwendet. Hat man also n verschiedene Gaborfilterpaare generiert und setzt diese an m festen Positionen im Bild auf, so erhält man $n*m$ Filterantworten für ein Bild. Diese bilden die Komponenten des Projektionsvektors der Ansicht im Merkmalsraum. Im folgenden wird der Projektionsvektor auch als *Merkmalsvektor* bezeichnet.

3.2 Vorverarbeitung des Bildes

Um eine einheitliche Verarbeitung der Ansichten von Objekten zu Merkmalsvektoren zu gewährleisten, ist es notwendig, die Bilddaten vorzubearbeiten. Dabei soll erreicht werden, daß die Position und die Größe eines Objektes im Bild konstant ist. Dafür ist zunächst eine

Segmentierung des Bildes in Objektregion und Hintergrundregion nötig. Es wird davon ausgegangen, daß der Hintergrund die niedrigste Helligkeit im Bild besitzt, damit kann die Segmentierung über ein Schwellwertverfahren auf den Pixelgrauwerten $g(x,y)$ durchgeführt werden.

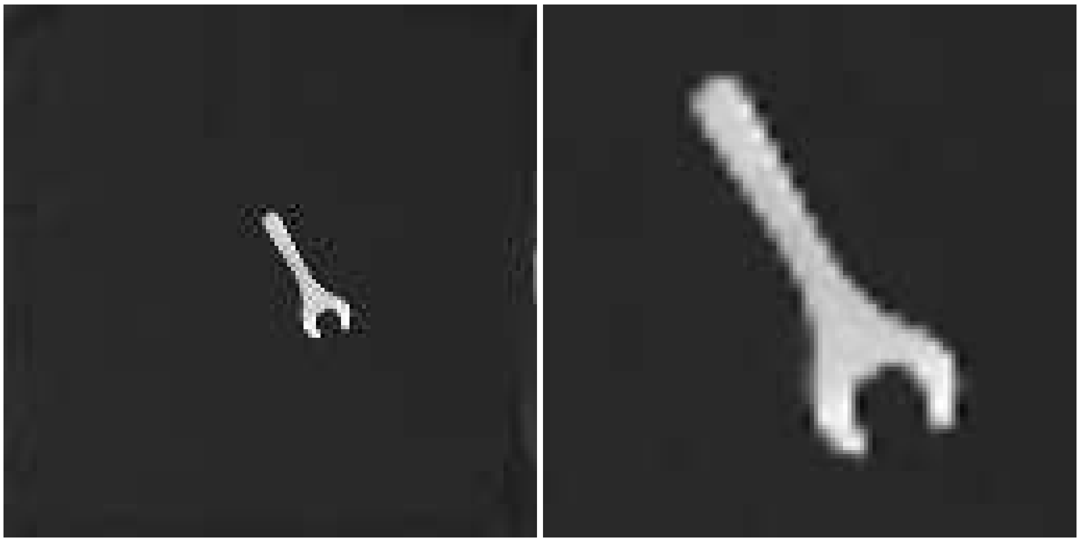

Abbildung 3.2: Links: Originalbild der Szene. Rechts: Vorverarbeitetes Bild nach Verschiebung in den Mittelpunkt und anschließender Skalierung.

3.2.1 Translation in den Mittelpunkt

Die konstante Position eines Objektes im Bild soll so gewählt sein, daß der Pixelschwerpunkt des Objektes mit dem Mittelpunkt des Bildes koinzidiert. Hat das Grauwertbild eine Größe von $N \times M$ Pixeln und ist t der vorgegebene Pixelschwellwert, dann berechnet sich die momentane Position (x_s, y_s) des Pixelschwerpunktes durch

$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} = \frac{1}{K} \sum_{g(x,y) \geq t} \begin{pmatrix} x \\ y \end{pmatrix}, \tag{3.1}$$

wobei K die Anzahl der Pixelwerte ist, die größer als der Schwellwert t sind. Es ist $(x_m = \frac{N}{2}, y_m = \frac{M}{2})$ der Mittelpunkt des Bildes und damit

$$\boldsymbol{T} = \begin{pmatrix} x_m \\ y_m \end{pmatrix} - \begin{pmatrix} x_s \\ y_s \end{pmatrix} \tag{3.2}$$

der Translationsvektor für die Verschiebung des Pixelschwerpunktes in den Mittelpunkt des Bildes.

3.2.2 Skalierung

Um eine einheitliche Größe aller Objekte zu erreichen, wird ein Kreis mit konstantem Radius r_c um den Mittelpunkt (x_m, y_m) definiert. Dieser Kreis beschreibt die einheitliche Ausdehnung der Objekte. Ein zweiter Kreis mit Radius r_a beschreibt die aktuelle Ausdehnung des Objektes im Bild. Der Kreis hat seinen Mittelpunkt ebenfalls im Mittelpunkt des Bildes, und r_a ist der maximale Radius, so daß der zugehörige Kreis mindestens ein Pixel des Objektes im Bild berührt. Damit ergibt sich der Faktor der Skalierung s durch

$$s = \frac{r_c}{r_a}. \tag{3.3}$$

Die Skalierung der Bilddaten geschieht über *bilineare Interpolation*.

3.2.3 Helligkeitsinvarianz

Für die Beleuchtung der Szene befinden sich die Beleuchtungsquellen an festen Positionen, welche nicht variiert werden sollen. Das System soll aber invariant gegenüber Helligkeitsschwankungen sein. Eine Helligkeitsänderung verhält sich multiplikativ auf den Pixelwerten und verändert damit die Amplituden der Wellen im Bild, was zu veränderten Ergebnissen in der Faltung führt.
Dieses Problem läßt sich folgendermaßen lösen: Nimmt man zwei Bilder mit Pixelwerten $g(x, y)$ und $\tilde{g}(x, y)$, die sich nur durch einen Helligkeitsfaktor c unterscheiden und logarithmiert [1] diese Pixelwerte, so erhält man:

$$\log p(x, y) = \log(\tilde{p}(x, y) * c) = \log(\tilde{p}(x, y)) + \log c.$$

Auf den logarithmierten Pixelwerten verhalten sich Helligkeitsänderungen also additiv. Hiermit und aufgrund der Gleichanteilsfreiheit des Gaboroperators ist gewährleistet, daß Helligkeitsänderungen keine Änderungen der Faltungsantworten bewirken (siehe dazu auch 3.3.3). Man erhält so durch Logarithmieren der Bilddaten ein helligkeitsinvariantes System.

Damit ist die vorverarbeitende Bildverarbeitung abgeschlossen, das Bild hat eine einheitliche Größe und Position im Bild. Nun kann die eigentliche Bildverarbeitung, die Faltung des Bildes mit den Gaboroperatoren vorgenommen werden.

---

[1] Da für den Wert null der Logarithmus nicht definiert ist, addiert man einheitlich zu den Pixelwerten den Wert eins.

3.3 Das Gaborfilter

3.3.1 Mathematische Funktion

Die Gaborfilter gehören in die Klasse der Quadraturfilterpaare. Ein Gaborfilter ist eine gaußmodulierte komplexharmonische Funktion $h(x)$ [18], verknüpft mit der um die Phase $\pm\pi/2$ verschobenen Funktion $\tilde{h}(\boldsymbol{x})$, der Hilbertransformierten von $h(\boldsymbol{x})$. Ein zweidimensionales Gaborfilterpaar läßt sich damit darstellen[2] als Paar einer gaußmodellierten Cosinusfunktion

$$G^{+}_{\boldsymbol{\Sigma},\boldsymbol{k},\boldsymbol{R}} = \exp\left(-\frac{1}{2}[\boldsymbol{Rx}]^T\boldsymbol{\Sigma}[\boldsymbol{Rx}]\right)\cos\left(2\pi\boldsymbol{k}(\boldsymbol{Rx}^T)\right), \tag{3.4}$$

dem geraden Anteil, und einer gaußmodellierten Sinusfunktion

$$G^{-}_{\boldsymbol{\Sigma},\boldsymbol{k},\boldsymbol{R}} = \exp\left(-\frac{1}{2}[\boldsymbol{Rx}]^T\boldsymbol{\Sigma}[\boldsymbol{Rx}]\right)\sin\left(2\pi\boldsymbol{k}(\boldsymbol{Rx}^T)\right), \tag{3.5}$$

dem ungeraden Anteil. Als Antwort des Filters wird der lokale Betrag des Bildes g($\boldsymbol{x}$) in einem Punkt $\boldsymbol{x}$ gewählt, also

$$G(\boldsymbol{x})_{\boldsymbol{\Sigma},\boldsymbol{\lambda},\boldsymbol{R}} = \sqrt{(G^{+}_{\boldsymbol{\Sigma},\boldsymbol{\lambda},\boldsymbol{R}}(\boldsymbol{x}) * g(\boldsymbol{x}))^2 + (G^{-}_{\boldsymbol{\Sigma},\boldsymbol{\lambda},\boldsymbol{R}}(\boldsymbol{x}) * g(\boldsymbol{x}))^2}. \tag{3.6}$$

Es ist $\boldsymbol{x} = (x, y)^T$ der Ortsvektor. Das Design des Filters wird über folgende Parameter gesteuert:

1. Der Wellenzahlvektor $\boldsymbol{k} = (\frac{1}{\lambda_x}, 0)^T$. Der Parameter λ_x gibt die Wellenlänge des Sinus' und des Cosinus' in x-Richtung an und bestimmt damit den Durchlaßbereich des Filters. Im weiteren spricht man von der *Skala* des Filters.

2. Die Varianzmatrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} \frac{1}{\sigma_x^2} & 0 \\ 0 & \frac{1}{\sigma_y^2} \end{pmatrix}. \tag{3.7}$$

   Die Parameter σ_x und σ_y geben die Varianz der Gaußfunktion in x- bzw. y-Richtung an.

3. Die Rotationsmatrix

$$\boldsymbol{R} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}. \tag{3.8}$$

   Der Parameter ϕ gibt dem Filter seine *Orientierung*.

Die Wellenlänge λ_x und die Orientierung ϕ sind die für die Aufgabe hauptsächlich interessanten Parameter, die Varianzen σ_x und σ_y bestimmen die Charakteristik des Filters.

[2] Im Ortsraum

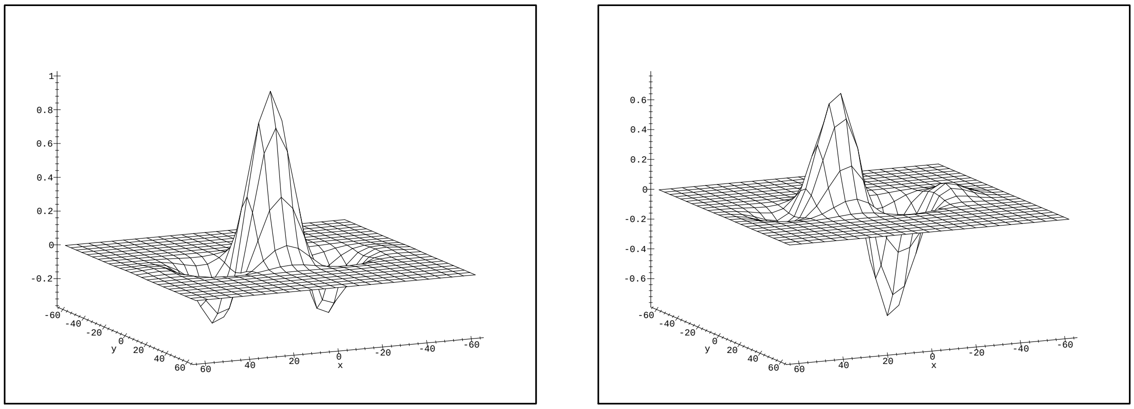

Abbildung 3.3: Graphen eines zweidimensionalen Gaborpaars. Links: Graph der geraden Funktion. Rechts: Graph der ungeraden Funktion.

3.3.2 Abhängigkeiten zwischen den Parametern

Theoretisch können die Parameter beliebige[3] Werte annehmen. Da der Filter eine gewünschte Charakteristik besitzen soll, sind gewisse Abhängigkeiten zwischen den Parametern zu beachten. Dieses gilt im besonderen für das Zusammenspiel der Parameter Wellenlänge und Varianz in x-Richtung. Es wird daher eine *Kopplungskonstante* s eingeführt. Wählt man eine bestimmte Wellenlänge λ_x, so berechnet sich die zugehörige Varianz σ_x durch

$$\sigma_x = \frac{\lambda_x}{s}. \tag{3.9}$$

Mit Hilfe von Gleichung 3.9 erhält man also Filter mit beliebiger Wellenlänge, deren Charakteristik jedoch sehr ähnlich ist. Versuche haben gezeigt, daß mit s in einem Wertebereich von $(0, 1]$ die Gaborfunktionen ihre typische Charakteristik erhalten. Dabei hat sich der Wert $s = 0.5$ bewährt, dieser wurde auch für die Funktionen in Abbildung 3.3 verwendet.

3.3.3 Gleichanteilsfreiheit

Eine notwendige Bedingung für ein Bandpaßfilter ist, daß es *gleichanteilsfrei* ist, da es ansonsten die Nullfrequenz durchlassen würde, das heißt auf konstanten Bildern einer Faltungsantwort ungleich null liefern würde. Gleichanteilsfrei bedeutet, daß die Funktionen sich zu null integrieren, also hier

$$\int_{x=-\infty}^{\infty}\int_{y=-\infty}^{\infty} G^{+}_{\boldsymbol{\Sigma},\boldsymbol{k},\boldsymbol{R}}(x,y)\,dx\,dy = \int_{x=-\infty}^{\infty}\int_{y=-\infty}^{\infty} G^{-}_{\boldsymbol{\Sigma},\boldsymbol{k},\boldsymbol{R}}(x,y)\,dx\,dy = 0$$

[3] Die Werte müssen dabei natürlich plausibel sein, also λ_x größer null und σ_x bzw. σ_y ungleich null.

gelten muß. Der ungerade Anteil des Gaborfilters ist definitionsgemäß immer gleichanteilsfrei, da die Funktion punktsymmetrisch zum Koordinatenursprung verläuft. Beim geraden Anteil ist dies jedoch im allgemeinen nicht der Fall. Es ist dann nach 3.4

$$\int_{x=-\infty}^{\infty}\int_{y=-\infty}^{\infty} G^{+}_{\boldsymbol{\Sigma},\boldsymbol{k},\boldsymbol{R}}(x,y)dxdy = \int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\frac{1}{2}[\boldsymbol{Rx}]^T\boldsymbol{\Sigma}[\boldsymbol{Rx}]\right)\cos\left(2\pi\boldsymbol{k}(\boldsymbol{Rx}^T)\right) d\boldsymbol{x} \neq 0. \tag{3.10}$$

Es ist daher notwendig, die Cosinusfunktion um ein Skalar d zu verschieben [13], so daß gilt:

$$\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\frac{1}{2}[\boldsymbol{Rx}]^T\boldsymbol{\Sigma}[\boldsymbol{Rx}]\right)\left(\cos\left(2\pi\boldsymbol{k}(\boldsymbol{Rx}^T)\right) - d\right)d\boldsymbol{x} = 0.$$

Der folgende Satz gibt eine Berechnung des Skalars d an:

Satz 2 *Es gilt:*

$$\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\tfrac{1}{2}[\boldsymbol{Rx}]^T\boldsymbol{\Sigma}[\boldsymbol{Rx}]\right)\left(\cos\left(2\pi\boldsymbol{k}(\boldsymbol{Rx}^T)\right) - d\right)d\boldsymbol{x} = 0$$
$$\Leftrightarrow$$
$$d = \frac{\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\frac{1}{2}[\boldsymbol{Rx}]^T\boldsymbol{\Sigma}[\boldsymbol{Rx}]\right)\cos\left(2\pi\boldsymbol{k}(\boldsymbol{Rx}^T)\right)d\boldsymbol{x}}{\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\frac{1}{2}[\boldsymbol{Rx}]^T\boldsymbol{\Sigma}[\boldsymbol{Rx}]\right)d\boldsymbol{x}}$$

Beweis: *Siehe Anhang A.*

Das Skalar d bildet sich also aus dem Quotienten des Integrals der ursprünglichen Gaborfunktion und des Integrals der Gaußfunktion.

3.4 Die Faltungsmaske

Bisher wurde die Gaborfunktion als kontinuierliche zweidimensionale Funktion mit infinitem Definitionsbereich betrachtet. Zur Faltung benötigt man jedoch eine zweidimensionale, quadratische Faltungsmaske mit einer endlichen Ausdehnung N in x- und y-Richtung. Die Gaborfunktion muß daher im Bereich von $-N/2$ bis $N/2$ für x und y diskret abgetastet werden, um so die Elemente der Faltungsmaske zu erhalten. Man kann sich die Faltungsmaske als Pixelbild vorstellen, bei dem die Pixel jedoch, anders als bei einem Grauwertbild, einen unbeschränkten, reellen Wertebereich haben. Damit läßt sich für die Parameter des Gaborfilters eine Einheit definieren, die Skala und die Varianzen des Gaborfilters werden von nun an in Pixeln angegeben.
Da die Funktion nunmehr diskret abgetastet wird, kann bei der Gleichanteilsbestimmung keine Integration durchführt werden. Stattdessen summiert man die diskret abgetasteten Funktionswerte der Gaborfunktion und des Gaußanteils auf und erhält so in Anlehnung an Satz 2 das Verschiebungsskalar d.

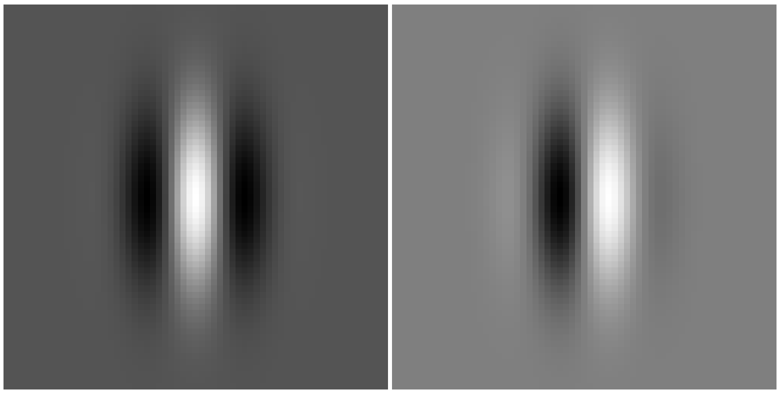

Abbildung 3.4: Die Faltungsmasken eines Gaborpaars bei einer Orientierung von 0 Grad. Links: Der gerade Gabor. Rechts: Der ungerade Gabor.

3.4.1 Randbehandlung

Die Randbehandlung ist eine entscheidende Problematik in der Bildfaltung. Pixel am Rand eines Bildes besitzen keine vollständige Umgebung, somit muß diese Umgebung künstlich erzeugt werden. Das einfachste Verfahren ist das Erweitern des Bildes mit Nullen. Dies ist aber gleichbedeutend mit dem Abschneiden von Maskenelementen, so daß die Bedingung der Gleichanteilsfreiheit der Maske verletzt werden kann. Ein geeigneteres Verfahren ist daher, das Bild durch Auffüllen mit der ersten bzw. letzten Zeile und Spalte ausreichend zu erweitern.
In dieser Arbeit bedient man sich der zyklischen Faltung. Man betrachtet dabei das Bild nicht mehr als endliche Fläche, sondern stellt es für x- und y-Richtung als toridiales Gebilde der Spalten bzw. Zeilen dar, so daß kein Randproblem auftritt.

3.4.2 Faltung des Bildes an ausgewählten Ortspunkten

Wie schon in der Einleitung des Kapitels bemerkt, wird nicht das gesamte Bild gefaltet, sondern es geschieht die Faltung an ausgewählten Ortspunkten, um somit das Problem zu großer Merkmalsvektoren zu umgehen. Die Ortsvektoren oder Aufpunkte der Faltung werden rasterförmig und äquidistant in x- und y-Richtung über das gesamte Bild gewählt, um zu gewährleisten, daß man einen repräsentativen Ausschnitt einer „echten" Bildfaltung erhält.

3.4.3 Faltungsergebnisse

In den vorangegangenen Abschnitten wurde ein Faltungsoperator entwickelt, der die gewünschten Eigenschaften besitzt. Es ist ein Bandpaßfilter mit orientierungsselektiver Eigenschaft. Die folgenden Bilder zeigen Faltungsergebnisse mit einem Gaborpaar mit konstanter Orientierung null auf drei verschiedenen Skalen. Es soll damit an zwei verschiedenen Objekten in verschiedenen Lagen verdeutlicht werden, wie sich die Filterantworten verhalten, um zu analysieren, ob sich die gewählten Filterantworten sowohl zur Objekterkennung als auch zur Posenbestimmung eignen. Es zeigt sich, daß die Antworten für verschiedene Objekte große Diskrepanzen aufweisen und so zur Diskriminierung von Objekten geeignet sind. Für die Posenbestimmung ergibt sich, daß sich für kleine Skalen der Gabormaske Kanten schärfer herausbilden. Dadurch werden jedoch größere Änderungen in den Filterantworten zwischen zwei aufeinanderfolgenden Ansichten erzeugt, was für die spätere Posenapproximation einen negativen Effekt hat. Gaborfilter mit größeren Skalen verwischen die Kanten mehr. Es hat sich herausgestellt, daß die Gaborfilter eine minimale Skala von 7 Pixeln haben sollten, da ansonsten zu große Sprünge in den Filterantworten zweier aufeinanderfolgender Ansichten auftreten. Man beachte jedoch auch, daß ein Gaborfilter mit zu großer Skala im Verhältnis zur Objektgröße im Bild seine Orientierungsselektivität verliert.

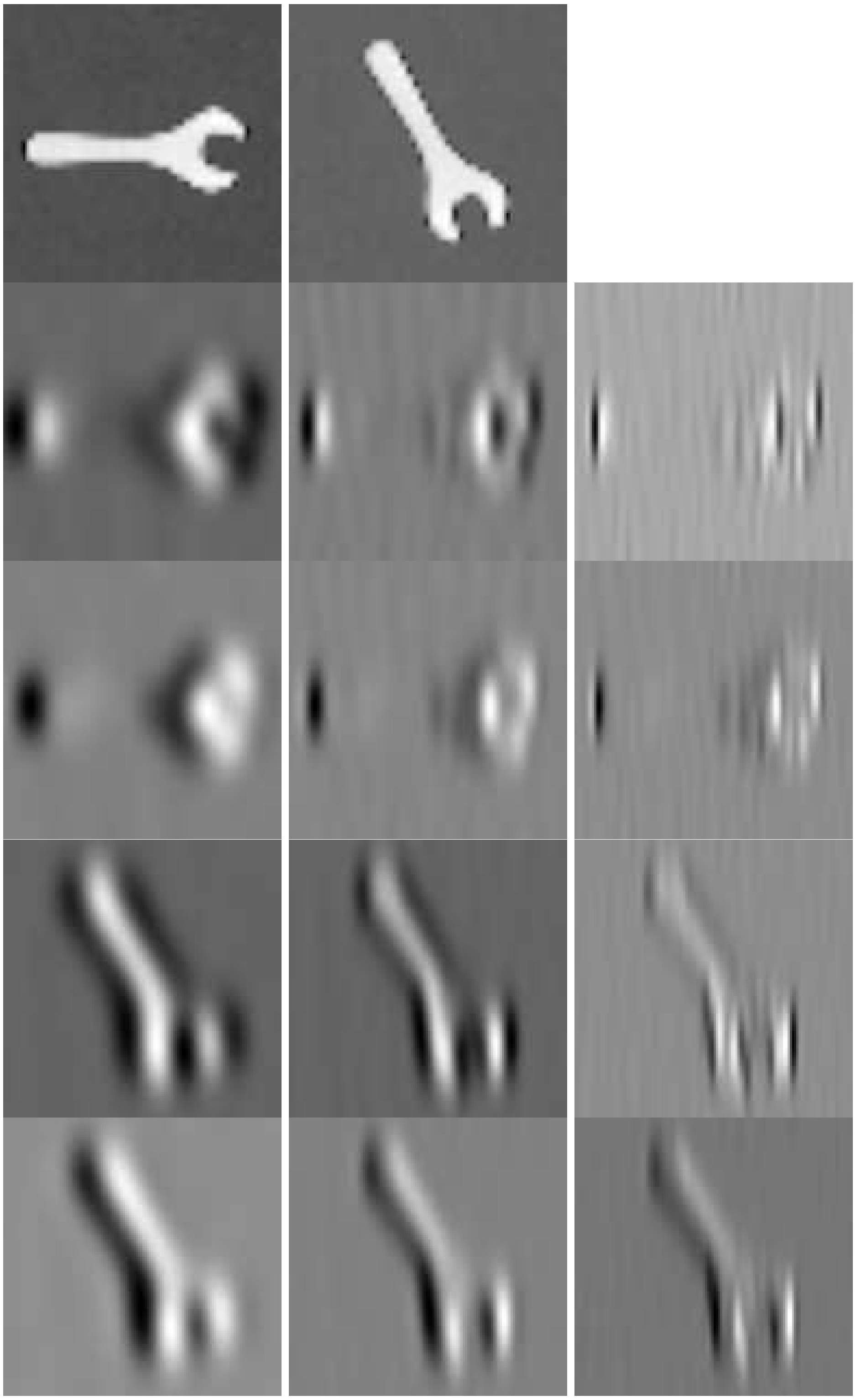

Abbildung 3.5: Faltungsergebnisse für Objekt „Schraubenschlüssel“. Obere Reihe: Das Objekt in zwei Orientierungen. Darunter in zwei Doppelreihen von links nach rechts: Die Faltungsergebnisse auf den beiden Ansichten mit einem Gaborpaar mit den Skalen 30, 15 und 7 Pixeln. Oberes Bild einer Doppelreihe: Antwort des geraden Gaborfilters. Unteres Bild einer Doppeleihe: Antwort des ungeraden Gaborfilters.

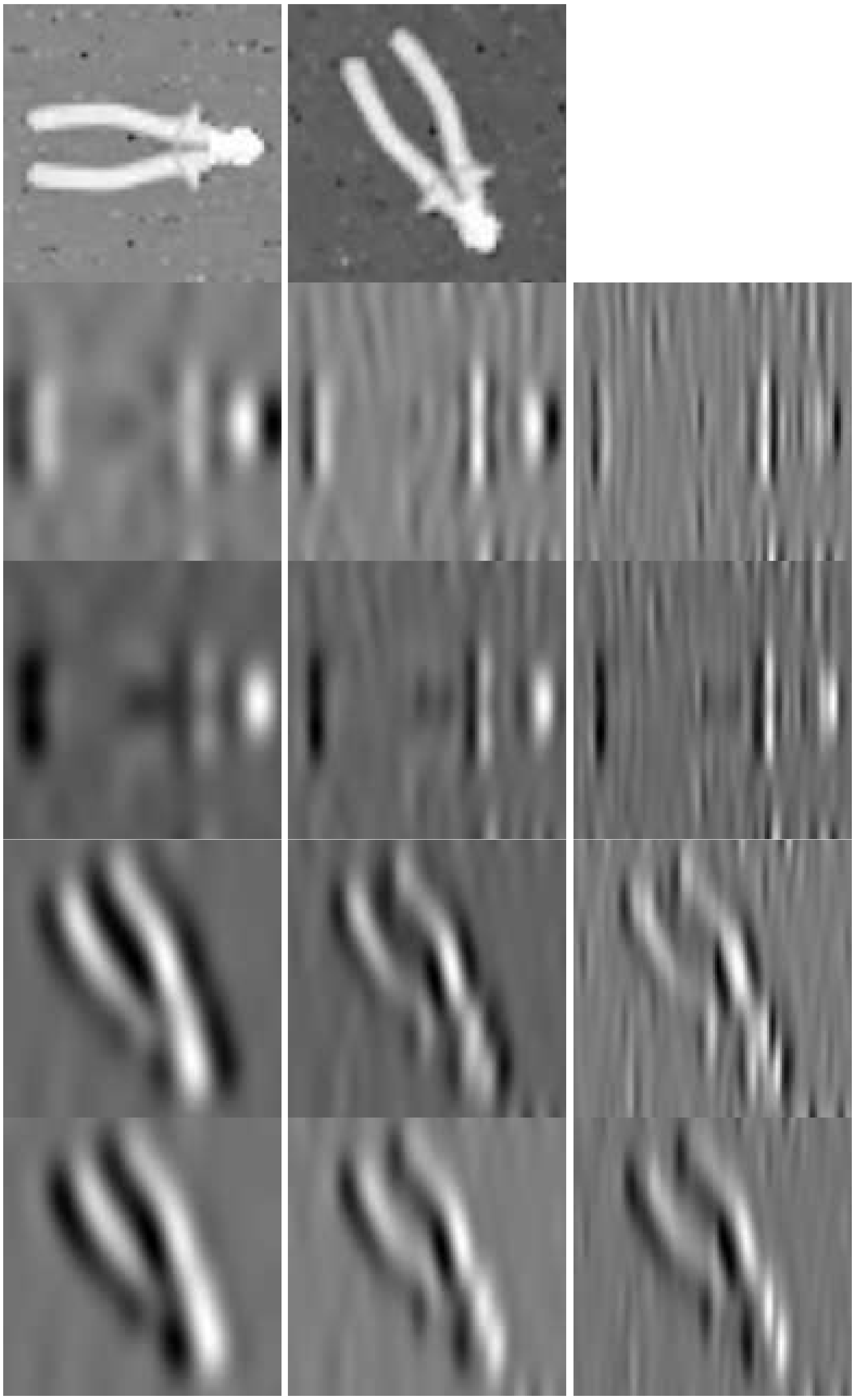

Abbildung 3.6: Faltungsergebnisse für Objekt „Kabelzange“. Obere Reihe: Das Objekt in zwei Orientierungen. Darunter in zwei Doppelreihen von links nach rechts: Die Faltungsergebnisse auf den beiden Ansichten mit einem Gaborpaar mit den Skalen 30, 15 und 7 Pixeln. Oberes Bild einer Doppelreihe: Antwort des geraden Gaborfilters. Unteres Bild einer Doppeleihe: Antwort des ungeraden Gaborfilters.

Kapitel 4

Die Greifaufgabe

4.1 Einführung

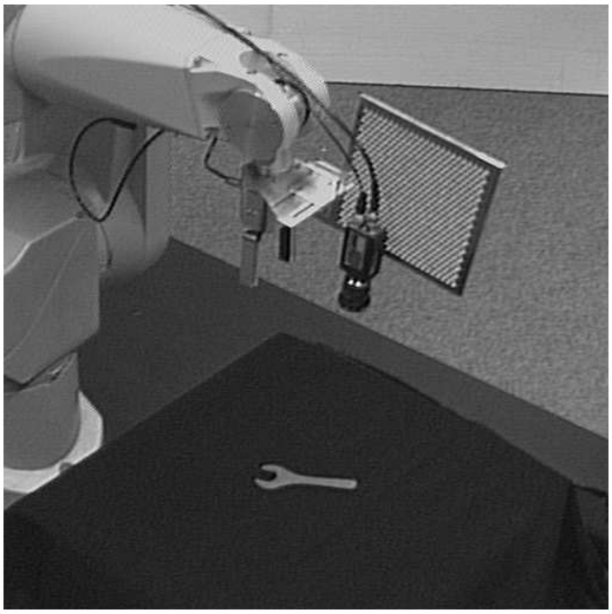

Abbildung 4.1: Roboterarm mit montierter Kamera für den visuell gesteuerten Greifvorgang.

Die beiden vorangegangenen Kapitel haben eine Basis geschaffen für die Lösung der visuell basierten Greifaufgabe. In diesem Kapitel folgt nun die Vorstellung des Gesamtsystems. Es soll das Zusammenspiel zwischen Erzeugung der Bilddaten durch die Kamera, Auswertung der Bilddaten und Steuerung der Roboterhand realisieren.
Es wird von folgender, konkreter Aufgabenstellung ausgegangen: Es sollen verschiedene, dem System bekannte Objekte gegriffen werden. Das jeweilige Objekt befindet sich auf einer Fläche in der Szene, auf die eine Kamera senkrecht gerichtet ist (siehe Abb. 4.1). Das

Objekt kann sich an einer beliebigen Position im Sichtbereich der Kamera befinden und hat in seiner Orientierung einen Freiheitsgrad, nämlich eine beliebige Rotation um eine vertikale Achse. Es stellen sich die drei typischen Hauptaufgaben: Die Objekterkennung, Posenbestimmung und Objektlokalisation. Die Objekterkennung und Posenbestimmung geschieht auf den Bilddaten, die in den Merkmalsraum projiziert werden. Der Projektionsvektor fungiert als Eingabe für ein spezielles neuronales Netzwerkkonstrukt, welches die in Kapitel 2 entwickelten SDCS-Netze mit linearer Erweiterung verwendet. Anschließend folgt die Objektlokalisation, also die Berechnung der Realdaten ebenfalls aus den Bilddaten. Hierfür wird ein geeignetes Berechnungsverfahren entwickelt.

4.1.1 Vorgehensweise

Der Ablaufplan in Abbildung 4.2 gibt einen Abriß der Vorgehensweise eines Greifvorgangs an. Man erhält von der Kamera ein Grauwertbild der Szene, welches das zu greifende Objekt beinhaltet. Das Bild ist die Eingabe für zwei Hauptprozesse. Der erste Prozeß (linke Spalte des Schaubildes) regelt die Objekterkennung und Posenbestimmung. Er besteht wiederum aus Teilprozessen. Zunächst geschieht eine Vorverarbeitung des Bildes, durch die unter anderem eine einheitliche Größe und Position des Objektes im Bild erreicht wird. Die Verfahren zur Bildvorverarbeitung sind in Kapitel 3 ausführlich behandelt worden. Aus diesen Daten wird dann der Merkmalsvektor durch Falten des Bildes an festgelegten Ortspunkten erstellt. Auch für dieses Thema sei auf das Kapitel 3 verwiesen. Der Merkmalsvektor fungiert als Eingabe für das neuronale System, ein Netzwerkkonstrukt, welches aus mehreren Teilnetzen besteht. Jedes dieser Teilnetze ist für die Bestimmung der Orientierung eines dem System bekannten Objekts zuständig, sie liefern damit die Posenbestimmung. Zunächst muß der Merkmalsvektor jedoch einem der Teilnetze zugeordnet werden. Dieses geschieht, indem der zum Merkmalsvektor naheliegendste Knoten im gesamten Netz gesucht wird. Die Bestimmung dieses Knotens und die daraus resultierende Zuordnung zu einem Teilnetz ist gleichbedeutend mit der Objekterkennung.
Der zweite Hauptprozeß ist die Lokalisation des Objektes in Weltkoordinaten. Der Prozeß benötigt als Eingabe neben dem Originalszenebild das Ergebnis der Objekterkennung und Posenbestimmung aus dem ersten Prozeß. Es wird dabei die Greifposition im Bild bestimmt und diese dann in Weltkoordinaten umgerechnet. Die Weltkoordinaten zusammen mit der Orientierungsangabe aus der Posenbestimmung sind für das Robotersystem hinreichende Angaben zum Anfahren der Greifstellung.

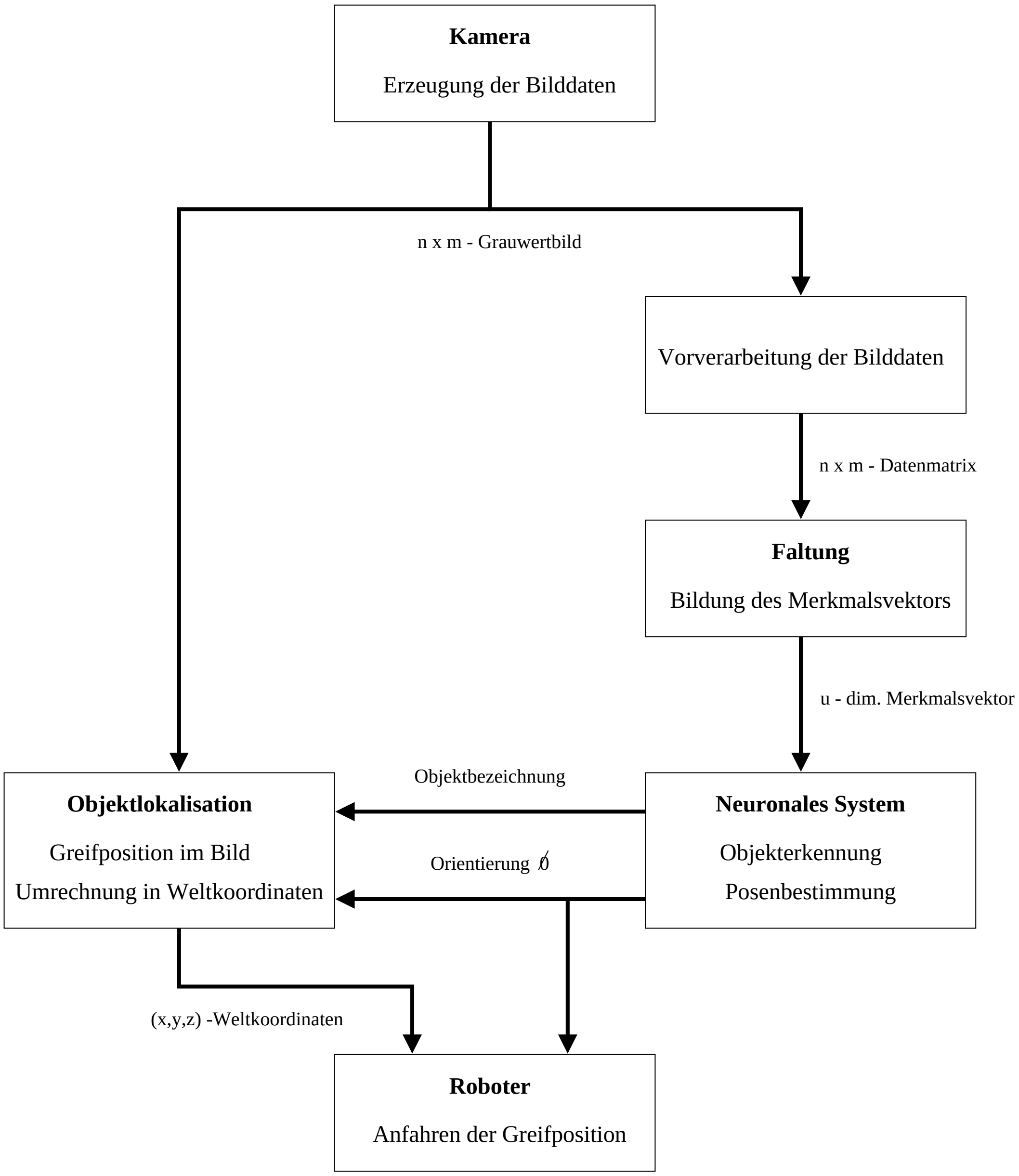

Abbildung 4.2: Ablaufplan für einen visuell geführten Greifvorgang.

4.2 Die technischen Systemkomponenten

Die drei Hauptkomponenten des verwendeten technischen Systems sind die Roboterhand, die den Greifvorgang durchführt, das Kamerasystem, welches die Bildinformationen liefert, und eine Sun-Workstation, die als Schnittstelle fungiert. Diese erhält die Bildinformationen von der Kamera, wertet sie aus und gibt dann Bewegungsbefehle an den Roboterarm bzw. die Roboterhand. Die Bildauswertung geschieht dabei unter anderem über das Bildverarbeitungssystem KHOROS, welches auf der Workstation installiert ist.

4.2.1 Das Robotersystem

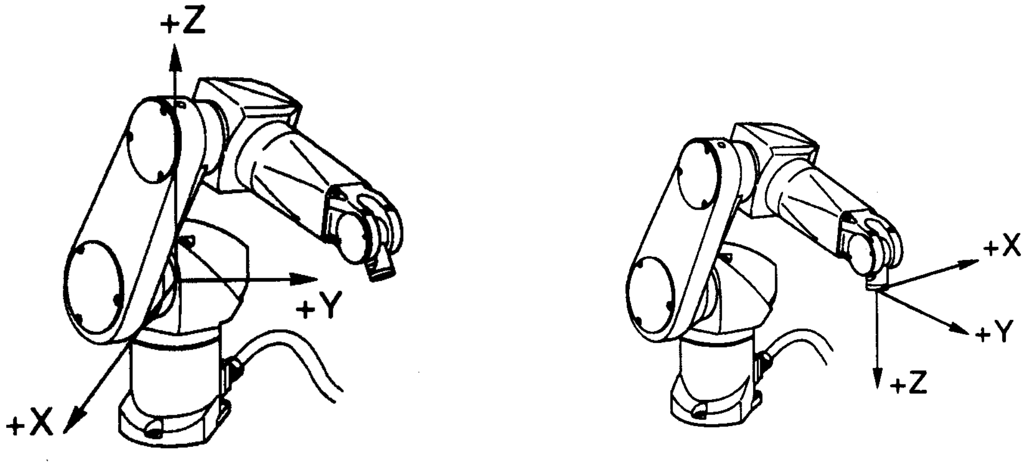

Abbildung 4.3: Die Koordinatensysteme des Roboters. Links: Das BASISsystem. Rechts: Das WERKZEUGsystem.

Für diese Arbeit wird der Roboterarm RX90 der Firma Stäubli verwendet. Der Arm besitzt 6 Drehgelenke, durch welche es ihm möglich ist, die Greifhand, oder auch Endeffektor genannt, an einem beliebigen Ort innerhalb seines Arbeitsbereichs bei nahezu beliebiger Orientierung zu positionieren. Die vorhandene Hardware löst die Probleme der inversen Kinematik und inversen Dynamik, so daß das Bewegen der Greiferhand allein durch Angabe von Position und Orientierung geregelt werden kann. Diese Positionierung definiert sich über zwei Koordinatensysteme, dem BASISsystem und dem WERKZEUGsystem, deren Lage in Abbildung 4.3 dargestellt ist. Sie wird durch einen sechsdimensionalen Vektor $(x, y, z, yaw, pitch, roll)$ definiert, wobei (x, y, z) in mm ausgedrückt wird und die Entfernung des Ursprungs des WERKZEUGsystems vom Ursprung des BASISsystems angibt. Die Winkel $(yaw, pitch, roll)$ werden in Grad angeben und bezeichnen die Orientierung des WERKZEUGsystems relativ zum BASISsystem. Die Beschreibung der Orientierung geschieht dabei in Euler-Notation. Die Werte $(yaw, pitch, roll) = (0,\ 180,\ \phi)$ entsprechen der einer Grundstellung, bei der die Finger der Greifhand senkrecht nach unten verlaufen,

ϕ beschreibt dabei eine Orientierung der Hand relativ zur vertikalen Koordinatenachse, der z-Achse des BASISkoordinatensystems. Bei der in dieser Arbeit verfolgten Greifstrategie wird davon ausgegangen, daß sich das Objekt aus dieser Grundstellung greifen läßt. Daher definiert sich eine Greifstellung über einen vierdimensionalen Vektor (x, y, z, ϕ), wobei (x, y, z), wie oben angegeben, die Position angibt und ϕ eine Orientierung um die z-Achse des BASISsystems. Das BASISsystem fungiert damit gleichzeitig als *Weltkoordinatensystem*. Wenn im weiteren Verlauf der Arbeit von Weltkoordinaten gesprochen wird, so sind Koordinaten bezüglich des BASISsystems gemeint.

4.2.2 Das Kamerasystem

Das Kamerasystem besteht aus einer starren CCD-Chip Schwarz-Weiß-Kamera. Die Kamera ist auf dem Manipulator montiert, der für die Bildaufnahme in eine vordefinierte Position gefahren wird. Die Kamera liefert ein quadratisches Grauwertbild von 512×512 Bildpunkten, welches aus Effizienzgründen auf eine Größe von 128×128 skaliert wird. Auf dem Grauwertbild ist ein Bildkoordinatensystem (X_b, Y_b) definiert, welches seinen Ursprung in der linken oberen Ecke des Bildes hat. Die Pixelwerte sind natürliche Zahlen im Bereich von 0 bis 255, wobei der Wert 255 die höchste Helligkeit symbolisiert.

4.2.3 Die Workstation

Die Workstation fungiert als Schnittstelle zwischen Kamera- und Robotersystem. Sie erhält die Bilddaten von der Kamera und gibt die Steuerbefehle an den Roboter. Die Hauptaufgabe besteht in der Auswertung der Bilddaten. Hierfür sind zum einen die neuronalen Netzwerke auf der Workstation implementiert, zum anderen das Bildverarbeitungssystem KHOROS. KHOROS stellt eine Vielzahl von Bildverarbeitungsmethoden zur Verfügung, wie Rotation, Translation, Skalierung von Bildern. Diese Verfahren werden für die Bildverabeitung des Greifsystem benötigt. Auch die spezielle Art der Faltung, die hier verwendet wird, wurde unter KHOROS implementiert.

4.3 Das neuronale System

Das *neuronale Netzwerkkonstrukt* zur Objektbestimmung und Posenbestimmung besteht aus Teilnetzen. Jedes dieser Teilnetze repräsentiert ein Objekt aus der Menge der dem System bekannten Objekte und ist ausschließlich für dessen Posenbestimmung zuständig. Es werden dafür SDCS-Netze mit linearer Erweiterung verwendet. Die topologische Karte eines solchen Netzes approximiert die Mannigfaltigkeit, auf die alle Ansichten des jeweiligen Objektes durch Projektion in den Merkmalsraum abgebildet werden. Die Menge der Teilnetze bildet also eine Art Objektdatenbank. Es werden die Teilnetze zwar zu einem einzigen Netzkonstrukt zusammengefügt, dabei bleiben sie jedoch untereinander unabhängig, das heißt, es existieren keine Verbindungen zwischen Neuronen verschiedener Teilnetze. Die Zusammenfügung zu einem Gesamtkonstrukt dient allein der Objekterkennung, also der Zuordnung eines Merkmalsvektors zu einem Teilnetz.

4.3.1 Objekterkennung und Posenbestimmung durch das neuronale Netzkonstrukt

Man betrachte das Netzkonstrukt als einen Graphen $G = (V, E)$. V ist die Menge der Neuronen, wobei $V = \bigcup_{i=1}^{k} V_i$ die Vereinigung der Neuronenmengen der k Teilnetze ist. E ist die Menge der Kanten in G, wobei gilt:

$$\forall (i, j) \in E, i \in V_m, j \in V_n \Rightarrow m = n.$$

Es existieren also nur Kanten zwischen Neuronen, die zum selben Teilnetz gehören. Damit läßt sich nun die Objekterkennung und Posenbestimmung für eine Ansicht, die in den Merkmalsraum projiziert wurde, durchführen:

1. Objekterkennung: Man bestimme für den Merkmalsvektor das naheliegendste Neuron bmu aus V. Das Objekt der Objektdatenbank, welches durch das Teilnetz $G_{bmu} = (V_{bmu}, E_{bmu})$, also das Netz, welches das Neuron bmu beinhaltet, repräsentiert wird, ist das erkannte Objekt.

2. Posenbestimmung: Das Teilnetz G_{bmu} berechnet die Pose des Objektes nach dem bekannten Verfahren (siehe Kapitel 3).

4.3.2 Training der Netze

Das neuronale Netzwerkkonstrukt muß zunächst trainiert werden. Das Training besteht aus zwei Teilen. Zunächst wird jedes Teilnetz einzeln trainiert, so daß es die richtige Konfiguration für die Posenbestimmung des Objektes besitzt, welches es repräsentieren soll. Anschließend folgt ein Training des Gesamtnetzes, durch welches die fehlerfreie Zuordnung von Merkmalsvektoren zu den Teilnetzen gewährleistet werden soll.

Einzeltraining der Teilnetze

Für jedes zu lernende Objekt existiert eine Menge von Ansichten des Objektes in verschiedenen Orientierungen. Zu jeder dieser Ansichten ist eine Orientierung ϕ zwischen 0 und 2π definiert. Man erhält so eine Trainingsmenge für ein neuronales Netz, welches das jeweilige Objekt repräsentieren soll. Der Projektionsvektor einer Ansicht im Merkmalsraum ist der Eingabevektor für ein Trainingsdatum, die zugehörige Orientierung ϕ die Ausgabe. Da die

Ansichten eines Objektes einen Zyklus bilden, so daß die Ansichten für die Orientierungen 0 und 2π identisch sind, ist es sinnvoll, daß der Wertebereich der Ausgabe auch einen zyklischen Verlauf hat. Man wähle daher für die Ausgabe des Netzes anstelle der Orientierung ϕ einen zweidimensionalen Ausgabevektor

$$\boldsymbol{o} = \begin{pmatrix} o_1 \\ o_2 \end{pmatrix} = \begin{pmatrix} \sin\phi \\ \cos\phi \end{pmatrix} . \tag{4.1}$$

Die Orientierung läßt sich dann wieder leicht aus einem Ausgabevektor $\boldsymbol{o} = (o_1, o_2)$ zurückrechnen durch

$$\phi = \begin{cases} \arccos o_2 & o_1 \geq 0 \\ 2\pi - \arccos o_2 & o_1 < 0 \end{cases} . \tag{4.2}$$

Das Abbruchkriterium für das Training kann über den durchschnittlichen Approximationsfehler definiert werden: Es wird ein SDCS-Netz mit der jeweiligen Trainingsmenge für ein Objekt trainiert, indem solange sukzessiv weitere Neuronen in das Netz einfügt werden und es anschließend konfiguriert wird, bis die Approximationsgüte auf den Trainingsdaten eine festgelegte Fehlertoleranz unterschreitet und man so die konfigurierten Teilnetze erhält. Alternativ dazu kann eine feste Zahl von Neuronen, die sukzessive eingefügt werden sollen, für alle Teilnetze vorgegeben werden, um so ein ausgewogenes System zu erhalten. Dabei sollte jedoch darauf geachtet werden, daß sich alle Teilnetze in einer gewissen Fehlertoleranz bewegen.

Training des Gesamtnetzes

Man ist bei der Extraktion der Merkmalsvektoren darauf bedacht, daß die Mannigfaltigkeiten, auf die die Ansichten von verschiedenen Objekten im Merkmalsraum projiziert werden, weit voneinander entfernt liegen, so daß es zu keiner Fehlklassifizierung bei der Objekterkennung kommt. Im allgemeinen ist es aber nicht möglich, dies allein durch die Bildverarbeitung zu erreichen. Gerade Objekte mit ähnlichem Erscheinungsbild, wie z.B. zwei verschiedenartige Zangen, werden in bestimmten Orientierungen ähnliche Merkmalsvektoren erzeugen. Die Konfiguration des Netzwerkkonstrukts muß letztendlich die fehlerfreie Klassifizierung sicherstellen. Die Objekterkennung durch das neuronale Netzwerkkonstrukt geschieht über die Suche des zum Merkmalsvektor naheliegendsten Neurons. Es ist sinnvoll, in den kritischen Bereichen, in denen es zu Fehlklassifizierungen kommen kann, zusätzliche Neuronen in die betroffenen Teilnetze einzufügen. Es werden daher die Trainingsmengen aller Teilnetze zu einer Gesamttrainingsmenge zusammengefaßt und die Klassifizierung aller Trainingsdaten getestet. Wird ein Trainingsdatum einem falschen Teilnetz zugeordnet, so wird an die Position des zugehörigen Eingabevektors ein neues Neuron in das richtige Teilnetz eingefügt und das Netz neu konfiguriert. Das Vorgehen wird im folgenden Algorithmus konkretisiert:

Algorithmus (Training des Gesamtnetzes)

Vorbedingung: Es sind $G_i = (V_i, E_i)$, $i = 1, \ldots, k$ die konfigurierten Teilnetze, $T = \bigcup_{i=1}^{k} T_i$ die Gesamttrainingsmenge.

repeat
  for all $t \in T$
    bestimme *bmu* von t;
    bestimme V_j mit $bmu \in V_j$;
    bestimme T_k mit $t \in T_k$;
    if $j \neq k$ /* falsche Klassifizierung */
      Füge in G_k ein neues Neuron ein mit dem Eingabevektor
      von t als Zentrum des neuen Knotens;
      Konfiguriere G_k neu;
      break; /* Abbruch der for-Schleife */
    end if;
  end for;
until keine weiteren falschen Klassifizierungen;

Der Algorithmus liefert ein konfiguriertes Netzkonstrukt, das auf den Trainingsdaten eine fehlerfreie Objekterkennung garantiert.

Generierung der Ansichten

Die Trainingsphase der einzelnen Teilnetze benötigt eine Menge von Ansichten des jeweiligen Objektes. Eine Möglichkeit zur Erzeugung dieser Ansichten ist, das Objekt in mehreren Orientierungen aufzunehmen. Das Objekt müßte dafür jedoch in der Realszene zum Beispiel durch den Manipulator schrittweise um die z-Achse rotiert werden. In dieser Arbeit wird dagegen ein Verfahren verwendet, bei dem eine Ansichtenmenge künstlich erzeugt wird. Grundlage des Verfahrens ist eine einzige durch die Kamera erzeugte Ansicht für ein Objekt. Hierfür wird die sogenannte Grundansicht verwendet, bei der sich das Objekt in einer Lage in der Realwelt befindet, in der es vom Manipulator in einer Grundgreifstellung gegriffen werden kann. Die Begriffe werden im Verlauf der Arbeit genauer erläutert.
Die Bilddaten der Ansichten für die Lernphase werden wie auch die Ansichten von Realszenen vorverarbeitet, das heißt, es erfolgt eine Translation des Pixelschwerpunktes des Objektes in den Bildmittelpunkt und eine Skalierung. Da es sich bei den zu lernenden Objekten um flache Objekte handelt und die Kamera senkrecht auf die Szene gerichtet ist, kann die reale Rotierung des Objektes um eine vertikale Achse durch eine Rotation der Bilddaten simuliert werden. Man nimmt als Ausgangsbild die vorverarbeitete Grundansicht eines Objektes, rotiert diese schrittweise und erhält so eine künstlich erzeugte Menge von Ansichten für die Lernphase. Die Orientierung, die jeder Ansicht zugeordnet wird,

ist der Grad der Rotation der Grundansicht. Damit haben die Orientierungsangaben der Objekte in den Ansichten einen relativen Bezug: Die Orientierung in der Grundansicht.

4.4 Objektlokalisation

Der zweite Hauptprozeß zur Realisierung des Greifvorgangs ist die Objektlokalisation in der Realszene. Bisher wurde nur die Orientierung des Objektes bestimmt. Für den Greifvorgang ist aber auch die Berechnung einer Position in Weltkoordinaten nötig, die der Manipulator anfahren muß, um den Greifvorgang durchzuführen. Die folgenden Abschnitte stellen ein geeignetes Verfahren vor.

4.4.1 Grundansicht und Grundgreifstellung

Das Verfahren zur Objektlokalisation in dieser Arbeit verwendet keine Projektionsverfahren, die die Abbildung von absoluten Bildkoordinaten in absolute Weltkoordinaten angeben, sondern es werden nur positionsunabhängige Differenzvektoren im Bild und deren korrespondierende Differenzvektoren in der Welt betrachtet. Dennoch ist mindestens ein absoluter Bezug zwischen Bild- und Weltkoordinaten nötig, auf den sich die relativen Differenzvektoren beziehen. Dieser Bezug wird über die schon erwähnte *Grundgreifstellung* des Manipulators realisiert. Diese besteht aus einer *Grundgreifposition* $(x_c^{(w)}, y_c^{(w)}, z_c^{(w)})^T$ und einer *Grundgreiforientierung* ϕ_c, welche eine feste Stellung des Manipulators in der Szene definieren. Diese Grundgreifstellung ist identisch für jedes Objekt und definiert für alle Objekte eine individuelle Ausgangslage, eine Position und Orientierung des jeweiligen Objektes in der Szene, in der es aus der Grundgreifstellung des Manipulators heraus gegriffen werden kann. Die Ansicht eines Objektes in der Ausgangslage ist die sogenannte *Grundansicht* des Objektes. Damit ist ein absoluter Bezug zwischen Bild- und Weltdaten gegeben.
Eine Abweichung der Lage eines Objektes in einer Szenenansicht von der Position in der entsprechenden Grundansicht bedeutet für den Greifvorgang eine Translation des Manipulators aus der Grundgreifstellung. Unter Objektlokalisation wird hier die Berechnung dieser Translation verstanden. Die Berechnung geschieht in zwei Schritten:

1. Finden des Greifpunktes in der Szenenansicht und Berechnung der Translation zur Position in der Grundansicht.

2. Berechnung der Translation in den Weltkoordinaten aus der Translation in den Bilddaten.

Der erste Schritt benötigt das Ergebnis der Objekterkennung, denn jedes Objekt besitzt eine eigene Greifposition, und der Posenbestimmung, denn auch die Pose hat einen Einfluß auf die Translationsvektoren. Der zweite Schritt benötigt eine Berechnungsgrundlage, die die Beziehung zwischen Translationen im Bild und in der Welt beschreibt. So eine Berechnungsgrundlage kann über eine Projektionsmatrix definiert werden.

4.4.2 Greifpunktbestimmung im Bild

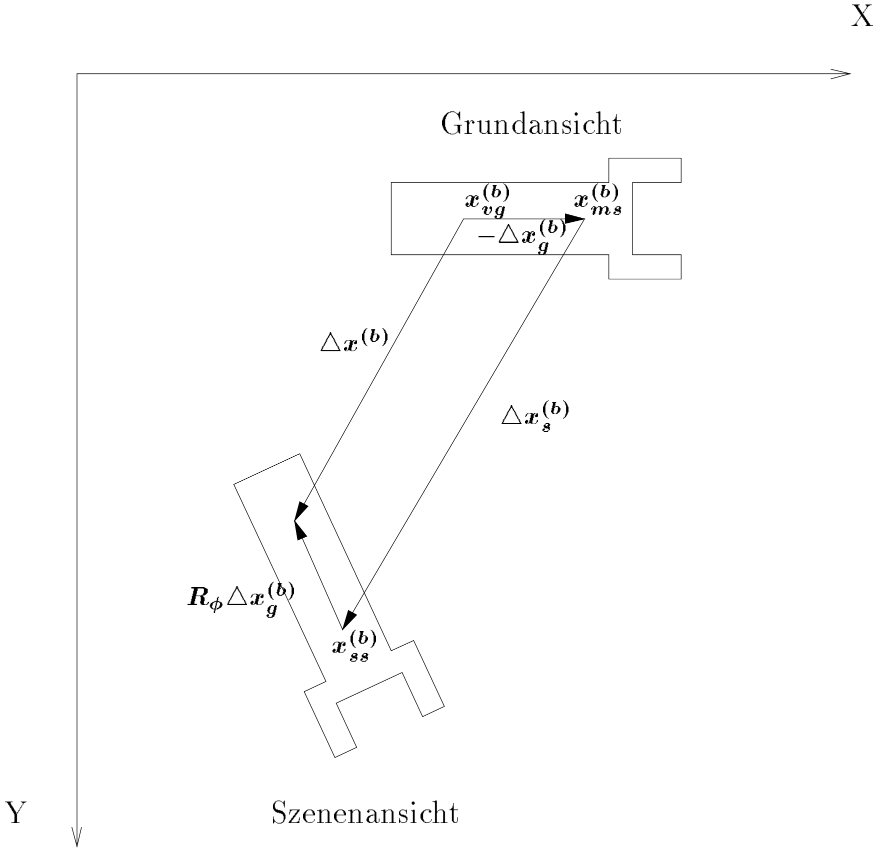

Abbildung 4.4: Skizze zur Berechnung der Translation $\triangle \boldsymbol{x}^{(b)}$ im Bild.

Auch die Greifpunktbestimmung teilt sich in zwei Phasen auf, in eine Modell- und eine Anwendungsphase. In der Modellphase geht man von der Grundansicht eines Objektes aus. Man bestimmt daraus den *Modellpixelschwerpunkt* $\boldsymbol{x}_{ms}^{(b)}$ und speichert den Vektor in der Objektdatenbank. Zusätzlich wird zum Objekt der *Modellgreifpunktdifferenzvektor* $\triangle \boldsymbol{x}_g^{(b)}$ bezüglich der Grundansicht bestimmt. Dieser gibt die Translation zwischen Modellpixelschwerpunkt des jeweiligen Objekts und der *virtuellen Greifposition*[1] $\boldsymbol{x}_{vg}^{(b)}$ an, an der sich der Manipulator im Bild befinden würde, wenn er sich in der Welt in seiner Grundgreifposition befindet. Die virtuelle Greifposition im Bild wird a priori vorgegeben. Damit berechnet sich der Modellgreifpunktdifferenzvektor des Objektes mit $\triangle \boldsymbol{x}_g^{(b)} = \boldsymbol{x}_{vg}^{(b)} - \boldsymbol{x}_{ms}^{(b)}$.

[1]Die Position wird als virtuell bezeichnet, weil der Manipulator in Wirklichkeit nie im Bild erscheinen kann, da die Kamera am Manipulator befestigt ist.

Soll nun in der Anwendungsphase zu einer Szenenansicht die Translation bezüglich der Grundansicht berechnet werden, so erhält man zunächst als Folge der Objekterkennung den Modellschwerpunkt $\boldsymbol{x}_{\boldsymbol{ms}}^{(\boldsymbol{b})}$, den Modellgreifpunktdifferenzvektor $\triangle\boldsymbol{x}_{\boldsymbol{g}}^{(\boldsymbol{b})}$ und das Ergebnis ϕ der Posenbestimmung. Man bestimmt anschließend den Pixelschwerpunkt $\boldsymbol{x}_{\boldsymbol{ss}}^{(\boldsymbol{b})}$ in der Szene und den *Schwerpunktdifferenzvektor* $\triangle\boldsymbol{x}_{\boldsymbol{s}}^{(\boldsymbol{b})} = \boldsymbol{x}_{\boldsymbol{ss}}^{(\boldsymbol{b})} - \boldsymbol{x}_{\boldsymbol{ms}}^{(\boldsymbol{b})}$. Die Translation $\triangle\boldsymbol{x}^{(\boldsymbol{b})}$ des Greifpunkts in der Szene relativ zum Greifpunkt in der Grundansicht im Bild berechnet sich dann durch

$$\triangle\boldsymbol{x}^{(\boldsymbol{b})} = -\boldsymbol{x}_{\boldsymbol{g}}^{(\boldsymbol{b})} + \triangle\boldsymbol{x}_{\boldsymbol{s}}^{(\boldsymbol{b})} + \boldsymbol{R}_{\boldsymbol{\phi}}\boldsymbol{x}_{\boldsymbol{g}}^{(\boldsymbol{b})}, \tag{4.3}$$

mit

$$\boldsymbol{R}_{\boldsymbol{\phi}} = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & cos\phi \end{pmatrix}.$$

Die Geometrie, die dieser Berechnung zugrunde liegt, ist in Abbildung 4.4 skizziert.

4.4.3 Umrechnung in Realdaten

Die Berechnung von Bildkoordinaten aus Weltkoordinaten und vice versa ist ein klassisches Problem der Bildverarbeitung. Die Abbildung eines Realpunktes in einem Bild hängt zum einem von den intrinsischen Parametern wie Brennweite und Pixelskalierung der Kamera und zum anderen von den extrinsischen Parametern wie Position und Orientierung der Kamera in der Welt ab. Die Abbildung kann durch eine sogenannte *Projektionsmatrix* repräsentiert werden, in der implizit die extrinsischen und intrinsischen Parameter enthalten sind. Im Fall von homogenen, dreidimensionalen Weltkoordinaten und homogenen, zweidimensionalen Bildkoordinaten ist die klassische Projektionsmatrix eine 3×4 Matrix.
In dem hier vorgestellten Szenario befinden sich die Objekte auf einer Ebene orthogonal zur z-Achse. Da es sich um flache Objekte handelt, kann die z-Koordinate der Weltkoordinaten als konstant angenommen und a priori vorgegeben werden. Weiterhin wird nicht von homogenen Koordinatenvektoren ausgegangen, sondern man betrachtet einfache Differenzvektoren. Man benötigt damit eine Abbildungsfunktion

$$F : \mathbb{R}^2 \longrightarrow \mathbb{R}^2, \quad \triangle\boldsymbol{x}^{(\boldsymbol{b})} \mapsto \triangle\boldsymbol{x}^{(\boldsymbol{w})}. \tag{4.4}$$

Die Abbildung läßt sich durch eine 2×2 - Matrix $\boldsymbol{P}$ repräsentieren. Man erhält dann die Translation $\triangle\boldsymbol{x}^{(\boldsymbol{w})}$ in den Weltkoordinaten aus der Translation in den Bilddaten durch

$$\begin{pmatrix} \triangle x^{(w)} \\ \triangle y^{(w)} \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix} \begin{pmatrix} \triangle x^{(b)} \\ \triangle y^{(b)} \end{pmatrix}. \tag{4.5}$$

Die Koeffizienten p_{ij} der Matrix $\boldsymbol{P}$ werden in der sogenannten Kalibrierungsphase bestimmt.

4.4.4 Kalibrierung

Die Vorgang der *Kalibrierung* ist die Bestimmung der extrinsischen und intrinsischen Parameter der Kamera. Da die Parameter implizit in den Koeffizienten der Projektionsmatrix enthalten sind, soll hier unter Kalibrierung die Bestimmung der Koeffizienten p_{ij} der Projektionsmatrix P verstanden werden. Die Kalibrierung geschieht über Stützstellen, das sind Paare $(\triangle \boldsymbol{x}_i^{(b)}, \triangle \boldsymbol{x}_i^{(w)})$, bei denen für eine Translation $\triangle \boldsymbol{x}_i^{(b)}$ im Bild die zugehörige Translation in der Welt bekannt ist. Existieren n dieser Stützstellen, so erhält man nach (4.5) n Gleichungssysteme

$$\begin{pmatrix} \triangle x_i^{(w)} \\ \triangle y_i^{(w)} \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix} \begin{pmatrix} \triangle x_i^{(b)} \\ \triangle y_i^{(b)} \end{pmatrix}.$$

Formt man die Gleichungen um und faßt sie zu einem Gleichungssytem zusammen, so erhält man folgendes, in der Regel überbestimmtes Gleichungssystem, über das die Koeffizienten der Projektionsmatrix bestimmt werden können:

$$\begin{pmatrix} x_1^{(b)} & y_1^{(b)} & 0 & 0 \\ 0 & 0 & x_1^{(b)} & y_1^{(b)} \\ x_2^{(b)} & y_2^{(b)} & 0 & 0 \\ 0 & 0 & x_2^{(b)} & y_2^{(b)} \\ \vdots & \vdots & \vdots & \vdots \\ x_n^{(b)} & y_n^{(b)} & 0 & 0 \\ 0 & 0 & x_n^{(b)} & y_n^{(b)} \end{pmatrix} \begin{pmatrix} p_{11} \\ p_{12} \\ p_{21} \\ p_{22} \end{pmatrix} = \begin{pmatrix} x_1^{(w)} \\ y_1^{(w)} \\ x_2^{(w)} \\ y_2^{(w)} \\ \vdots \\ x_n^{(w)} \\ y_n^{(w)} \end{pmatrix}. \tag{4.6}$$

Generierung der Stützstellen

Es werden zur Generierung der Stützstellen Weltkoordinaten benötigt, deren Abbildung in den Bilddaten bestimmt werden kann. Jeweils zwei verschiedene Weltkoordinaten und deren korrespondierende Punkte im Bild ergeben einen Translationsvektor in der Welt und einen im Bild und damit eine Stützstelle für die Kalibrierung.
Zu diesem Zweck wird mit Hilfe des Manipulators ein flaches Objekt an beliebigen Positionen bei konstanter Orientierung in der Szene positioniert. Die jeweilige Position des Manipulators ergibt eine Koordinate in der Welt. Die korrespondierende Koordinate im Bild erhält man, indem man den Pixelschwerpunkt des Objektes in der Ansicht bestimmt.

4.5 Anfahren der Greifstellung

Basis für die Berechnung der Koordinaten der einzunehmenden Greifstellung ist die Grundgreifstellung des Manipulators, bestehend aus Grundgreifposition $(x_c^{(w)}, y_c^{(w)}, z_c^{(w)})^T$ und

Grundgreiforientierung ϕ_c. Da die z-Koordinate für alle Greifstellungen identisch sein soll, wird der aus den Bilddaten berechnete Translationsvektor $\triangle\boldsymbol{x}^{(\boldsymbol{w})}$ zu einem dreidimensionalen Translationsvektor $\triangle\boldsymbol{x}^{(\boldsymbol{w})} = (\triangle x^{(w)}, \triangle y^{(w)}, 0)$ erweitert. Die anzufahrende Greifposition $\boldsymbol{x}_{\boldsymbol{g}}^{(\boldsymbol{w})}$ ergibt sich dann als

$$\boldsymbol{x}_{\boldsymbol{g}}^{(\boldsymbol{w})} = \begin{pmatrix} x_g^{(w)} \\ y_g^{(w)} \\ z_g^{(w)} \end{pmatrix} = \begin{pmatrix} x_c^{(w)} \\ y_c^{(w)} \\ z_c^{(w)} \end{pmatrix} + \begin{pmatrix} \triangle x^{(w)} \\ \triangle y^{(w)} \\ 0 \end{pmatrix}. \tag{4.7}$$

Die durch die Posenbestimmung berechnete Orientierung ϕ des Objektes ist eine relative Orientierung bezüglich der Grundansicht, daher kann ϕ auch als $\triangle\phi$ aufgefaßt werden, so daß sich die Greiforientierung ϕ_g des Manipulators durch

$$\phi_g = \phi_c + \triangle\phi \tag{4.8}$$

berechnet.
Insgesamt erhält man damit den vierdimensionalen Vektor $(x_g^{(w)}, y_g^{(w)}, z_g^{(w)}, \phi_g)$, der eine eindeutige Greifstellung des Manipulators im Arbeitsbereich definiert. Das Robotersystem kann damit den Greifvorgang durchführen.

Kapitel 5

Versuche

In diesem Kapitel werden nun die experimentellen Ergebnisse des Greifsystems vorstellt. Es beginnt mit einer genauen Beschreibung des Szenarios. Danach folgt ein Abschnitt mit Versuchsreihen, in dem zunächst die Approximationsgüte der vorgestellten neuronalen Netze zur Posenbestimmung betrachtet wird. In dem darauffolgenden Abschnitt folgt dann der Test des Gesamtsystems.

5.1 Beschreibung des Szenarios

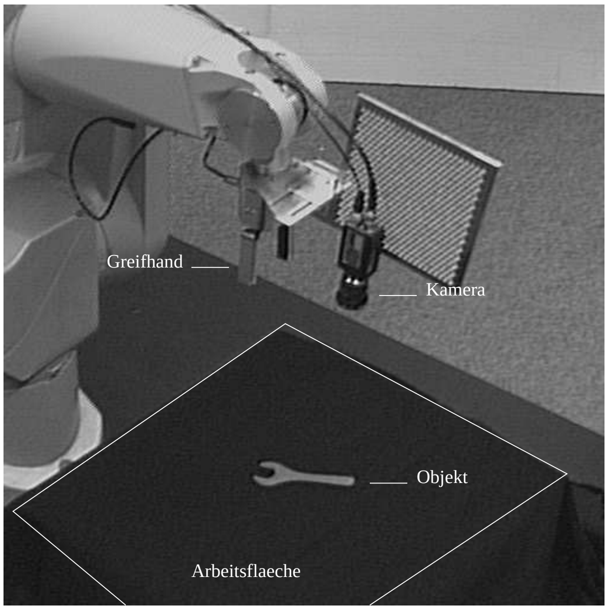

Abbildung 5.1: Roboterarm mit montierter Kamera, Arbeitsfläche und Objekt.

Das Greifszenario ist noch einmal in Abbildung 5.1 dargestellt. Das zu greifende Objekt befindet sich auf einer quadratischen Arbeitsfläche von 50×50 cm im Arbeitsbereich des Robotersystems. Die Kamera befindet sich, am Manipulator montiert, c.a. 50 cm oberhalb der Arbeitsfläche, auf die sie senkrecht gerichtet ist. Die optische Achse verläuft also senkrecht zur Arbeitsfläche. Die Kamera besitzt ein Weitwinkelobjektiv mit einer Brennweite von 4,2 mm, so daß die gesamte Arbeitsfläche im Bild ist. Das Kamerabild hat eine Größe von 128×128 Pixel. Der Manipulator befindet sich mit der montierten Kamera für die Bildaufnahme an einer festgelegten Position und fährt anschließend die berechnete Greifstellung an. Das Szenario befindet sich in einem durch Deckenlicht gleichmäßig beleuchteten Raum, so daß keine störenden Schatten entstehen. Es wird in den Versuchen die Helligkeit variiert, um die Robustheit des Systems gegenüber Helligkeitsschwankungen zu demonstrieren.

5.2 Versuchsreihen zur Posenbestimmung

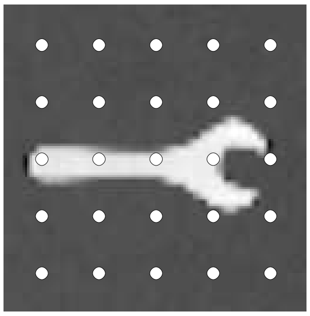

Abbildung 5.2: Vorverarbeitete Grundansicht des Objektes „Schraubenschlüssel“ mit den rasterförmig verteilten Ortspunkten der Faltung.

In einem vorangegangenen Abschnitt wurde die Approximationsgüte eines SDCS-Netzes mit linearer Erweiterung an künstlich erzeugten Daten gezeigt. Es soll nun an Realdaten aus unserem Greifszenario getestet werden, inwieweit es dort zur Posenbestimmung geeignet ist. Als Testobjekt wird zunächst der schon bekannte „Schraubenschlüssel“ verwendet. Der „Schraubenschlüssel“ wird aus der Grundstellung mittels des Manipulators in Ein-Grad-Schritten um die Objektachse gedreht und ein Bild in jeder Orientierung erzeugt. Man erhält somit 360 Ansichten. Es wird dabei nicht die Position des Objektes verändert. Die Ansichten werden in eine Trainingsmenge (Ansichten mit gerader Gradzahl) und eine Testmenge (Ansichten mit ungerader Gradzahl) aufgeteilt. Zur Extraktion der Merkmalsvektoren wird jede Ansicht an 25 Ortspunkten mit Gaboroperatoren gefaltet, wobei die Ortspunkte über ein rasterförmiges 5×5 Gitter festgelegt sind. Die Faltungsmasken haben eine Größe von 63×63 Pixeln. Es werden Gaboroperatoren in drei verschiedenen Skalen von 34, 17 und 8 Pixeln und in drei verschiedenen Orientierungen von 0, 60 und 120 Grad verwendet. Man erhält also pro Ortspunkt 9 Faltungsantworten, somit hat der

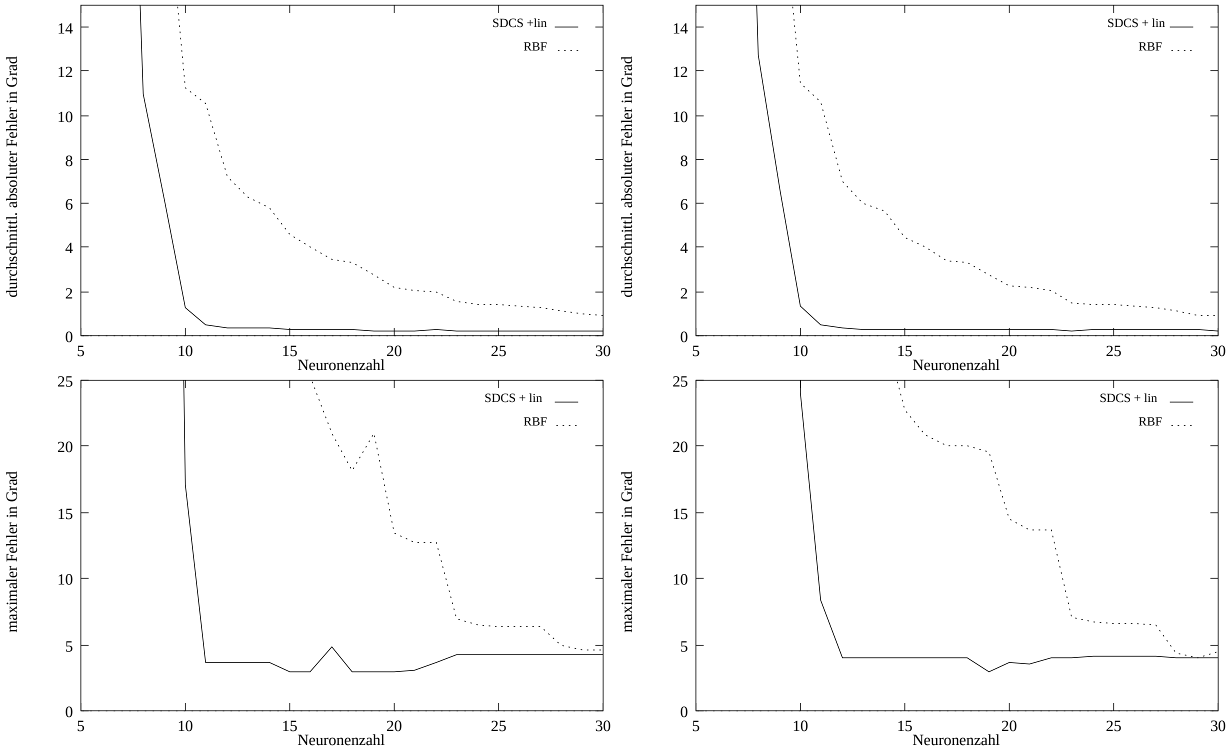

Abbildung 5.3: Fehlerkurven in Grad eines SDCS-Netzes mit linearer Erweiterung und eines RBF-Netzes mit optimal angepaßten Varianzen. Obere Reihe: Durchschnittlicher, absoluter Fehler links auf den Trainingsdaten, rechts auf den Testdaten. Untere Reihe: Maximaler, absoluter Fehler links auf den Trainingsdaten, rechts auf den Testdaten.

gesamte Merkmalsvektor 225 Dimensionen. Die jeweilige Orientierungsangabe einer Ansicht wird nach (4.1) in einem zweidimensionalen Vektor verschlüsselt, um dem zyklischen Verlauf der Ansichten zu entsprechen. Es wird ein SDCS-Netz mit lineare Erweiterung mit den Trainingsdaten trainiert und zusätzlich als Vergleich ein RBF-Netz mit optimal angepaßten Varianzen.

Die Fehlerkurven in Abbildung (5.3) zeigen den durchschnittlichen, absoluten Fehler bzw. den maximalen, absoluten Fehler in Grad auf den Trainings- und Testdaten. Das SDCS-Netz liefert schon mit wenigen Neuronen ein sehr gutes Ergebnis sowohl auf den Trainings- als auch auf den Testdaten, während das RBF-Netz wesentlich mehr Neuronen für ein ähnlich gutes Ergebnis benötigt. Interessant ist die Betrachtung des maximalen, absoluten Fehlers, also die schlechteste Approximation für ein Datum. Dieser pendelt sich beim SDCS-Netz auf den Testdaten sehr früh bei 4 Grad ein, ein für den späteren Greifvorgang akzeptables Ergebnis.

Man beachte dabei, daß das SDCS-Netz nicht beliebig mit Neuronen erweitert werden kann, da ansonsten das lineare Gleichungssystem, mit welchem die Ausgabevektoren und Jacobi-Matrizen der Neuronen bestimmt werden, im Verhältnis zu der Zahl von Trainingsdaten (in diesem Fall 180) zu viele Freiheitsgrade erhält und allmählich zu einem unterbestimmten Gleichungssystem wird. Das Result wäre der schon erwähnte Effekt des „Auswendigler-

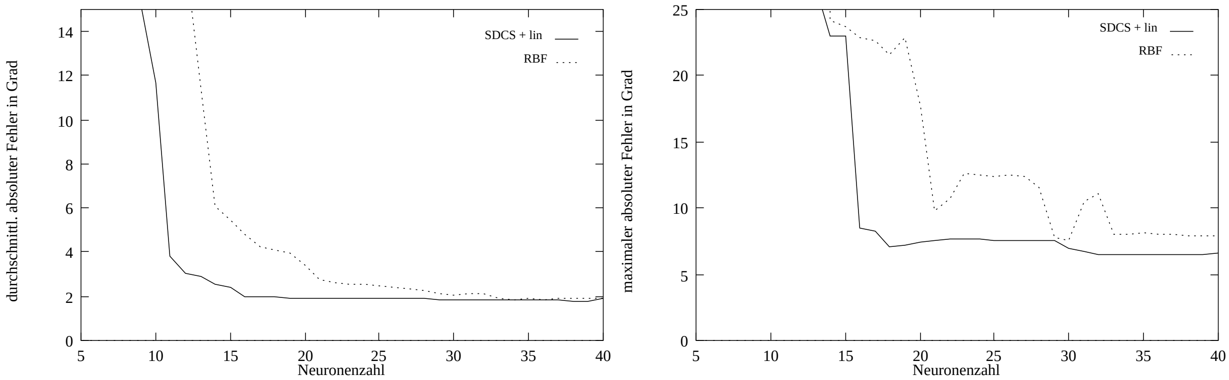

Abbildung 5.4: Training mit generierten Daten: Fehlerkurven für das Objekt „Schraubenschlüssel" auf den Testdaten in Grad eines SDCS-Netzes mit linearer Erweiterung und eines RBF-Netzes mit optimal angepaßten Varianzen. Links: Durchschnittlicher, absoluter Fehler. Rechts: Maximaler, absoluter Fehler.

nens" der Trainingsdaten, also ein nahezu perfektes Ergebnis bei der Approximation der Trainingsdaten, jedoch ein schlechtes Ergebnis auf den Testdaten.

Im vorangegangenen Versuch sind die Trainingsdaten über Szenenbeispiele erzeugt worden, indem das Objekt mittels Roboterhand in verschiedene Orientierungen rotiert wurde. Es wurde jedoch in (4.3.2) ein Verfahren vorgestellt, bei dem aus der Grundansicht mittels Rotation auf den Bilddaten eine Ansichtenmenge generiert wurde. Dieses Verfahren soll nun getestet werden. Es werden aus der Grundansicht des „Schraubenschlüssels" in Ein-Grad-Schritten 359 weitere Ansichten als Trainingsmenge erzeugt, man erhält also eine doppelt so große Trainingsmenge. Als Testmenge fungiert dieselbe Testmenge wie im ersten Versuch. Die Aufnahme der Grundansicht geschieht unabhängig von der Aufnahme der Testdaten. Es wird ein SDCS-Netz mit linearer Erweiterung trainiert und als Vergleich ein RBF-Netz mit adaptierten Varianzen. Abbildung 5.4 läßt erkennen, daß man etwas schlechtere Ergebnisse erhält als in dem ersten Versuch. Dies liegt daran, daß eine Ansicht, die durch Bildrotierung künstlich enstanden ist, die wirkliche Ansicht nie exakt wiedergeben kann. Der durchschnittliche Fehler beim SDCS-Netz mit linearer Erweiterung auf den Testdaten pendelt sich bei 2 Grad ein, der maximale Fehler bei 6 Grad. Diese Werte befinden sich für den späteren Greifvorgang jedoch im Toleranzbereich. Das Verfahren wird noch an einem zweiten Beispielobjekt getestet, dem Objekt „Zange". Auch hier kommt man zu ähnlichen Ergebnissen (siehe Abbildung 5.6). Der durchschnittliche Fehler liegt bei ca. 2 Grad, der maximale bei ca 4,5 Grad. Bei beiden Beispielen erweist sich eine Anzahl von ca. 30 Knoten im SDCS-Netz als vollkommen ausreichend.
Im Vergleich dazu liefert das RBF-Netz erst bei einer hohen Anzahl von Neuronen im Netz ähnliche Ergebnisse wie das SCDS-Netz. Dabei weist das RBF-Netz durchgängig einen höheren, maximalen Fehler auf. Beim Objekt „Zange" ist die maximale, absolute Abweichung mit ca. 10 Grad doppelt so groß.

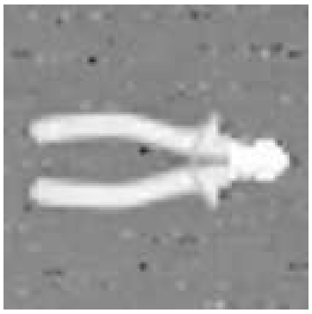

Abbildung 5.5: Vorverarbeitete Grundansicht des Objektes „Zange“.

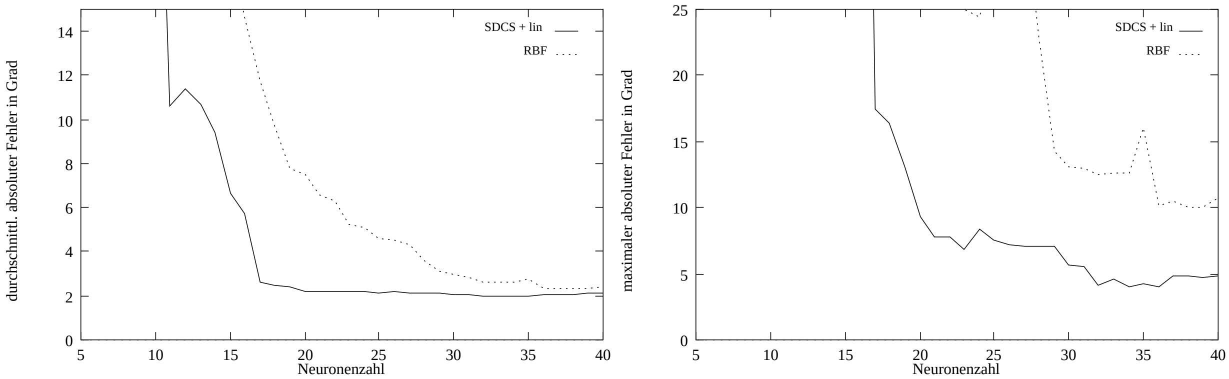

Abbildung 5.6: Training mit generierten Daten: Fehlerkurven für das Objekt „Zange“ auf den Testdaten in Grad eines SDCS-Netzes mit linearer Erweiterung und eines RBF-Netzes mit optimal angepaßten Varianzen. Links: Durchschnittlicher, absoluter Fehler. Rechts: Maximaler, absoluter Fehler.

5.3 Versuchsreihen für die Greifaufgabe

In diesem Abschnitt folgt nun der Test des Gesamtsystems zur Erkennung, Posenbestimmung und Lokalisation eines Objektes. Als Versuchsobjekte dienen im ersten Versuch zunächst sechs Objekte. Sie werden "Hammer", "Kabelzange", "Schraubenschlüssel", "Spachtel", "Tesarolle" und "Schere" genannt und sind in Abbildung 5.7 zu sehen. Es wird künstliches, gleichmäßiges Licht für die Beleuchtungsverhältnisse bei der Aufnahme der Grundansichten verwendet. In der Trainingsphase werden für jedes Objekt 359 weitere Ansichten aus der Grundansicht als Trainingsmenge generiert. Zur Extraktion der Merkmalsvektoren werden die Ansichten an 25 Ortspunkten mit Gaborfiltern in drei verschiedenen Orientierungen (0, 60, 120 Grad) und auf drei verschiedenen Skalen (34, 17, 8 Pixel) gefaltet. Die Größe der Faltungsmasken beträgt 63×63 Pixel. Die Merkmalsvektoren sind also 225-dimensional. Für jedes Objekt wird ein SDCS-Netz mit linearer Erweiterung mit den betreffenden Merkmalsvektoren trainiert. Jedes Netz wird dabei auf eine feste Größe von 30 Neuronen trainiert. Anschließend folgt die Zusammenfügung zum Gesamtnetz. Bei den Netzen dieser sechs Objekte kommt es dabei zu keinen Fehlklassifizierungen von Trainingsdaten, so daß keinerlei Veränderung der Teilnetze nötig ist.
In die Trainingsphase des Gesamtsystems fällt schließlich noch die Kalibrierung, also die

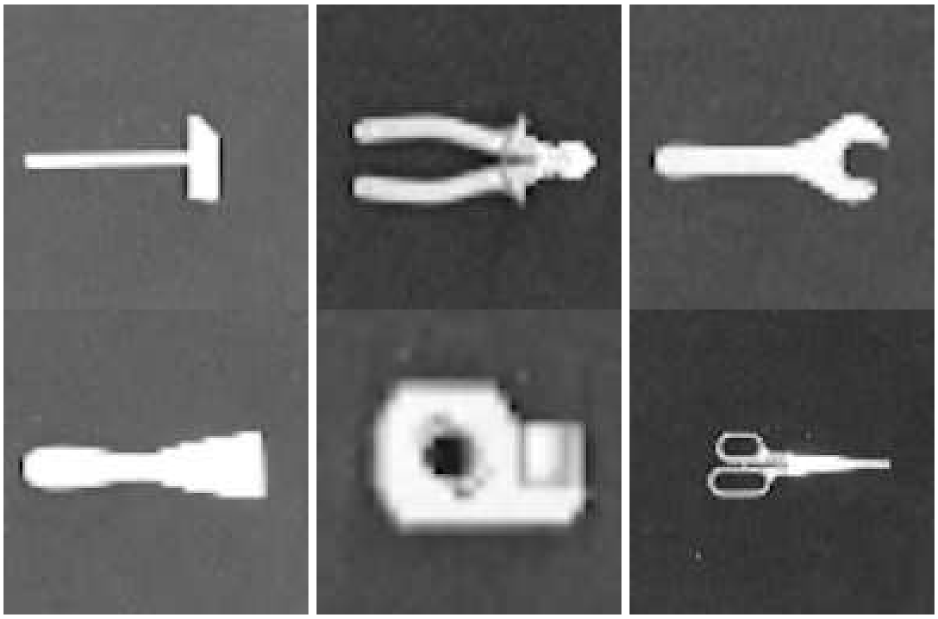

Abbildung 5.7: Die Versuchsobjekte in vorverarbeiteter Grundansicht: Von links nach rechts, von oben nach unten: „Hammer“, „Kabelzange“, „Schraubenschlüssel“, „Spachtel“, „Tesarolle“, „Schere“.

Bestimmung der Projektionsmatrix $\boldsymbol{P}$, die die Umrechnung von Bilddaten in Weltdaten realisiert. Die Berechnung der Matrix geschieht nach dem vorgestellten Verfahren, indem 50 zufällig gewählte Punkte in der Szene und deren korrespondierende Punkte im Bild bestimmt und paarweise die Differenzvektoren aller Punkte in Szene und Bild berechnet werden. Die anschließende Ausgleichsrechnung liefert die Matrix

$$\boldsymbol{P} = \begin{pmatrix} -4.005 & 0.003 \\ 0.008 & 4.073 \end{pmatrix}.$$

Das Ergebnis läßt sich folgendermaßen interpretieren: Die Matrix $\boldsymbol{P}$ ist in diesem Fall näherungsweise eine Diagonalmatrix. Durch die realitiv geringe Entfernung von der Kamera zur Arbeitsfläche treten keine wahrnehmbaren perspektivischen Verzerrungen auf, so daß die Umrechnung von Bilddaten in Realdaten eine reine Skalierung ist. Der Skalierungfaktor in x-Richtung beträgt ~ -4 und in y-Richtung ~ 4. Das bedeutet, daß ein Pixel im Bild etwa vier Milimetern in der Szene entspricht.
Um das Greifsystem nun an Realszenen zu testen, wird jedes Objekt vom Manipulator aus der Grundgreifstellung heraus an zufällig gewählten Positionen und in zufällig gewählten Orientierungen auf die Arbeitsfläche gelegt. Man erhält so feste Referenzwerte. Anschließend wird das Greifsystem auf die so konstruierte Szene angewendet. Der Vergleich zwischen den Referenzwerten und den berechneten Werten des Systems liefert die Berechnungsgüte des Systems. Die Bewertung geschieht nach den drei, schon bekannten Kriterien:

1. Objekterkennung: Stimmt das klassifizierte Objekt mit dem tatsächlich in der Szene befindlichen Objekt überein?

2. Posenbestimmung: Berechnung der absoluten Differenz $\triangle\phi$ zwischen berechneter und tatsächlich vorliegender Orientierung in Grad.

3. Lokalisation: Berechnung der absoluten Differenz $\triangle x$ in x-Richtung und $\triangle y$ in y-Richtung zwischen berechneter und tatsächlicher Position in Millimetern.

5.3.1 Versuch mit konstanten Beleuchtungsverhältnissen

Jedes der sechs Objekte wird mittels des Manipulators zehnmal in der Szene positioniert, und man erhält somit ingesamt 60 Szenenbeispiele. Die Aufnahme der jeweiligen Szene geschieht bei künstlichem Licht, so daß in Trainings- und Testphase ähnliche Beleuchtungsverhältnisse vorherrschen. Aus der jeweiligen Aufnahme berechnet das System die Klassifizierung, Posenbestimmung und Lokalisation. Der Vergleich mit den Referenzwerten findet sich in der folgenden Tabelle 5.1. Er ist in den durchschnittlichen ($\emptyset\ \triangle\phi$) und maximalen (MAX $\triangle\phi$), absoluten Fehler der Orientierung und den durchschnittlichen ($\emptyset\ \triangle x$ bzw. $\emptyset\ \triangle y$) und maximalen (MAX $\triangle x$ bzw. MAX $\triangle y$), absoluten Fehler der Position in x- bzw. y-Richtung unterteilt.

Objekt	Orientierungsfehler		Positionsfehler			
	$\emptyset\ \triangle\phi$	MAX $\triangle\phi$	$\emptyset\ \triangle x$	MAX $\triangle x$	$\emptyset\ \triangle y$	MAX $\triangle y$
Schraubenschlüssel	1.09	4.27	3.07	8.01	1.87	7.44
Kabelzange	1.00	2.35	2.96	7.08	2.42	6.03
Hammer	1.29	3.21	4.38	8.68	3.18	7.09
Schere	1.91	5.12	5.09	11.67	4.86	12.84
Spachtel	2.03	3.64	4.63	7.84	4.34	10.81
Tesarolle	2.49	4.99	3.03	7.05	3.13	7.35
Gesamt	**1.63**	**5.12**	**3.92**	**11.67**	**3.30**	**12.84**

Tabelle 5.1: Die Ergebnisse der Posenbestimmung und Objektlokalisation unter künstlichen Lichtbedingungen.

Es kommt bei der Objekterkennung zu keiner Fehlklassifizierung. Die gewählten Merkmalsvektoren haben also genügend diskriminierende Eigenschaften, um die Objekte bei beliebiger Orientierung unterscheiden zu können. Das Ergebnis der Posenbestimmung besitzt die Güte, die wir schon in dem vorherigen Versuch erreicht haben. Der durchschnittliche, absolute Fehler aller Versuchsbeispiele liegt mit 1.63 Grad im erwarteten Bereich, der maximale Fehler mit 5.12 Grad ebenfalls. Bei den Objekten „Schere", „Spachtel" und „Tesarolle" ergibt sich im Vergleich ein etwas schlechteres Ergebnis. Bei der „Schere" und dem „Spachtel" liegt dies nicht allein am neuronalen System, sondern auch daran, daß an diesen Objekten keine stabile Greifposition gefunden werden kann. Dadurch kommt es

beim Positionieren der Objekte in der Szene und damit bei der Angabe der Referenzwerte zu Ungenauigkeiten, weil das Objekt beim Ablegen auf die Arbeitsfläche verrutschen kann. Bei der „Tesarolle“ resultiert der etwas höhere Fehler aus dem kleineren Umfang des Objektes im Verhältnis zu den anderen Objekten. Die Ansicht muß mit einem höheren Faktor skaliert werden, wodurch größere Ungenauigkeiten in der skalierten Ansicht auftreten als bei den übrigen Objekten. So benötigt die "Tesarolle" einen Skalierungsfaktor von 4.5, während der Faktor der übrigen Objekte im Schnitt 2.7 beträgt.
Man betrachte nun das Ergebnis der Lokalisation. Trotz des relativ einfachen Verfahrens zeigt sich, daß das gewählte Vorgehen Ergebnisse liefert, die für ein Greifvorgang ausreichend sind. Die Abweichung von der Referenzposition ist in einen Fehler in x-Richtung und einen in y-Richtung aufgeschlüsselt. Der durchschnittliche Fehler liegt in beiden Richtungen bei c.a. 4 Millimetern. Es kommt jedoch in einigen Szenenbeispielen zu deutlicheren Abweichungen bis zu 12 Millimetern. Eine Fehlerquelle ist dabei wiederum in der teilweise ungenauen Positionierung eines Objektes durch den Manipulator zu finden. Das jeweilige Objekt läßt sich aber auch bei dieser Abweichung sicher greifen.

5.3.2 Versuch mit veränderten Beleuchtungsverhältnissen

In einem weiteren Versuch soll bei der Aufnahme der Beispielszenen anstatt künstlichem Licht Tageslicht verwendet werden. Man erhält damit in der Testphase veränderte Beleuchtungsbedingungen im Vergleich zur Trainingsphase, denn das Tageslicht besitzt eine höhere Helligkeit. Die Pixel, die zum Hintergrund gehören, besitzen dabei im Durchschnitt einen um 30 Einheiten (der multiplikative Faktor beträgt c.a. 1.75) höheren Grauwert als die in den Ansichten bei künstlichem Licht. Hiermit soll die Helligkeitsinvarianz des Systems getestet werden. Jedes Objekt wurde wiederum zehnmal in der Szene positioniert und das Greifsystem auf die insgesamt 60 Ansichten angewendet. Das Ergebnis zeigt sich in der folgenden Tabelle:

Objekt	Orientierungsfehler in Grad		Positionsfehler in mm			
	$\emptyset\ \triangle\phi$	MAX $\triangle\phi$	$\emptyset\ \triangle x$	MAX $\triangle x$	$\emptyset\ \triangle y$	MAX $\triangle y$
Schraubenschlüssel	1.34	5.61	3.13	5.92	2.83	6.69
Kabelzange	1.33	2.90	5.90	11.00	1.23	4.27
Hammer	1.43	3.54	2.62	5.63	2.67	6.83
Schere	1.57	2.62	6.81	13.40	7.72	15.27
Spachtel	2.24	4.13	6.03	17.01	4.90	17.01
Tesarolle	1.93	3.37	3.60	7.35	3.91	9.10
Gesamt	**1.64**	**5.61**	**4.68**	**17.01**	**3.87**	**15.27**

Tabelle 5.2: Die Ergebnisse der Posenbestimmung und Objektlokalisation unter Tageslichtbedingungen.

Es kommt auch hier zu keinen Fehlklassifizierungen. Betrachtet man weiterhin das Ergebnis der Posenbestimmung, so sieht man, daß sich sehr ähnliche Ergebnisse ergeben wie bei dem Versuch unter künstlichem Licht. Sowohl der durchschnittliche als auch der maximale Fehler haben sich nur unwesentlich verändert. Es bestätigt sich also die Helligkeitsinvarianz des Systems in Bezug auf Objekterkennung und Posenbestimmung.
Auf die Objektlokalisation hat die Helligkeitsänderung jedoch einen leichten Einfluß. Der durchschnittliche Fehler sowohl in x- als auch in y-Richtung ist um ca. 1 mm gestiegen. Dies hat folgenden Grund: Die Lokalisation geschieht über den Pixelschwerpunkt eines Objektes im Bild. Die Entscheidung, welche Pixel zum Objekt gehören und welche zum Hintergrund geschieht wiederum nach der Höhe der Grauwerte der Pixel über ein Schwellwertverfahren. Durch die Erhöhung der Helligkeit werden also in der Regel dem Objekt mehr Pixel zugeordnet, wodurch es zur Verschiebung des Schwerpunktes kommen kann. Um dem entgegenzuwirken, kann eine Erhöhung des Schwellwertes vorgenommen werden. Da die Fehler der berechneten Werte weiterhin im Toleranzbereich für den Greifvorgang liegen, wird auf eine a priori-Veränderung des Schwellwertes verzichtet.

5.3.3 Versuch mit zueinander ähnlichen Objekten

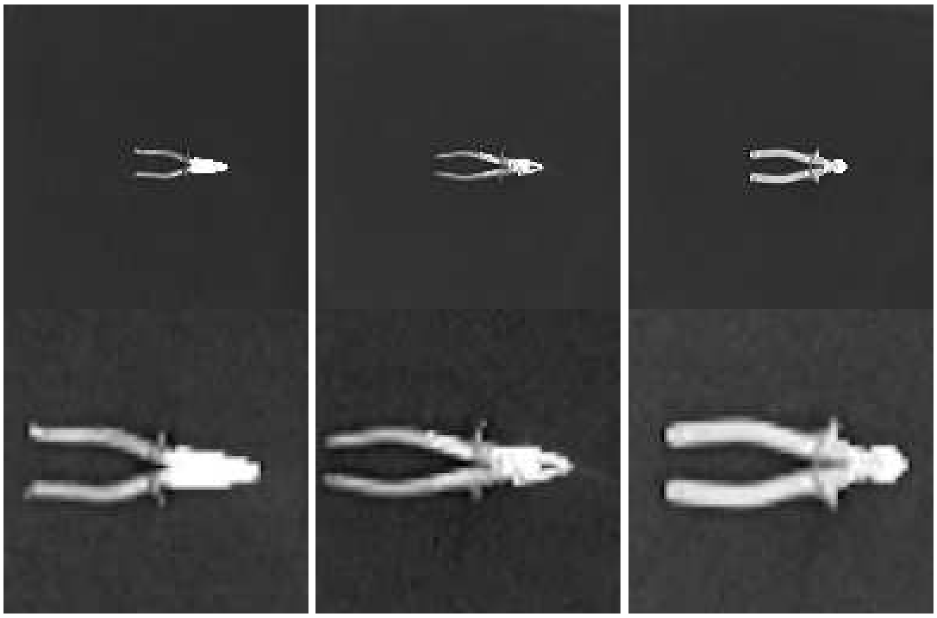

Abbildung 5.8: Drei Versuchsobjekte mit ähnlichem Aussehen: Von links nach rechts: „kleine Zange“, „Zange“, „Kabelzange“. In der oberen Reihe jeweils die originalen Grundansichten, in der unteren Reihe die vorverarbeiteten Grundansichten.

Die sechs bisher verwendeten Versuchsobjekte unterscheiden sich relativ stark voneinander, so daß keine zu großen Anforderungen an die Objekterkennung gestellt werden. Daher werden in diesem Versuch drei Objekte verwendet, die sich vom Erscheinungsbild her sehr ähnlich sind. Es sind drei verschiedenartige Zangen, sie werden als „kleine Zange“, „Zange“

und „Kabelzange“ bezeichnet. Die „Kabelzange“ und die „kleine Zange“ haben ungefähr die gleiche Größe, während die „kleine Zange“ und die „Zange“ eine ähnliche Form besitzen, jedoch unterschiedlich groß sind. Die „Kabelzange“ und die „Zange“ haben subjektiv die geringste Ähnlichkeit. Es wird für jedes Objekt ein Netz mit jeweils 30 Neuronen trainiert. Bei der Zusammenfügung der Netze zum Gesamtnetz kommt es im Test der Trainingsdaten wiederum zu keinen Fehlklassifizierungen. Es wird wieder jedes der drei Objekte zehnmal in der Szene positioniert. Als Beleuchtung wird zunächst künstliches Licht gewählt.
Es soll hier nur das Ergebnis der Objekterkennung betrachtet werden, da die Posenberechnung und die Objektlokalisation in den vorherigen Versuchen ausreichend getestet worden sind. In der Tabelle 5.3 sind die Ergebnisse der Objekterkennung zu sehen.

Nr.	Objekt	Anzahl			
		Klassifizierungen als Objekt			Fehlklassifizierungen
		1	2	3	
1	kleine Zange	10	0	0	0
2	Zange	2	8	0	2
3	Kabelzange	0	0	10	0
Gesamt		12	8	10	2

Tabelle 5.3: Die Ergebnisse der Objekterkennung unter künstlichen Lichtbedingungen.

Es kommt bei den insgesamt 30 Beispielen zu zwei Fehlklassifizierungen. In beiden Fällen wird das Objekt „Zange“ als Objekt „kleine Zange“ erkannt, also als das Objekt, welches von der Form her sehr ähnlich, jedoch kleiner ist. Durch die Skalierung der Ansichten erhalten die Objekte aber eine konstante Größe im Bild, so daß sich die vorverarbeiteten Ansichten der beiden Objekte bei identischer Orientierung sehr ähnlich sind, woraus eine Fehlklassifizierung resultieren kann. Es sei jedoch angemerkt, daß die Pose auch bei einer Fehlklassifizierung weiterhin sehr genau approximiert wird.
Auch für diese Objekte soll das Verhalten des Systems bei natürlicher Beleuchtung durch Tageslicht getestet werden, also bei größerer Helligkeit in der Szene. Die Auswertung dieses Versuchs in Tabelle 5.4 läßt erkennen, daß dabei insgesamt 4 Fehlklassifizierungen auftreten. Wiederum wird das Objekt „Zange“ zweimal als Objekt „kleine Zange“ erkannt. Zusätzlich treten beim Objekt „kleine Zange“ zwei Fehlklassifizierungen auf. In beiden Fällen wird es als „Kabelzange“ erkannt. Auch hier zeigt sich jedoch die Posenberechnung gegenüber der Fehlklassifizierung robust. Beim Objekt „Kabelzange“ geschieht die Erkennung weiterhin fehlerfrei. Es sei daran erinnert, daß diese beiden Objekte subjektiv die geringste Ähnlichkeit besitzen.

Nr.	Objekt	Anzahl			
		Klassifizierungen als Objekt			Fehlklassifizierungen
		1	2	3	
1	kleine Zange	8	0	2	2
2	Zange	2	8	0	2
3	Kabelzange	0	0	10	0
Gesamt		10	8	12	4

Tabelle 5.4: Die Ergebnisse der Objekterkennung unter Tageslichtbedingungen.

5.3.4 Resümee

Das System wurde in den vorangegangenen Versuchsreihen auf seine Leistungsfähigkeit getestet, dabei die Rahmenbedingungen variiert und die Versuchsobjekte in unterschiedlichen Kombinationen zu Testmengen zusammengestellt. Als Fazit kann man generell feststellen, daß das so konzipierte System die gestellte Aufgabe zufriedenstellend lösen kann. Die Objekterkennung funktioniert unter unterschiedlichen Bedingungen nahezu fehlerlos, erst bei untereinander sehr ähnlichen Objekten kommt es zu einigen wenigen Fehlklassifizierungen, jedoch wird der Großteil der Beispiele in diesen Versuchsreihen weiterhin richtig erkannt.

Das Ergebnis der Posenbestimmung schließt sich dem an. Die Approximationsfehler liegen im Toleranzbereich für einen realen Greifvorgang. Das System zeigt sich dabei unbeeinflußt von Änderungen der Beleuchtungsbedingungen. Auch die etwas stärkere Orientierungsabweichung bei der Posenapproximation einiger „Ausreißer“ ist nicht so vehement, daß sie zu Problemen beim Greifen führt. Weiterhin erwies sich die Posenbestimmung sogar robust gegenüber den Fehlklassifizierungen.
Die Objektlokalisation ist im Vergleich der beiden anderen Prozesse das fehleranfälligere Verfahren. Es zeigt zum Beispiel eine Anfälligkeit gegenüber Helligkeitsschwankungen. Aber auch diese erzielten Ergebnisse sind für die Realisierung eines Greifvorgangs ausreichend.

Kapitel 6

Diskussion und Ausblick

6.1 Diskussion

In dieser Arbeit wird die Entwicklung eines lernenden Systems vorgestellt, welches in der Lage ist, allein durch visuelle Informationen gelernte Objekte zu unterscheiden und zu greifen. Dafür ist neben der Klassifizierung die Orientierungbestimmung der Objekte in der Szene von Hauptinteresse. Die Orientierung der Objekte unterliegt dabei einer gewissen Einschränkung, sie ist auf einen Freiheitsgrad beschränkt, nämlich eine beliebige Rotation um eine vertikale Achse. Die Erkennung und Posenbestimmung geschieht über die photometrischen Eigenschaften der Objekte, das heißt die Erscheinung oder Ansicht der Objekte in einem Grauwertbild, indem durch Faltung der Ansichten an festen Ortspunkten Merkmalsvektoren extrahiert werden. Es zeigt sich, daß sich als Faltungsoperatoren Gaborfilter eignen, da sie als orientierungsselektive Bandpaßfilter gleichsam einen Beitrag zur Klassifizierung und Orientierungsbestimmung leisten. Weiterhin erreicht man durch die Gleichanteilsfreiheit des Gaborfilters eine gewisse Helligkeitsinvarianz in den Merkmalsvektoren.
Eine wesentliche Eigenschaft der so gewählten Merkmalsvektoren ist, daß Merkmalsvektoren von Ansichten desselben Objektes einen Unterraum aufspannen, während die Merkmalsvektoren von verschiedenen Objekten in unterschiedlichen Unterräumen liegen. Die Eigenschaft resultiert unter anderem aus der Verwendung von Gaborfiltern mit unterschiedlichen Bandbreiten.
Diese Eigenschaft macht man sich zu Nutze, indem die Approximation der Orientierung auf den Merkmalsvektoren über spezielle neuronale Netze geschieht, welche mit einer Menge von Ansichten des jeweiligen Objektes in verschiedenen Orientierungen trainiert werden, also die Merkmalsvektoren als Eingabe erhalten. Ausgehend von einem klassischen RBF-Netz wird veranschaulicht, wie die Verwendung einer topologischen Karte im Merkmalsraum für den Approximationsvorgang eines neuronalen Netzes genutzt werden kann. Eine solche topologische Karte beschreibt die Topologie des Unterraums, in dem die Eingabedaten liegen, und berechnet dessen intrinsische Dimensionalität. Die Verwendung

einer Karte ermöglicht einerseits einen Effizienzgewinn, denn die Ausgabe berechnet sich im Gegensatz zum RBF-Netz nicht aus allen Knoten des Netzes, sondern nur aus denen im lokalen Bereich eines Stimulus, wie zunächst die Verwendung des DCS-Netzes zeigt. Andererseits kann damit auch die Approximationsgüte verbessert werden, denn mit Hilfe der topologischen Karte können verrauschte Daten erkannt werden, wenn sie sich außerhalb der Mannigfaltigkeit befinden. Durch Projektion zurück in den Unterraum läßt sich der Rauschanteil herausfiltern. Das Gesamtsystem ergibt sich aus der Zusammenfügung aller Netze zu einem Netzkonstrukt. In dem Schritt erfolgt zusätzlich eine Anpassung der Teilnetze, welche mögliche Defekte bei der Klassifizierung der Objekte vermeiden soll.
Zum System gehört schließlich noch ein Verfahren zur Objektlokalisation in der Szene. Auf der Basis der Flächenschwerpunkte der Objekte im Bild wird die Translation des jeweiligen Objektes aus einer Ausgangsposition berechnet, diese Translation in Weltdaten umgerechnet und so die durchzuführende Bewegung des Manipulators zum Objekt in der Realszene bestimmt.
Die anschließenden Versuchsreihen zeigen die Leistungsfähigkeit des Gesamtsystems. Bei Objekten, die sich relativ stark voneinander unterscheiden, funktioniert die Objekterkennung fehlerfrei. Aber auch bei Objekten mit großer Ähnlichkeit zueinander wird eine hohe Erfolgsquote in der Erkennung erreicht. Die Posenbestimmung liefert ebenfalls ein gutes Ergebnis und erweist sich auch gegenüber Helligkeitsveränderungen robust. Die Objektlokalisation dagegen weist in Einzelfällen gerade bei Helligkeitsänderungen etwas größere Abweichungen auf, die sich jedoch noch im Toleranzbereich für einen Greifvorgang bewegen.

6.2 Ausblick

Das System ist in der vorgestellten Konfiguration in der Lage, die gestellte Greifaufgabe zufriedenstellend zu lösen. Aber es existieren noch Möglichkeiten zur Erweiterung, die eine weitere Verbesserung des Systems ermöglichen. Auch sind neue Anforderungen denkbar, die an das System gestellt werden, indem bestimmte Konventionen in der Aufgabenstellung aufgehoben werden. Es sollen im folgenden dafür einige Beispiele gegeben werden.

Verwendung von Farbbildern

Bei sich ähnelnden Objekten kann es bei der Erkennung zu Fehlklassifizierungen kommen, wie die Versuche zeigen. Eine Idee ist es, dieses Problem durch die Wahl von mehr Ortspunkten bei der Faltung zu lösen. Alternativ kann man dagegen anstatt Grauwertbildern Farbilder verwenden, so daß Farbunterschiede der Objekte die Erkennung unterstützen. Es wird dabei ein Pixel nicht mehr nur durch einen Grauwert beschrieben, sondern durch jeweils drei sogenannte *Farbbänder* in einem festen Farbwertbereich. Die Faltung des Bildes geschieht getrennt auf den Bändern. Man erhält damit also auch höherdimensionale Merkmalsvektoren.

Beachtung der Symmetrieeigenschaften von Objekten

Manche Objekte besitzen gewisse Symmetrieeigenschaften, so daß sich die Erscheinung eines Objektes im Bild nicht, wie bisher, nach einer Rotation von 360 Grad wiederholt, sondern schon nach einer Drehung um 180 Grad oder sogar weniger. Dies kann bei dem bisherigen Verfahren zu Konflikten bei der Approximation der Pose führen. Ist es dagegen möglich, diese Symmetrien zu erkennen, so kann dieses Wissen zur Verbesserung der Posenbestimmung genutzt werden, denn es sind weniger Ansichten nötig, um ein solches Objekt ausreichend zu beschreiben.

Mehr Freiheitsgrade in der Orientierung

In dieser Arbeit wird die Pose der Objekte beschränkt, indem nur eine Rotation um eine vertikale Achse als Freiheitsgrad zugelassen wird. Die intrinsische Dimension der Eingabevektoren beträgt damit ebenfalls eins, und es ist bei den Nachbarschaftsbeziehungen zu erwarten, daß jeder Knoten genau zwei Nachbarn besitzt. Es stellt sich die Frage, wie sich der Unterraum der Eingabedaten und mit ihm die topologische Karte gestaltet, wenn in den Eingabedaten zwei oder mehr Freiheitsgrade zugelassen werden. So hat ein „echtes", dreidimensionales Werkstück, also ein Objekt, welches nicht als „flach" bezeichnet und damit auf ein zweidimensionales Objekt reduziert werden kann, in seiner Orientierung drei Freiheitsgrade, nämlich Rotationen um drei zueinander orthogonale Achsen. Hierbei tritt zusätzlich das Problem von Verdeckungen auf, das heißt, in bestimmten Orientierungen sind bestimmte Bereiche des Objektes nicht sichtbar, da sie von anderen Komponenten des Objektes verdeckt werden. Auch sind Fremdüberdeckungen durch andere Objekte (schon im 2-D Fall) möglich. Es ist interessant zu untersuchen, wie sich die Extraktion von Merkmalsvektoren und die darauf aufbauende Approximation der Pose durch ein neuronales Netzwerk unter diesen Umständen verhält. Einen nicht neuronalen Lösungsansatz zur Behandlung von Verdeckungen beschreibt [11].

Anhang A

Beweise einiger Sätze

A.1 Beweis von Satz 1 (siehe [14])

$$\begin{aligned}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{v}}_i = \mu_i\tilde{\boldsymbol{v}}_i \quad &\Leftrightarrow \quad \tfrac{1}{N}\boldsymbol{A}\boldsymbol{A}^T\tilde{\boldsymbol{v}}_i = \mu_i\tilde{\boldsymbol{v}}_i \qquad\qquad \mid \boldsymbol{A}^T \\ &\Leftrightarrow \quad \tfrac{1}{N}\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{A}^T\tilde{\boldsymbol{v}}_i = \mu_i\boldsymbol{A}^T\tilde{\boldsymbol{v}}_i \\ &\Leftrightarrow \quad \boldsymbol{\Sigma}\boldsymbol{A}^T\tilde{\boldsymbol{v}}_i = *\boldsymbol{A}^T\tilde{\boldsymbol{v}}_i\end{aligned}$$

Damit ist

$$\boldsymbol{h}_i = \boldsymbol{A}^T\tilde{\boldsymbol{v}}_i \tag{A.1}$$

Eigenvektor von $\boldsymbol{\Sigma}$, jedoch nicht normiert, und μ_i Eigenwert von $\boldsymbol{\Sigma}$.
Weiterhin gilt:

$$\begin{aligned}\boldsymbol{h}_i{}^T\boldsymbol{h}_i &= \tilde{\boldsymbol{v}}_i{}^T\boldsymbol{A}\boldsymbol{A}^T\tilde{\boldsymbol{v}}_i \\ &= \boldsymbol{v}_i{}^T\underbrace{[\tilde{\boldsymbol{v}}_1,\ldots,\tilde{\boldsymbol{v}}_N]N\mathrm{diag}\{\mu_1,\ldots,\mu_N\}[\tilde{\boldsymbol{v}}_1,\ldots,\tilde{\boldsymbol{v}}_N]^T}_{\boldsymbol{A}\boldsymbol{A}^T=\tilde{\boldsymbol{\Sigma}}*N=[\cdots]N\mathrm{diag}\{\cdots\}[\cdots]}\tilde{\boldsymbol{v}}_i \\ &= \mu_i N\end{aligned}$$

Also ist

$$\boldsymbol{v}_i = \frac{\boldsymbol{A}^T\tilde{\boldsymbol{v}}_i}{\sqrt{\mu_i N}} \tag{A.2}$$

orthonormaler Eigenvektor von $\boldsymbol{\Sigma}$.

A.2 Beweis von Satz 2

Es gilt:

$$\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\tfrac{1}{2}[\boldsymbol{R}\boldsymbol{x}]^T\boldsymbol{\Sigma}[\boldsymbol{R}\boldsymbol{x}]\right)\left(\cos\left(2\pi\boldsymbol{k}(\boldsymbol{R}\boldsymbol{x}^T)\right) - d\right)d\boldsymbol{x} = 0$$

$$\Leftrightarrow$$

$$\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\tfrac{1}{2}[\boldsymbol{R}\boldsymbol{x}]^T\boldsymbol{\Sigma}[\boldsymbol{R}\boldsymbol{x}]\right)\cos\left(2\pi\boldsymbol{k}(\boldsymbol{R}\boldsymbol{x}^T)\right) - \exp\left(-\tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{\Sigma}\boldsymbol{x}\right) d \;\; d\boldsymbol{x} = 0$$

$$\Leftrightarrow$$

$$\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\tfrac{1}{2}[\boldsymbol{R}\boldsymbol{x}]^T\boldsymbol{\Sigma}[\boldsymbol{R}\boldsymbol{x}]\right)\cos\left(2\pi\boldsymbol{k}(\boldsymbol{R}\boldsymbol{x}^T)\right) d\boldsymbol{x} - \int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\tfrac{1}{2}[\boldsymbol{R}\boldsymbol{x}]^T\boldsymbol{\Sigma}[\boldsymbol{R}\boldsymbol{x}]\right) d \;\; d\boldsymbol{x} = 0$$

$$\Leftrightarrow$$

$$\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\tfrac{1}{2}[\boldsymbol{R}\boldsymbol{x}]^T\boldsymbol{\Sigma}[\boldsymbol{R}\boldsymbol{x}]\right)\cos\left(2\pi\boldsymbol{k}(\boldsymbol{R}\boldsymbol{x}^T)\right) d\boldsymbol{x} - d\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\tfrac{1}{2}[\boldsymbol{R}\boldsymbol{x}]^T\boldsymbol{\Sigma}[\boldsymbol{R}\boldsymbol{x}]\right) d\boldsymbol{x} = 0$$

$$\Leftrightarrow$$

$$d = \frac{\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\frac{1}{2}[\boldsymbol{R}\boldsymbol{x}]^T\boldsymbol{\Sigma}[\boldsymbol{R}\boldsymbol{x}]\right)\cos\left(2\pi\boldsymbol{k}(\boldsymbol{R}\boldsymbol{x}^T)\right)d\boldsymbol{x}}{\int_{\boldsymbol{x}=-\infty}^{\infty} \exp\left(-\frac{1}{2}[\boldsymbol{R}\boldsymbol{x}]^T\boldsymbol{\Sigma}[\boldsymbol{R}\boldsymbol{x}]\right)d\boldsymbol{x}} \tag{A.3}$$

Literaturverzeichnis

[1] Andrew Blake. A symmetry theory of planar grasp. *The International Journal of Robotics Research*, 14(5):425–444, October 1995.

[2] Jörg Bruske and Gerald Sommer. Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7:845–865, 1994.

[3] Jörg Bruske and Gerald Sommer. An algorithm for intrinsic dimensionality estimation. In *Conference for Analysis of Images and Patterns - CAIP*. Computer Science Institute, Christian-Albrechts-University Kiel, 1997.

[4] Jörg Bruske and Gerald Sommer. Intrinsic dimensionality estimation with optimally topology preserving maps. Technical report, Computer Science Institute, Christian-Albrechts-University Kiel, 1997.

[5] Bernd Fritzke. Fast learning with incremental rbf networks. *Neuronal Processing Letters*, 1(1):2–5, 1994.

[6] Patrick Gros. Matching and clustering: Two steps toward automatic object modeling in computer vision. *The International Journal of Robotic Research*, 14(6):633–642, 1995.

[7] Don R. Hush and Bill G. Horne. Progress in supervised neural networks. In *IEEE Signal Processing Magazine*, 1993.

[8] Bernd Jähne. *Digitale Bildverarbeitung*. Springer-Verlag, 1993.

[9] C. Koch and S. Ullmann. Shifts in selective visual attention: Towards the underlying neural circuitry. *Human Neurobiol*, 4:219–227, 1985.

[10] T. Kohonen. *Self-Organzizing Maps*. Springer Verlag, Berlin Heidelberg, 1995.

[11] A. Leonardis and H. Bischof. Robust recovery of eigenimages in the presence of outliers and occlusions. *Journal of Computing and Information Technology - CIT 4*, 1:25–38, 1996.

[12] T. Martinetz and K. Schulten. Topology representing networks. *Neuronal Networks*, 7:505–522, 1994.

[13] M. Michaelis. *Low Level Image Processing Using Steerable Filters.* Diss. Universität Kiel, Techn. Fak., 1995.

[14] H. Murase and S. K. Nayar. Visual learning and recognition of 3-D objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.

[15] William H. Press. *Numerical Recipes in C.* Cambridge University Press, 1992.

[16] H. Ritter, T. Martinetz, and K. Schulten. *Neuronale Netze - Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke.* Addison-Wesley, 1991.

[17] Rojas. *Theorie der neuronalen Netze.* Springer-Lehrbuch, 1993.

[18] Gerald Sommer, Kostas Daniilidis, and Udo Mahlmeister. Skript zur Vorlesung Computer Vision, 1994.

[19] Michael Taylor, Andrew Blake, and Adrian Cox. Visually guided grasping in 3D. *IEEE International Conference on Robotics and Automation*, 1994.

[20] Z. Zhang and O. Faugeras. *3D Dynamic Scene Analysis*, chapter 10. Springer-Verlag, 1992.

Danksagung

Hiermit möchte ich meinen Betreuern Dipl.-Inform Jörg Bruske und Dr. Josef Pauli für die uneingeschränkte Unterstützung bei der Erstellung dieser Arbeit danken. Ich danke Britta und Erika für das Korrekturlesen und weiterhin allen Mitgliedern des Lehrstuhls, die auf vielfältige Weise zum Gelingen meiner Diplomarbeit beigetragen haben.
Ein besonderer Dank gilt meiner Mutter für ihre geduldige Unterstützung während meines gesamten Studiums.

Erklärung:

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

Kiel, den