Schnelle Objekterkennung für die Robotik
Proseminar Bioinformatik SS 1999

Manfred Eckschlager
meckschl@cosy.sbg.ac.at
Thomas Gangl
tgangl@cosy.sbg.ac.at
Michael Liedlgruber
mliedl@cosy.sbg.ac.at

Version: 30. Juni 1999

LV-Leiter: R. Schwaiger

18. März 1999

Zusammenfassung

Dieses Dokument stellt einen groben Überblick über Methoden zur schnellen Objekterkennung dar. Im Zuge dessen, werden anfangs die der Objekterkennung zugrunde liegenden Verfahren betrachtet und verglichen. Erst danach erfolgt eine nähere Betrachtung der wichtigsten Algorithmen und deren Realisierung mit Hilfe von neuronalen Netzen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dieses Dokument beschäftigt sich mit den Methoden der schnellen Objekterkennung. Die Objekterkennung soll derart realisiert werden, daß sie geometrische Strukturen aus einem über eine Hardware eingelesenen Rasterbild extrahieren kann. Dazu werden künstliche neuronale Netze (ANN) herangezogen.

1.1 Motivation

Das in diesem Projekt erarbeitete System soll in einem vom Institut für Computerwissenschaften und Systemanalyse entwickelten Roboter namens EMMA[1] eingesetzt werden. Die zur Verfügung stehende Hardware besteht aus einer CCD-Kamera mit 64 Graustufen und einer Auflösung von 80x60 Bildpunkten, 2x512kb RAM und einem SAB C176CR CPU. Dieser Roboter soll in einer homogenen Umwelt gegen einen anderen Roboter Fußball spielen. In dieser Umwelt existieren nur zwei Tore, ein Ball und der Gegner. EMMA erhält jede halbe Sekunde ein neues Bild von der CCD-Kamera, welches die Grundlange für die Objekterkennung ist.

1.2 Was sind Neuronale Netze

Neuronale Netze (NN), oft auch als künstliche neuronale Netze (KNN) oder artificial neural networks (ANN) bezeichnet, sind informationsverarbeitende Systeme, die aus einer großen Anzahl einfacher Einheiten (Zellen, Neuronen) bestehen, die sich Information in Form der Aktivierung der Zellen über gerichtete Verbindungen (connections, links) zusenden.[?] Neuronale Netze eignen sich aus folgenden Gründen am besten zur Objekterkennung sprich Analyse von eingelesenen Bildern: neuronale Netze sind lernfähig, das heißt, sie können sich neuen Anforderungen anpassen, sie sind robust gegenüber Störungen oder verrauschten Daten und haben eine hohe Fehlertoleranz. Außerdem sind sie durch ihre starke Parallelität sehr schnell, jedoch schwindet dieser Vorteil bei der Simulation durch den Computer.

[1] EMbedded Mobile Agent

1.3 Die biologische Motivation

Bevor man technische neuronale Netze betrachtet, ist es interessant, das biologische Vorbild, Nervenzellen und Zellverbände von Tieren bzw. Menschen, anzusehen. Man stellt dabei sehr schnell fest, daß die technische Realisierung künstlicher neuronaler Netze nur sehr wenig mit dem biologischen Vorbild zu tun hat, weil sie zu vieles zu stark vereinfacht. Dies ist auch einer der Hauptkritikpunkte von Forschern aus den Biowissenschaften an den aus der Physik, Informatik und Mathematik stammenden vereinfachten Modellen. Andererseits sind nur diese stark vereinfachten Modelle derzeit einer genaueren theoretischen Analyse zugänglich. Auch die hier gegebene Übersicht über biologische Neuronen ist bereits stark vereinfacht.

1.3.1 Der Aufbau einer Nervenzelle

Nervenzellen, die Grundbausteine des Gehirns, unterscheiden sich von anderen Zellen, mit denen sie sehr viele Gemeinsamkeiten haben, vor allem in der Zellform, der Art der Zellmembran und der Eigenschaft, an den Enden von Ausläufern der Zellen Verdickungen (Synapsen) bilden zu können, die mit anderen Nerven- oder Muskelzellen in Kontakt treten und Nervenimpulse weiterleiten können. Weiterhin können sich Nervenzellen im Unterschied zu anderen Zellen nach Abschluß der Embryonalentwicklung nicht mehr teilen. Die meisten Schätzungen gehen von 10^{11} Nervenzellen des Menschen aus. Obwohl sie alle unterschiedliche Form haben, kann man sie grob nach ihrer Form und ihrer Lage in verschiedenen Bereichen des Gehirns in Klassen aufteilen. Unipolare Zellen haben neben dem Zellkörper (Soma) mit Zellkern (Nucleus) nur einen Fortsatz, die Nervenfaser (Axon), bipolare Zellen zwei (ein Axon und einen Dendriten), multipolare Zellen besitzen ein Axon und viele Dendriten. Abbildung 1.3.1 zeigt verschiedene Klassen von multipolaren Zellen. Multipolare Zellen kommen nur in Wirbeltieren vor.

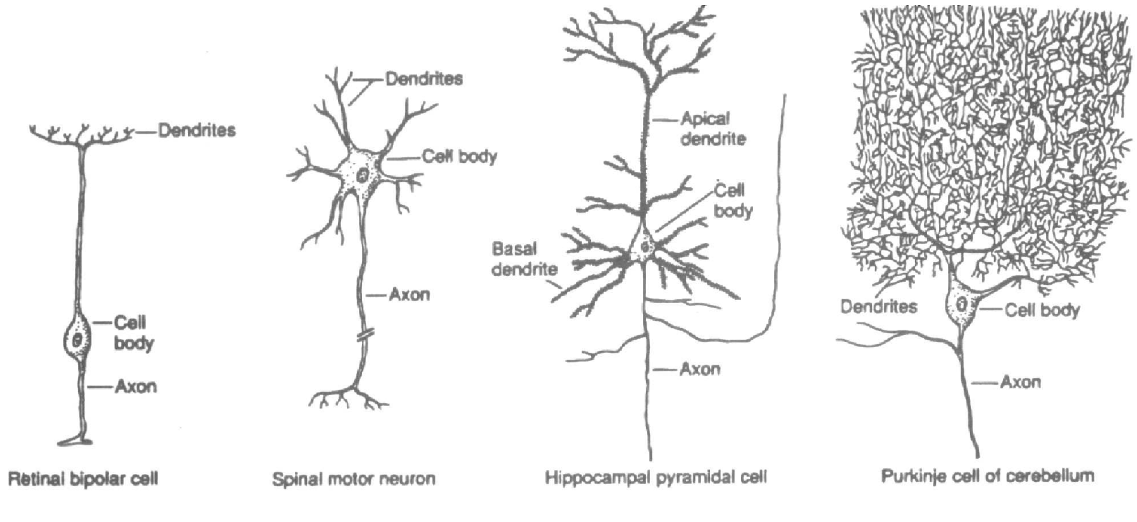


Abb. 1.3.1: Verschiedene Typen multipolarer Nervenellen [?].

Der kugel- oder pyramidenförmige Zellkörper enthält den Zellkern und viele Organellen, wie Mitochondrien oder das endoplasmatische Retikulum, welche die Energieversorgung der Zelle decken bzw. Enzyme und andere notwendige Zellbestandteile herstellen. Vom endoplasmatischen Retikulum und dem Golgi-Apparat werden auch die synaptischen Vesikel produziert, kleine Bläschen, welche chemische Botenstoffe, die Neurotransmitter, enthalten, die zur Weiterleitung der Nervenimpulse zwischen Zellen benötigt werden. Dendriten sind dünne, röhrenförmige und meist stark verästelte Fortsätze der Zelle, mit denen die Zelle Eingangssignale aufnimmt. Die Weiterleitung der elektrischen Nervenimpulse erfolgt über die normalerweise viel längere Nervenfaser, das Axon. Das Axon verästelt sich erst im Zielgebiet. Dort bildet es an den Enden der Verästelungen Synapsen. Üblicherweise besitzen menschliche Zellen zwischen 10^3 und 10^4 solcher Synapsen. Purkinje-Zellen besitzen bis zu $1.5 * 10^5$. Die meisten Zellen empfangen auch von etwa $2 * 10^3$ bis 10^4 Vorgängerzellen Eingangssignale. Die Synapsen, die Endknöpfchen der Nervenfaser, enthalten die synaptischen Vesikel, kleine Bläschen, die den Neurotransmitter enthalten. Ein Impuls bewirkt das Freisetzen des Neurotransmitters, welcher sich an die Rezeptoren in der Membran der Nachfolgerzelle festsetzt. Diese Rezeptoren ändern dadurch ihre Struktur und bewirken ein Einströmen von Ionen in die Dendriten. Somit wird das synaptische Potential der Nachfolgerzelle verändert.

Kleine Verbände von Nervenzellen - Die Meeresschnecke Aplysia

Während es extem schwierig ist, die Struktur und Funktion der Gehirne hochentwickelter Wirbeltiere, etwa der Säugetiere, zu untersuchen, ist es vergleichsweise einfach, Informationen über die Funktion von Zellverbänden und Zellgruppen bei einfacheren, wirbellosen Tieren zu erhalten. Diese haben eine wesentlich kleinere Zahl von Nervenzellen: bei Tintenfischen, Schnecken und Blutegeln beispielsweise findet man zwischen 10^4 und 10^5 Nervenzellen. Diese Nervenzellen sind zudem noch in räumlich getrennten Gruppen von 500 bis 1500 Zellen, den Ganglien, angeordnet, die unabhängig voneinander unteruscht werden können. Weiterhin sind diese wirbellosen Tiere teilweise durchsichtig, sodaš man einfacher mit ihnen experimentieren kann. Die Nervenzellen dieser einfachen Tiere haben keile prinipiellen Unterschiede zu denen höherentwickelter Tiere. Ein bekannter Forschungsgegenstand ist die kalifornische Meeresschnecke Aplysia californica.

Steuerung des Herzschlags und Blutdrucks von Aplysia

Abbildung 1.3.2 zeigt das Eingeweideganglion der Meeresschnecke Aplysia. Die einzelnen Zellen unterscheiden sich durch ihre Färbung, Lage, Grö̈se und Form, sowie durch ihre Neurotransmitter. Manche dieser Zellen feuern periodisch, andere nur bei Reizung. Erstaunlicherweise ist die Verteilung der Nervenzellen des Eingeweideganglions und ihr Verbindungsmuster (die "Verschaltung") bei allen Aplysia-Schnecken gleich.

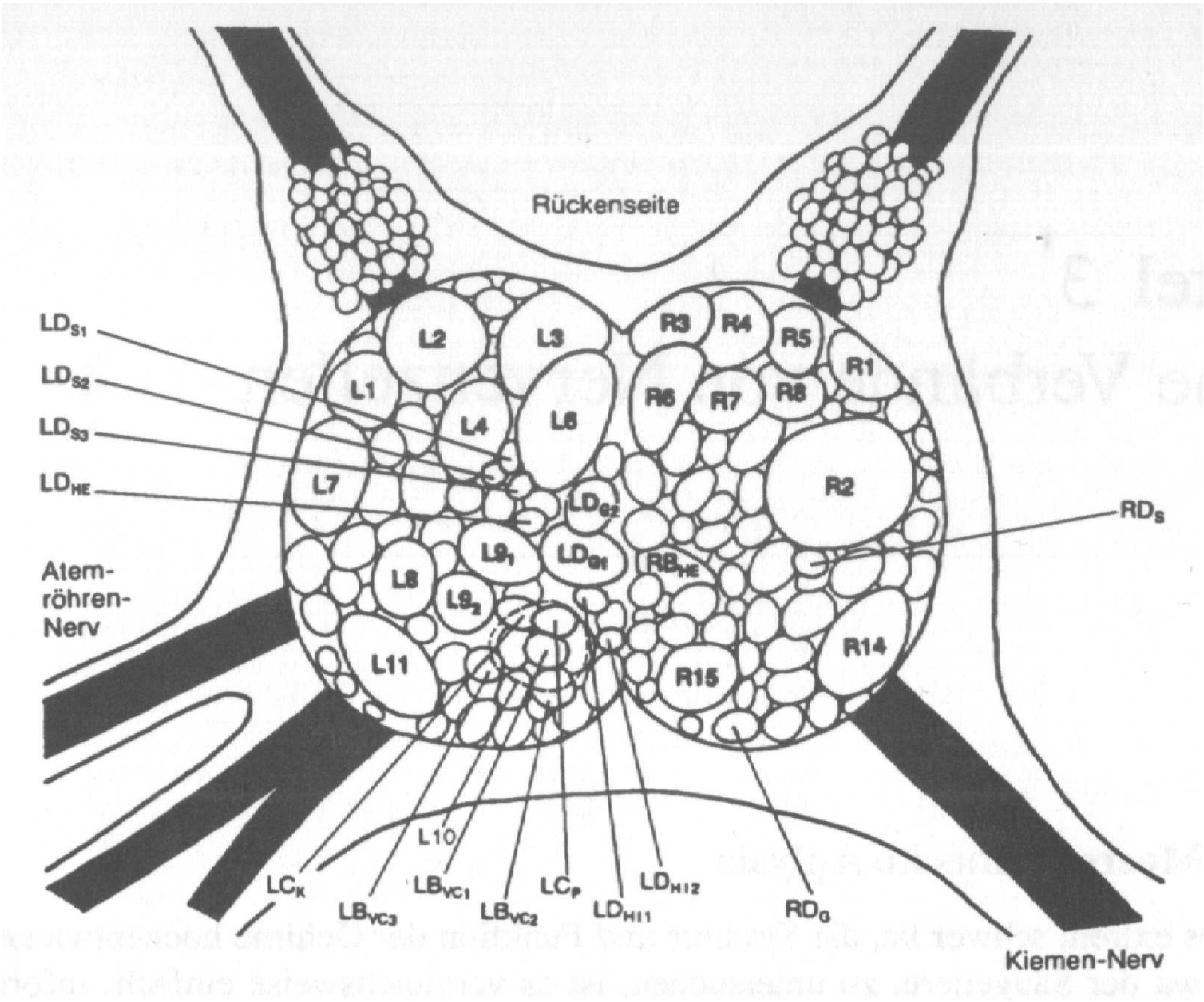


Abb. 1.3.2: Das Eingeweidenganglion der Aplysia. Die Buchstaben L und R stehen für das linke bzw. rechte Halbganglion, Doppelbuchstaben kennzeichnen Zellgruppen. Tiefgestellte Buchstaben geben Auskunft über die Funktion der Zellen, beispielsweise regen Zellen mit Index HE den Herzschlag an, Zellen mit Index HI hemmen ihn. Die Zelle L10 steuert den Herzschlag und Blutdruck von Aplysia [?].

Die Zelle L10 wirkt auf manche Zellen (RBhe) erregend, auf andere (LDhi1, LDhi2, LBvc1 bis LBvc3) hemmend, obwohl sie bei allen den Neurotransmitter Acetylcholin verwendet. die Zielzellen müssen also unterschiedliche Rezeptormoleküle in der Membran besitzen. Ein Ersatzschaltbild dieser Zellgruppe ist in Abbildung 1.3.3 dargestellt. Vier Nervenzellen genügen dieser Schnecke, um den Herzschlag zu steuern, drei weitere, um den Blutdruck zu regulieren (bei Menschen sind viele Tausend Nervenzellen mit dieser Aufgabe betraut). Sie arbeiten mit Serotonin bzw. Acetylcholin als Neurotransmitter. Sechs dieser Zellen werden von der Kommandozelle L10 gesteuert. Ein kurzes, starkes Signal von L10 hemmt die Aktivität der Zellen LDhi und verringert die Dämpfung der Herzfrequenz. Gleichzeitig wird durch Aktivierung von RBhe die Herzfrequenz gesteigert. Dadurch resultiert ein kurzzeitiger Anstieg der Herzfrequenz bei nahezu konstantem Blutdruck.

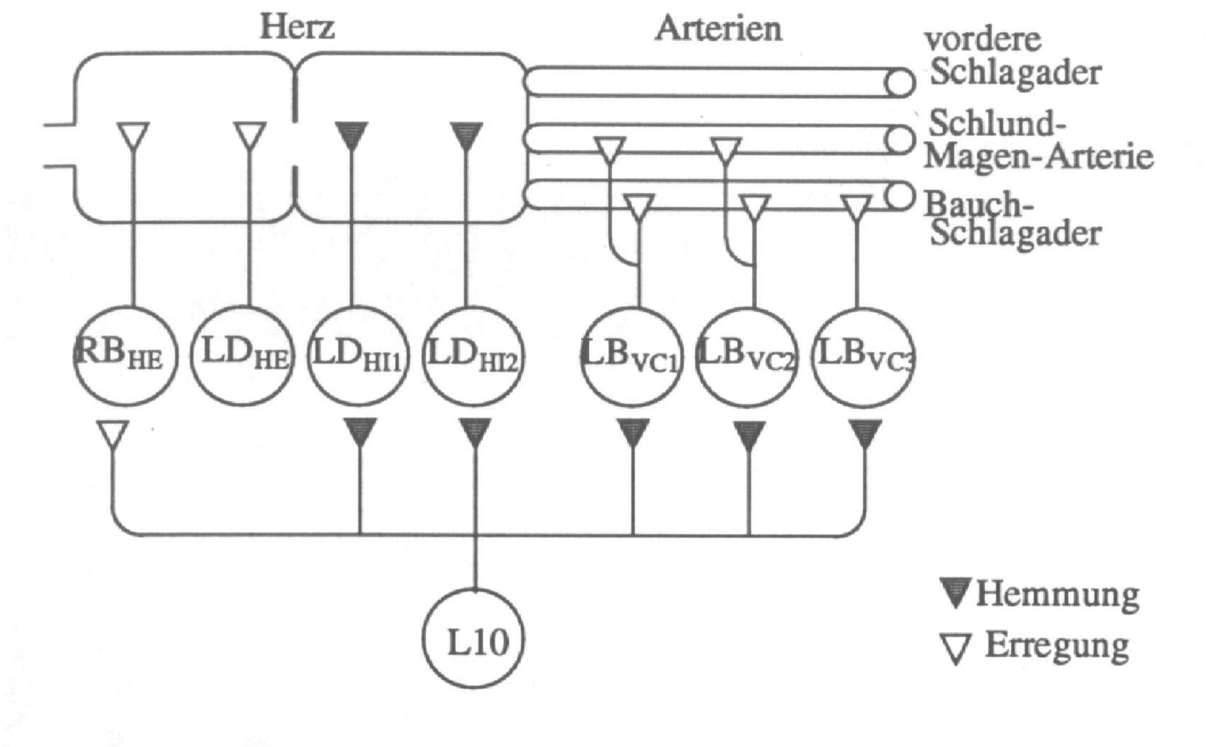


Abb. 1.3.3: Steuerung des Herzschlags und Blutdrucks von Aplysia [?].

Kapitel 2

Konzepte der neuronalen Netze

2.1 Der Grundlegende Aufbau von neuronalen Netzen

Neuronale Netze (Künstliche neuronale Netze) bestehen aus Zellen (Neuronen), welche zwar die Neuronen aus der Biologie zum Vorbild nehmen, aber dennoch stark vereinfacht und idealisiert sind. So bestehen sie in Anlehnung an das biologische Vorbild aus drei Komponenten: einem Zellkörper, den Dendriten, welche die Eingabe des Netzes in die Zelle aufsummieren, und einem Axon, welches die Ausgabe einer Zelle nach außen weiterleitet, sich verzweigt und mit Dendriten nachfolgender Neuronen über Synapsen in Kontakt tritt. Die Stärke der Verbindung zwischen zwei Zellen wird meist durch einen numerischen Wert, das Verbindungsgewicht, dargestellt (siehe Abb. 2.1.1).

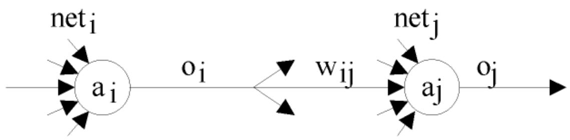


Abb. 2.1.1: Darstellung von Zellen eines neuronalen Netzes, wobei net_i (net_j) die Eingabe der Vorgängerneuronen für Zelle i (j), a_i (a_j) die aktuelle Aktivierung der Zelle i (j), o_j (o_j) die Ausgabe der Zelle i (j) und w_{ij} die gewichtete Verbindung zwischen Zellen i und j ist. [?].

2.2 Bestandteile neuronaler Netze

Neuronale Netze bestehen im allgemeinen aus folgenden Komponenten:

1. Zellen (Neuronen, Elementen, Units). Die Zellen besitzen die Bestandteile:
   - Aktivierungszustand $a_i(t)$ (activation). Er gibt den Grad der Aktivierung der Zelle i zu einem Zeitpunkt t an.

- Aktivierungsfunktion f_{act}. Sie gibt an, wie sich ein neuer Aktivierungszustand $a_i(t+1)$ aus dem alten Aktivierungszustand $a_i(t)$ und der Netzeingabe $net_j(t)$ ergibt, meist nach der allgemeinen Formel

  $$a_i(t+1) = f_{act}(a_i(t), net_i(t), \theta_i),$$

  wobei θ_i der Schwellwert des Neurons i ist und f_{act} die Aktivierungsfunktion, die aus den angegebenen Parametern die Aktivierung der Zelle i bestimmt.

- Ausgabefunktion f_{out}. Die Ausgabe o_i der Zelle i wird durch eine sogenannte Ausgabefunktion aus der Aktivierung der Zelle bestimmt.

  $$o_i = f_{out}(a_i).$$

2. Verbindungsnetzwerk der Zellen. Ein neuronales Netz kann man sich als einen gewichteten, gerichteten Graphen vorstellen, wobei die Kanten dieses Graphen die gewichteten Verbindungen zwischen den Neuronen darstellen. Das Gewicht der Verbindung von Zelle i nach Zelle j bezeichnet man als w_{ij}. Die Matrix der Verbindungen aller Zellen (Gewichtsmatrix) wird mit W bezeichnet.

3. Propagierungsfunktion. Sie gibt an, wie sich die Netzeingabe eines Neurons aus den Ausgaben der Vorgängerneuronen und den Verbindungsgewichten berechnet. Dabei wird folgende Formel verwendet:

   $$net_j(t) = \sum_i o_i(t) w_{ij},$$

   wobei hier $net_j(t)$ die Netzeingabe von Zelle j ist, welche aus der Summe der Ausgaben o_i ihrer Vorgängerzellen multipliziert mit dem jeweilegen Gewicht w_{ij} zwischen Zelle i und Zelle j berechnet wird.

4. Lernregel. Die Lernregel stellt einen Algorithmus dar, gemäß dem das Netz lernt für eine vorgegebene Eingabe eine gewünschte Ausgabe zu produzieren. Dies wird meistens erreicht durch das geschickte Verändern der Gewichtsmatrix. Dabei werden dem neuronalen Netz viele Eingangsmuster zur Verfügung gestellt und es soll dann eine Minimierung des Fehlers zwischen Ausgabe und erwünschter Ausgabe erreicht werden.

2.3 Verschiedene Klassen von Neuronen und deren Funktionalität

Viele in der Praxis verwendete neuronale Netze haben Verbindungen nur in eine Richtung - von den Eingabeneuronen (input units) in Richtung der Ausgabeneuronen (output units). Man kann dann die Zellen nach ihrer Position im Netzwerk klassifizieren. Dieser sehr häufig verwendete Spezialfall von sog. *feedforward-Netzen* ist in Abb. 2.3.1 graphisch dargestellt.

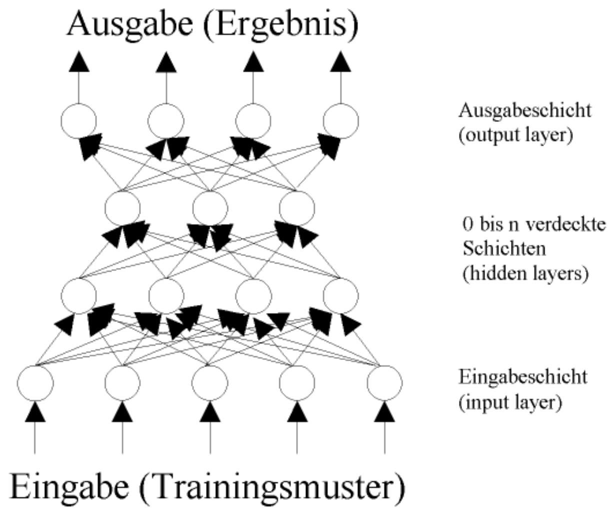


Abb. 2.3.1: Ein feedforward-Netz mit 3 Schichten von Verbindungen und 4 Zellschichten [?].

Es handelt sich hier um ein 3-stufiges Netz, d.h. um ein Netz mit 3 Schichten trainierbarer Neuronen. Zellen der Eingabeschicht leiten die Eingabe in das Netz weiter und werden deshalb als Eingabeneuronen (input units) bezeichnet, Zellen der Ausgabeschicht geben die Ausgabe des Netzes nach außen weiter und werden somit Ausgabeneuronen (output units) genannt. Zellen in der mittleren Schicht(en) dienen zur Informationsverarbeitung und werden als verdeckte Neuronen (hidden units) bezeichnet, weil sie für einen außenstehenden Betrachter nicht sichtbar sind bzw. keine direkte Ausgabe nach außen abgeben. Dabei ist zu beachten, daß die Eingabeneuronen keine Vorgängerneuronen haben und deshalb auch keine gewichteten Verbindungen von Vorgängern zu ihnen existieren. Genauso hat die Ausgabeschicht keine Nachfolgerneuronen und es existieren daher keine gewichteten Verbidungen zu Nachfolgern. Es existieren also folgende verschiedene Neuronentypen:

- *Eingabeneuronen:* Dieser Typ von Neuronen erhält die Eingabe von außen, die ist meistens ein Trainingsmuster, mit welchem das Netz trainiert werden soll. Die Eingabeneuronen geben diese Eingabe (input) dann 1:1 weiter an die Verarbeitungsneuronen, welche verdeckt sind (hidden layer). Dieser Neuronentyp hat also keine Verbindungsgewichte zu Vorgängern, da solche nicht existieren.

- *Verarbeitungsneuronen:* Sie erhalten Informationen (input net_j) von ihren Vorgängern, also entweder von den Eingabeneuronen oder wenn es sich um Verarbeitungsneuronen einer anderen Schicht als der zweiten handelt, von ihren Vorgängern, welche dann selbst Verarbeitungsneuronen sind. Bei diesem Neuronentyp existieren nun gewichtete Verbindungen, sowohl von den Vorgängern als auch zu den Nachfolgern. Diese Gewichte (Gewichtsmatrix W) werden dann im laufe der Trainingsphase des Netzes so modifiziert, daß irgendwann die tatsächliche Ausgabe des Netzes der erwünschten Ausgabe entspricht.

- *Ausgabeneuronen:* Sie erhalten auch Informationen von Vorgängern, geben diese aber im Normalfall nicht mehr an andere Neuronen weiter, sondern bilden aus diesen Informationen die Netzausgabe.

- *Kontextzellen:* Dieser Neuronentyp erhält seine Eingabe in vielen Fällen von den Ausgabeneuronen (oft auch von Neuronen der verdeckten Schichten) über 1:1-Verbindungen. Eine Kontextzelle nimmt nicht direkt am Trainingsprozess teil, verändert seine gewichteten Verbindungen also nicht in ihrer Stärke, sondern nimmt nur Informationen auf, welche dann wieder als Eingabe für das ganze Netz dienen. Es handelt sich hier also um eine Informationsspeicherung für spätere Verwendung.

- *On-Neuron:* Hier handelt es sich um ein Neuron, dessen Ausgabe immer 1 ist und welches mit allen anderen Neuronen verbunden ist, welche einen Schwellwert θ_j haben. Dabei entspricht das Gewicht w_{Oj} vom On-Neuron zu einem anderen Neuron j genau dem negativen Schwellwert, also $-\theta_j$.

2.4 Lernregeln in einem neuronalen Netz

Die Lernregel ist die interessanteste Komponente in einem neuronalen Netz, weil sie es erlaubt, daß ein Netz eine gegebene Aufgabe (weitgehend) selbstständig aus Beispielen lernt. Um also ein gegebenes Problem (Aufgabe) zu lösen, muß eine Lernregel bestehen. Theoretisch gibt es sehr viele Arten, wie ein neuronales Netz lernen kann, nämlich durch

1. Entwicklung neuer Verbindungen,
2. Löschen existierender Verbindungen,
3. Modifikation der Stärke w_{ij} von Verbindungen,
4. Modifikation des Schwellwertes von Neuronen,
5. Modifikation der Aktivierungs-, Propagierungs-, oder Ausgabefunktion,
6. Entwicklung neuer Zellen, oder
7. Löschen von Zellen.

Von diesen Alternativen, die man einzeln oder in Kombination verwenden kann, ist Alternative 3 die mit Abstand am häufigsten verwendete Art des Lernens. Die Methoden 1 und 2 können durch eine Veränderung von gewichteten Verbindungen realisiert werden. Dabei setzt man ein Gewicht w_i auf den Wert 0 um eine Verbindung zu "löschen" und um eine Verbindung zu "erzeugen", setzt man das betreffende Gewicht auf einen Wert ungleich Null. Dies setzt aber voraus, daß von Anfang an Verbindungen von jeder Zelle zu jeder anderen Zelle existieren. Dies ist aber bei den meisten Softwaresimulationen nicht der Fall, sodaß das Hinzufügen neuer Verbindungen nur mit einem sehr großen Aufwand realisierbar ist.

Im folgenden werden einige bekannte Lernregeln kurz vorgestellt.

2.4.1 Hebbsche Lernregel

Die Hebbsche Lernregel wurde in ihrer ersten Form bereits 1949 von Donald O. Hebb formuliert und ist die Grundlage für die meisten komplizierteren Lernregeln. Sie besagt folgendes:

> *Wenn Zelle j eine Eingabe von Zelle i erhält und beide gleichzeitig stark aktiviert sind, dann erhöhe das Gewicht w_{ij} (die Stärke der Verbindung von Zelle i nach Zelle j).*

Mathematisch angeschrieben sieht die Hebbsche Regel wie folgt aus:

$$\Delta w_{ij} = \eta o_i a_j$$

Dabei ist Δw_{ij} die Änderung des Gewichtes w_{ij}, η eine Konstante (Lernrate), o_i die Ausgabe der Vorgängerzelle i und a_j die Aktivierung der Nachfolgerzelle j. Sehr häufig findet die Hebbsche Regel Verwendung bei binären Aktivierungswerten. Dabei muß man aber beachten, daß sich in diesem Fall nur positive Gewichtsänderungen oder Null ergeben können. Deshalb verwendet man hier sehr häufig die binären Aktivierungen -1 und 1 anstelle von 0 und 1, dann liefert die Hebbsche Regel entweder eine Abschwächung der Verbindung (wenn die beiden Zellen i und j nicht übereinstimmen) oder eine Verstärkung der Vebindung (falls die beiden Zellen i und j übereinstimmen).

Die allgemeine Formel der Hebbschen Regel lautet mathematisch angeschrieben (nach [**?**])

$$\Delta w_{ij} = \eta h(o_i, w_{ij}) g(a_j, t_j)$$

Hier erkennt man, daß die Gewichtsänderung als ein Produkt aus zwei verschiedenen Funktionen definiert ist. Die Funktion $h(o_i, w_{ij}$ hat als Paramter die Ausgabe o_i der Vorgängerzelle und das Gewicht w_{ij} von der Vorgängerzelle i zur Zelle j. Die andere Funktion $g(a_j, t_j$ hat die Aktivierung a_j der Zelle und die erwartete Aktivierung t_j (teaching input) als Paramter.
Viele der bekannten Lernregeln sind Spezialisierungen der allgemeinen Hebbschen Regel.

2.4.2 Delta-Regel

Bei der Delta-Regel, auch Widrow-Hoff-Regel [**?**] genannt, ist die Gewichtsänderung direkt proportional zur Differenz δ_j der Aktuellen Aktivierung a_j und der erwarteten Aktivierung t_j (teaching input).

$$\Delta w_{ij} = \eta o_i (t_j - a_j) = \eta o_i \delta_j.$$

Häufig wird auch folgende Form der Formel verwendet:

$$\Delta w_{ij} = \eta o_i (t_j - o_j) = \eta o_i \delta_j,$$

wobei jetzt zum Überprüfen des Lernerfolges statt der erwarteten Aktivierung a_j die erwarte Ausgabe o_j vom teaching input abgezogen wird. Die Delta-Regel wird nur bei linearen Aktivierungsfunktionen bei einer einzigen Schicht trainierbarer Gewichte verwendet und ist somit ein Sonderfall der als nächste erläuterten Backpropagation-Regel.

2.4.3 Backpropagation-Regel

Backpropagation ist eine Verallgemeinerung der Delta-Regel für Netze mit mehr als einer Schicht trainierbarer Gewichte mit einer nichtlinearen Aktivierungsfunktion. Die Aktivierungsfunktion muß monoton steigend und differenzierbar sein.
Die Regel für Backpropagation lautet analog wie die Delta-Regel:

$$\Delta w_{ij} = \eta o_i \delta_j,$$

mit dem einzigen Unterschied, daß die Berechnung von δ_j etwas komplizierter ist:

Falls j einge Ausgabezelle ist gilt

$$\delta_j = f'_j(net_j)(t_j - o_j),$$

für den Fall, daß j eine verdeckte Zelle ist gilt jedoch:

$$\delta_j = f'_j(net_j) \sum_k (\delta_k w_{jk}).$$

Man beachte, daß hier der Summenindex k über alle direkten Nachfolger der aktuellen Zelle j läuft (dies erkennt man auch an der Indizierung des Gewichtes w_{jk}).

2.5 Verschiedene Klassen von neuronalen Netzen

Die Familie der neuronalen Netze läßt sich in zwei große Gruppen einteilen (siehe Abb. 2.5.1): Die eine Gruppe beinhaltet die Netze welche rückgekoppelt sind und die andere Gruppe die Netze, welche vorwärts gekoppelt sind. Bei vorwärts gekoppelten Netzen, auch als feedforward-Netze bezeichnet, sind die gewichteten Verbindungen nur in Richtung der AusgabeZelle verbunden, es gibt also keine rekursiven Gewichtsverbindungen. Bei rückgekoppelten Netzwerktopologien, auch als recurrent-Netze bekannt, existieren zusätzlich zu den vorwärts gerichteten Verbindungen auch Gewichtsverbindungen, welche Informationen wieder als Eingabe für eine Vorgängerzelle zu diesen zurückleiten. Normalerweise unterscheidet man also zwischen den folgenden 3 wichtigsten Gruppen (Klassen) von neuronalen Netzen:

2.5.1 Single-Layer Feedforward Netzwerke

Ein geschichtetes Netzwerk von Neuronen ist aus verschiedenen Schichten von Neuronen aufgebaut. Die einfachste Form ist ein einschichtig vorwärts gekoppeltes Netzwerk (single layer feedforward network [?]). Bei diesem Netztyp existiert lediglich eine Eingabeschicht mit ihren Eingabeneuronen und eine Ausgabeschicht mit Ausgabeneuronen, auf welche die Eingabe von den Eingabeneuronen projeziert wird.

2.5.2 Multilayer Feedforward Netzwerke

Dieser Netzwerktyp ist eine Erweiterung von single-layer feedforward-Netzwerken. Der Unterschied besteht darin, daß bei mehrschichtig vorwärts gekoppelten Netzen (multilayer

feedworward network) ein oder mehrere verdeckte Schichten (hidden layer) verwendet werden. Die Neuronen dieser Schichten werden dann verdeckte Neuronen (hidden neurons, hidden units) genannt. Durch das Hinzufügen solcher verdeckten Schichten ist das neuronale Netz nun in der Lage, komplexere Aufgaben zu lösen bzw. zu lernen. So werden nun die Informationen der Eingabeneuronen weitergegeben an die erste verdeckte Schicht, diese gibt die Informationen weiter an die nächste Schicht, bis schließlich die letzte verdeckte Schicht von Neuronen die Informationen an die Ausgabeneuronen in der Ausgabeschicht weiterreicht, welche dann das Ergebnis repräsentieren.
Bei einem allgemeinen feed-forward Netzwerk gibt es auch die Möglichkeit, daß ein Neuron der Schicht i mit einem Neuron der Schicht k verbunden ist und somit die Schicht j überspringt. Eine Verbindung, welche eine oder mehrere Schichten überspringt nennt man shortcut connection.
Ein Netz mit p Eingabeneuronen, h_1 Neuronen in der ersten verdeckten Schicht, h_2 Neuronen in der zweiten verdeckten Schicht und q Neuronen in der Ausgabeschicht wird zum Beispiel p-h_1-h_2-q Netzwerk genannt [?].

2.5.3 Netze mit Rückkopplungen

Netze mit Rückkopplungen unterteilt man meistens in die Klassen der Netze mit direkten Rückkopplungen (direct feedback), Netze mit indirekten Rückkopplungen (indirect feedback) und Netze mit Rückkopplungen innerhalb einer Schicht (lateral feedback) und vollständig verbundenen Netze.

1. Netze mit direkten Rückkopplungen (direct feedback)
   Diese Netze erlauben es, daß ein Neuron seine eigene Aktivierung über eine Verbindung von seinem Ausgang zu seinem Eingang verstärkt oder abschwächt. Diese Verbindungen bewirken oft, daß Neuronen die Grenzzustände ihrer Aktivierungen annehmen, weil sie sich selbst hemmen oder verstärken.

2. Netze mit indirekten Rückkopplungen (indirect feedback)
   Bei diesen Netzen gibt es Rückkopplungen von Neuronen höherer Ebenen zu Neuronen niederer Ebenen.

3. Netze mit Rückkopplungen innerhalb einer Schicht (lateral feedback)
   Netze mit Rückkopplungen innerhalb einer Schicht werden oft verwendet, wenn nur ein Neuron einer Gruppe aktiv werden soll. Jedes Neuron erhält hemmende (inhibitorische) Verbindungen zu anderen Neuronen und oft noch eine aktivierende (exzitatorische) Verbindung zu sich selbst. Das Neuron mit der stärksten Aktivierung (der Gewinner) hemmt dann die anderen Neuronen, daher heißt eine solche Topologie auch winner takes all-Netzwerk [?].

4. Vollständig verbundene Netze
   Vollständig verbundene Netze haben Verbindungen zwischen allen Neuronen. Sie sind insbesondere als Hopfield-Netze bekannt geworden. Bei diesen gibt es allerdings noch 2 Einschränkungen: die Verbindungsmatrix W muß symmetrisch sein und die Diagonale der Verbindungsmatrix darf nur Nullen enthalten.

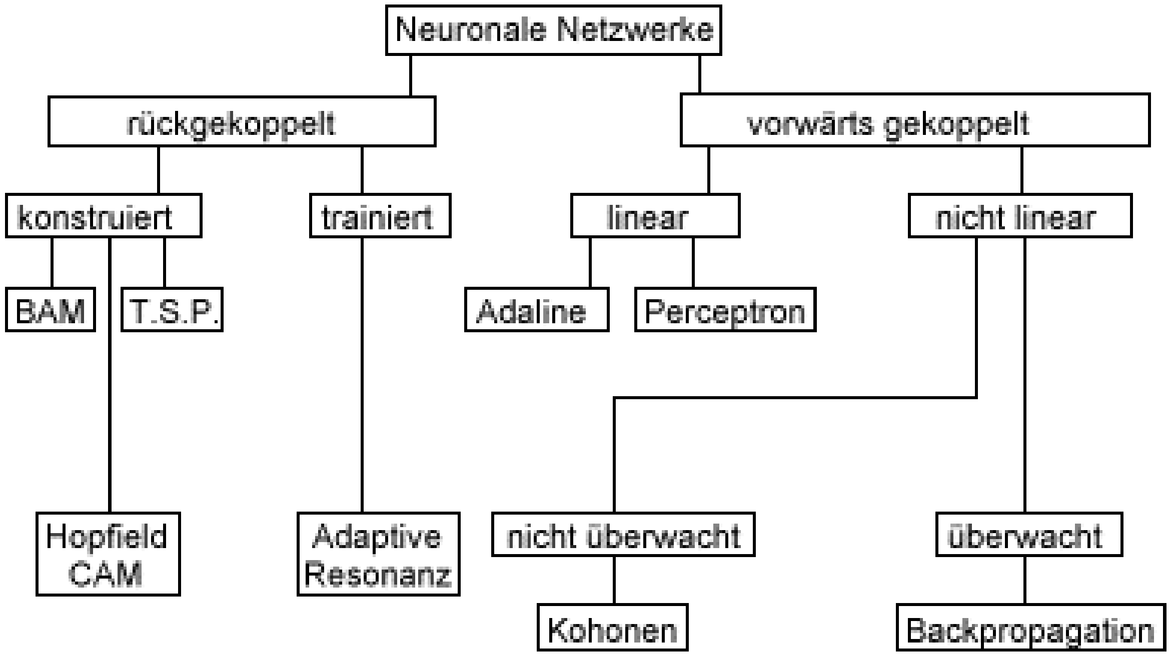


Abb. 2.5.1: Die Neuro-Netz-Familie untergliedert nach den Kategorien Kopplung, Lernmethode und Linearität [?].

Kapitel 3

Objekterkennung

3.1 Was ist Objekterkennung

Objekterkennung ist eine computergetützte Methode, um Objekte, welche auf einem digitalisierten Bild abgebildet sind, zu erkennen. Dabei wird versucht, im vorliegenden Bild relevante Merkmale hervorzuheben und vorhandene Störungen wie Bildrauschen abzuschwächen bzw. ganz zu entfernen.
Um diese Aufgaben zu bewältigen, gibt es die verschiedensten Algorithmen. Oft genügt es, einen einfachen Smoothing-Filter zu verwenden, um Rauschen zu entfernen. Jedoch kann es dann auch vorkommen, daß auch wichtige Bildelemente wie Kanten verwischt werden und somit nicht mehr richtig erkannt werden können. Darum verwendet man hier sehr oft mehrere Filter zusammen, welche je nach Aufgabenstellung mit den richtigen Parametern ein gutes Ergebnis liefern.
Zusätzlich verwendet man sehr oft neuronale Netze, welche mit bereits gelerntem Wissen die wichtigsten Bildelemente schneller und besser erkennen.

3.2 Erfassen von Bilddaten

Damit Objekte auf einem Bild erkannt werden können, muß ein Bild zuerst in eine digitale Form umgewandelt werden. Dazu gibt es mehrere Möglichkeiten (Scanner, Digitalkamera, ...). Die eingelesenen Daten werden dann in digitaler Form in einer Matrix m x n gespeichert, wobei m die Bildbreite und n die Bildhöhe ist. Bei einer Anzahl von c möglichen Farben im Bild benötigt man $log_2(c)$ Bits pro Pixel zum Speichern der Daten.

3.3 Bildaufbereitung

In vielen Fällen kommt es vor, daß der Benutzer die digitalen Rasterbilder gar nicht sieht (z.B. Mobiler Agent). Deren Qualität ist oft so schlecht, daß sie in Bildaufbereitungsschritten erst verbessert werden muß. Oft dient die Bildaufbereitung auch der Anpassung des Bildmaterials an die gegebene Problemstellung [?]. Dabei können radiometrische oder geometrische Eigenschaften des Bildes verbessert oder verändert werden.

Beispiele für radiometrische Verbesserungen:

- Korrektur
  - atmosphärische Einflüsse
  - richtungsabhängige Reflexion
  - photographische Einflüsse
- Verschärfung
  - Grauwertmanipulationen
  - Farbkodierung
  - Detailkontrasterhöhung
  - Filterung

3.3.1 Kontrastverbesserung

Wenn man ein digitales Rasterbild betrachtet, kann es anfangs sehr hilfreich sein, wenn man ein Histogramm zu Hilfe nimmt. Ein Histogramm ist die graphische Darstellung einer einfachen Grauwertstatistik über das ganze Bild.
Ein solches Histogramm wird nun verwendet um zum Beispiel den Kontrast des Bildes zu analysieren. So kann es bei digitalisierten Bildern sehr hilfreich sein, ein flaues Bild (schlechtes Kontrastverhältnis) zu verbessern, indem man den Kontrast des Bildes erhöht.
Wenn man annimmt, daß E das Eingangsbild und A das Ausgabebild ist, und weiters $E(x,y)$ der Bildpunkt von E an der Position x/y bzw. $A(x,y)$ der Bildpunkt von A an der Position x/y ist, dann berechnet man die neue Farbe des Bildpunktes $A(x,y)$ nach der folgenden Formel

$$A_{x,y} = \frac{A_{max} - A_{min}}{E_{max} - E_{min}}(E_{x,y} - E_{min}) + A_{min}$$

wobei A_{min} der kleinste und A_{max} der maximale im Ausgabebild vorkommende Helligkeitswert ist. Weiters ist E_{min} der kleinste und E_{max} der maximale im Eingabebild vorkommende Helligkeitswert.

3.4 Bildglättung

3.4.1 Smoothing

Ein weiteres Problem bei digitalisierten Bildern ist die Tatsache, daß sie fast immer Störungen beinhalten, welche den Prozeß der Objekterkennung negativ beeinflussen. Das Ziel eines Smoothing-Vorganges ist es nun, dieses sogenannte Bildrauschen zu minimieren oder gänzlich zu eliminieren.
Die einfachste Möglichkeit dies zu erreichen ist die lineare Mittelwertberechnung. Dabei werden alle Punkte des Eingangsbildes E durchlaufen und immer mit ihren acht Nachbarn

verknüpft. Die neue Farbe des Punktes an der jeweiligen Stelle x/y errechnet sich nach folgender Formel.

$$A_{x,y} = \frac{1}{9} \sum_{i=-1}^{1} \sum_{j=-1}^{1} E_{x+i,y+j}$$

Das Problem bei dieser einfachen Form des Smoothings ist, daß sich mit hoher Wahrscheinlichkeit neue Kanten und Artefakte (neue "Objekte") im Bild bilden.

3.4.2 Smoothing mit dem Gauß-Filter

Eine weitaus bessere Methode zum Glätten eines Bildes ist der sogenannte Gaußfilter. Dieser Filter ist ähnlich der Methode wie gerade beschrieben, jedoch werden hier auch die Entfernungen der Nachbarn in die Berechnungen miteinbezogen. Diese Form der Bildglättung verhindert, daß neue Artefakte und Kanten im Bild entstehen. Die Formel dazu lautet ähnlich wie oben

$$A(x,y) = 0.017 * \sum_{i=-2}^{2} \sum_{j=-2}^{2} E_{x+i,y+j} * G_{i,j}$$

wobei hier G die Matrix ist, welche immer über einen 5x5-Ausschnitt des Bildes gelegt wird. Die Matrix G ist dabei wie folgt definiert

$$G = \begin{pmatrix} 0 & 1 & 2 & 1 & 0 \\ 1 & 3 & 5 & 3 & 1 \\ 2 & 5 & 10 & 5 & 2 \\ 1 & 3 & 5 & 3 & 1 \\ 0 & 1 & 2 & 1 & 0 \end{pmatrix}$$

Beim Glätten werden nun alle Punkte des Eingangsbildes durchlaufen. Für jeden Punkt werden dann alle Farben in einem $5x5$-Ausschnitt addiert, wobei jeder Farbwert vor der Addition noch zusätzlich mit dem entsprechenden Wert der Gaußmatrix multipliziert wird.

3.5 Kantendetektion

Wenn das Bild nun eine bestmögliche Qualität erreicht hat (das Ideal, also ein total störungsfreies Bild, ist praktisch nicht möglich), kann nun zur Stufe der Kantendetektion übergegangen werden. In diesem Abschnitt wird versucht, Kanten im vorliegenden Bild hervorzuheben. Dabei hängt der Erfolg sehr stark vom verwendeten Algorithmus ab, denn es gibt (noch) keinen Algorithmus, welcher sich auf beliebige Bilder anwenden läßt und dabei immer optimale Ergebnisse liefert. Darum gibt es sehr viele verschiedene Algorithmen [?].

3.5.1 Sobel-Operator

Der Sobeloperator ist ein sogenanntes Gradientenverfahren. Dabei wird für jeden Bildpunkt $E(x,y)$ (Eingangsbild) ein neuer Wert berechnet, welcher sich aus den Farben der

8 Nachbarpunkten verknüpft mit einer $3x3$-Maske (Operator) errechnet. Die Formel dafür lautet

$$A(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} E_{x+i,y+j} * S_{i,j},$$

wobei $E(x,y)$ das Eingangsbild und $A(x,y)$ das Ausgangsbild. i und j entsprechen dem Offset zum Ausgangspixel (i=0 und j=0 ist Zielpixel) und S ist der verwendete Operator. Dabei gibt es zwei verschiedene Masken, eine um Kanten in x-Richtung hervorzuheben und eine um Kanten in y-Richtung hervorzuheben. Die Maske für vertikale Kantendetektion sieht wie folgt aus

$$S_{ver} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

und die Maske für horizontale Kantendetektion sieht folgendermaßen aus:

$$S_{hor} = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Werden diese Masken auf ein Bild angewandt, entsteht ein Bild, in welchem die Helligkeitswerte den Kantensteilheiten an den betreffenden Punkten entsprechen.

3.5.2 Prewitt-Operator

Der Prewittoperator funktioniert wie der Sobeloperator, lediglich die verwendeten Masken für die Bildoperation sind unterschiedlich. Die Maske für vertikale Kantendetektion sieht wie folgt aus

$$P_{ver} = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

und die Maske für horizontale Kantendetektion sieht folgendermaßen aus:

$$P_{hor} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

Wie auch beim Sobeloperator entsteht ein Bild, dessen Helligkeitswerte die Stärke der Kante an einer bestimmten Position im Bild entsprechen.

3.5.3 Marr-Hildreth Transformation

Wird dieser Algorithmus auf ein Bild angewandt, ensteht ein Bild mit negativen und positiven Werten. Dabei enstprechen diese Werte der Steigung einer Kante in einem Punkt $A(x,y)$. Die hier verwendete Maske entspricht dabei einem Gaußfilter, auf welchen der sogenannte Laplaceoperator angewandt wird.
Um nun aus diesen Daten Kanten zu extrahieren, geht man wie im folgenden Pseudo-Code

vor [?]:

for all x,y:
if (A(x,y)< −t **and** anyneighbourof(A(x,y))>t)
or (A(x,y)>t **and** anyneighbourof(A(x,y))< −t)
then A(x,y) is an edge pixel

Dabei entspricht $A(x, y)$ dem aktuellen Pixel und t einem Schwellwert.

3.6 Segmentierung

Bei der Segmentierung wird ein Bild in Regionen aufgeteilt, welche die gleichen Merkmale aufweisen. Es wird versucht, Objekte, welche sich auf dem Bild befinden, vom Hintergrund zu unterscheiden und verschiedenen Klassen zuzuordnen. Im Gegensatz zu den in den vorigen Abschnitten behandelten Algorithmen, ist die Segmentierung keine Bild-zu-Bild Tranformation, sondern im vereinfachten Sinn eine Extraktion von Wissen [?].

3.6.1 Grauwert Thresholding

Diese Methode ist die bekannteste, jedoch auch nur in wenigen Fällen anwendbar. Am besten funktioniert diese Methode bei Bildern, welche Objekte beinhalten, die sich klar vom Hintergrund abheben und außerdem eine homogene Helligkeit aufweisen.
Dabei wird jeder Pixel mit einem globalen Schwellwert (Threshold) verglichen. Ist der Wert des Pixels niedriger (höher) als der Schwellwert, wird dieser dem Objekt (Hintergrund) zugeordnet. In Folge entsteht ein binäres Bild. Dieses Bild kann dann weiterverarbeitet werden, so kann man zum Beispiel die Pixel zählen, welche sich in einem Objekt befinden. Daraus kann man dann weitere Schlüsse ziehen.
Die Schwierigkeit bei diesem Verfahren liegt darin, einen geeigneten Schwellwert zu finden. Am Besten nimmt man das Histogramm und sucht globale Maxima. Ein solches Maximum kennzeichnet also einen Farbwert, welcher im Bild am häufigsten vorkommt und wahrscheinlich ein Objekt darstellt.

3.6.2 Texturbasierende Segmentierung

Bilder, welche mit einer Kamera aufgenommen werden, können mit Grauwert Thresholding nur sehr schwer, wenn überhaupt richtig klassifiziert werden. Der Grund dafür ist, daß bei digitalisierten nur sehr selten wirklich homogene Flächen auftreten, und wenn dies doch der Fall sein sollte, dann sind diese Flächen nur sehr klein und für eine korrekte Segmentierung nicht brauchbar.
Darum werden bei der texturbasierenden Segmentierung nicht mehr nur die einzelnen Pixel in die Segmentierung einbezogen, sondern Regionen. Der Begriff der Textur ist dabei etwas schwierig zu definieren. Im allgemeinen kann man den Begriff der Textur in etwa wie folgt definieren:

> *Eine Textur ist eine Variation in einer Fläche, welche sich wiederholt* [?].

Die Beschaffenheit einer Textur kann dabei durch Begriffe wie Feinheit, Granularität, Richtung und Dichte beschrieben werden. Um nun eine Textur möglichst eindeutig klassifizieren zu können, versucht man dieser eine eindeutige Nummer zuzuordnen.

Hurst Koeffizient

Der Hurst Koeffizient ist eine Nummer, welche sich aus einer oktagonalen Umgebung um den Ausgangspunkt errechnen läßt. Dabei geht man folgendermaßen vor.

1. Es wird ein Kreis in einem Punkt erzeugt (der Punkt, welcher einer Kategorie zugeordnet werden soll) und der Radius dieses Kreises wird solange vergrößert, bis alle gewünschten Nachbarpixel erreicht sind.

2. Für jeden Radius nimmt man nun alle Pixel, welche auf dem Kreis liegen und sucht den kleinsten und den größten vorkommden Helligkeitswert. Die Differenz dieser Werte ergibt dann einen bestimmten Bereich.

3. Dann erhält man den Koeffizienten, indem man den Logarithmus der Helligkeit durch den Logarithmus des Radius dividiert.

In der Praxis verwendet man für die verwendeten Kreise keine Kreisformel, sondern eine Matrix mit bereits vorberechneten Werten. Dies hat zwei Gründe:

1. Die zeitaufwendige Berechnung der Kreisbahn entfällt.

2. Es wird verhindert, daß sich Pixel von verschiedenen Kreisen überlappen.

Die dafür verwendete Matrix sieht wie folgt aus:

$$H = \begin{pmatrix} & & h & g & h & & \\ & f & e & d & e & f & \\ h & e & c & b & c & e & h \\ g & d & b & a & b & d & g \\ h & e & c & b & c & e & h \\ & f & e & d & e & f & \\ & & h & g & h & & \end{pmatrix}$$

In diesem sogenannten Hurst-Opeartor gibt es sieben verschiedene Regionen, welche in der Matrix durch die Buchstaben a bis h gekennzeichnet sind.
In jeder dieser Regionen, werden nun der dunkelste und der hellste Pixel gesucht.

Die Formel für die Berechnung des Hurst Koeffizienten lautet wie folgt:

$$slope = \frac{n * \sum log(range) * log(distance) - \sum log(distance) * \sum log(range)}{n * \sum log(distance^2) - \sum log(distance^2)}$$

Dabei werden folgende Variablen verwendet:

- *slope* ist der berechnete Koeffizient, welcher einen texturierten Bereich eindeutig klassifiziert

- n ist die Anzahl der verschiedenen Regionen (im gegebenen Fall ist n 7)

- *range* ist die Differenz zwischen dem hellsten und dem dunkelsten Pixel in der aktuellen Region
- *distance* ist die Distanz vom zentralen Pixel zur aktuellen Region

3.6.3 Segmentierung mit Region Growing

Beim Region Growing wird ähnlich wie beim Thresholding vorgegangen, jedoch geht man hier nicht von einem Histogramm aus, sondern direkt von den Bilddaten. Zuerst muß definiert werden, was eine Region ist. Gonzalez und Woods sagten, daß eine Region folgende Eigenschaften haben muß:

- Jeder Pixel im Bild muß nach der Segmentierung einer bestimmtem Region zugehören. Es sollten also keine nicht-klassifizierten Pixel übrigbleibel.
- Alle Punkte in einer Region müssen miteinander verbunden sein. Das bedeutet also, daß jeder Pixel durch einen bestimmten Algorithmus von einem anderen Pixel in der Region erreichbar sein muß.
- Die Regionen dürfen sich nicht überlappen und jeder Pixel sollte nur einer Region zugeordnet sein.
- Alle Pixel in einer Region müssen eine gleiche Eigenschaft P_i aufweisen.
- Zwei Regionen sind unterschiedlich, wenn sie unterschiedliche Eigenschaften P_i aufweisen.

Region Growing Algorithmus

Um verschiedene Regionen in einem Bild zu finden, wählt man zuerst einige zufällige Punkte in diesem Bild. Dann durchsucht man, ausgehend von diesem sogenannten Seed-Punkten, die Umgebung und vergleicht die umgebenden Pixel mit dem aktuellen Pixel. Ist die Differenz deren Helligkeitswerten unter einem bestimmten Schwellwert, dann wird der Pixel der Umgebung zur aktuellen Region hinzugefügt. Dies wird dann solange rekursiv wiederholt, bis alle im Bild befindlichen Pixel einer bestimmten Region zugeordnet sind.

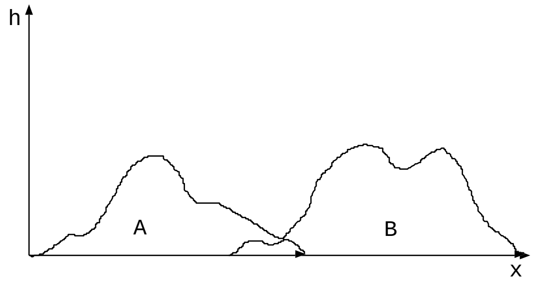
3.6.4 Kanten und Segmentierung

Am einfachsten erscheint es, geschlossene Formen zu finden. Dann ist alles in dieser Form im Segment, und alles außerhalb dieser Form in einem anderen Segment. Jedoch können geschlossene Formen nur durch komplexe Verfahren gefunden werden, wenn überhaupt.

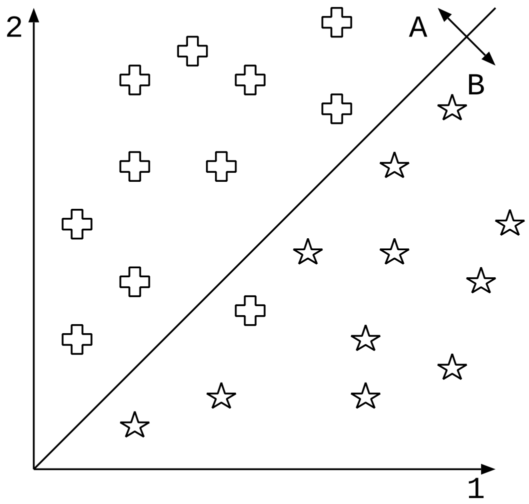
3.7 Feature Extraction

3.7.1 Allgemeines

Um nun die extrahierten Segmente klassifizieren zu können, muß man zuerst eine Anzahl relevanter Parameter in den sog. Feature Space übertragen. Diese Parameter dienen dann als Eingabewerte für das neuronale Netz. Der Feature Space besteht aus n-Dimensionen, wobei jede Dimension für einen Parameter steht. Die Anzahl und die Art der Features muß individuell für jede Aufgabenstellung gewählt werden.



In dieser Graphik wird ein 1 dimensionaler Feature Space zur Klassifizierung zwischen A und B benutzt. h ist die Anzahl der Segmente, die A oder B zugehören. Wie man gut erkennen kann, befindet sich in der Mitte ein Streubereich, welcher zu falsch klassifizierten Objekten führt. Ein besseres Ergebnis bekommt man, wenn man noch eine weitere Dimension einführt. Diese soll so gewählt werden, daß die Streubereiche möglichst voneinander unabhängig sind.



Nun, da die 2. Dimension benutzt wird, erhöht sich die Trefferquote der Klassifizierung.
Dieses Beispiel, es handelt sich ja nur um die Erkennung von 2 Klassen mit einem (zwei) relativ gut abgegrenzten Features, ist sehr einfach.
Für die Objekterkennung werden viel komplexere Systeme benötigt, da zum Einen die Anzahl der Klassen höher ist und zum Anderen die Segmente durch die physikalischen Eigenschaften und die Geometrie oft stark verfälscht werden. Somit haben die meisten Features einen breiten Streuschlauch zwischen den einzelnen Klassen, oft kann in einem Feature gar nicht zwischen 2 Klassen unterschieden werden.
Es gibt zwei Möglichkeiten Features zu verbinden:

- Wie im obrigen Beispiel können sie in einen n-Dimensionalen Raum übertragen werden, in dem dann die Klassifizierung anhand von Abgrenzung der einzelnen Cluster stattfindet.

- Aber gerade wenn Features nicht alle Klassen segmentieren können ist es oft praktikabel, mehrere Stufen der Klassenerkennung durchzuführen. Dabei wird zuerst in Oberklassen, die sehr sicher von einander getrennt werden können geteilt. Diese Oberklassen werden dann noch in ihre eigentlichen Klassen unterteilt. Dieses Modell würde sich auch gut in ein verteiltes System implementieren lassen, da hier die Rechenleistung gut aufgeteilt werden kann.

3.7.2 Feature Extraction in der Bildverarbeitung

Betrachten wir nun die verschiedenen Features, die aus einem gescannten Bild extrahiert werden können.

Helligkeitswerte

Wenn die verschiedenen zu klassifizierenden Objekte eine unterschiedliche Helligkeit haben, bietet sich diese als Eigenschaft des Objektes an. Als Feature würde sich die mittlere Helligkeit des Objektes eignen:

$$h_s = \frac{1}{s} \cdot \sum s_{x,y}$$

Benutzte Variablen:

- h_s die berechnete mittlere Helligkeit des Segmentes.

- s die Größe des Segmentes (Anzahl Pixel).

- $s_{x,y}$ Ein Pixel des Segments mit den Koordinaten x, y. Das Pixel muß natürlich von dem Originalbild kommen, und nicht etwa von einem bereits überarbeitetem Bild, welches verfälschte Helligkeiten hat.

Zu beachten ist hierbei aber auch noch, daß sich die Illumination des Objektes bedingt durch die Veränderung der geometrischen Anordnung stark ändern kann (Schatten, globale Helligkeitsänderungen). Es ergibt sich also ein großer Streubereich, welcher durch ein geeignetes Netz bearbeitet werden sollte.
Dieses Feature eignet sich aber nur für relativ einfache Umgebungen, da bei einer komplexeren Umwelt die Helligkeit vieler Objekte gleich oder zumindest sehr ähnlich ist. Nichts destro trotz kann man eine grobe Klassifizierung damit vornehmen. Durch die Einfachheit der Berechnung ist es leicht möglich dieses Feature in ein System einzubinden.

Seitenverhältnis des Segments

Ein weiteres Feature ist das Seitenverhältnis des Segments $v = l/b$. Hierbei muß bei nicht rechteckigen Segmenten ein Rechteck berechnet werden, das die Form des Segmentes möglichst gut umschließt. Danach wird das Seitenverhältnis dieses Rechteckes benutzt. Hauptsächlich verwendet wird dieses Feature bei der Texterkennung. Es sollte sich aber auch auf Gebiete der Bilderkennung übertragen lassen. Zu beachten in 3 dimensionalen Räumen ist, daß ein Objekt von verschiedenen Blickwinkeln aus andere Seitenverhältnisse besitzen. Dieses Feature eignet sich besonders für genau abgeschlossene und definierte Umgebungen und zur Erkennung von Objekten, die entweder nur von einer Seite betrachtet werden, oder die von allen Seiten ähnliche Abbildungen liefern (zB eine Kugel).

Segmentgröße

Da bis jetzt keine Information über die größe eines Segmentes verfügbar ist, führen wir das Feature Segmentgröße ein. Dies ist vor allem wichtig, um Störfelder im Bild, welche als Segment extrahiert wurden zu eleminieren, da Störfelder meist nur aus ein paar Pixeln bestehen und somit weit ausserhalb des Bereiches der zu Klassifizierenden Features liegt. Dieses Feature beschreibt die Anzahl der Pixel, die in einem Segment enthalten sind, und liefert so einen ganzzahligen von der Auflösung abhängigen Wert. Zu beachten ist natürlich, daß sich die Größe eines Objektes mit der Entfernung ändert. Dieses Feature könnte auch dazu benutzt werden. Bereits in nahe und ferne Objekte zu unterteilen.

Texture Extraction

Der bereits erwähnte Hurst Koeffizient eignet sich nicht nur zur Segmentierung, sondern bietet auch ein brauchbares Feature des extrahierten Segments. Da nun eine Matrix von Werten existiert, die die gleiche Dimension hat, wie das Segment, muß zuerst noch das arithmetische Mittel der Koeffizienten berechnet werden. Diese sollten ohnehin (bis auf Störfunktionen) identisch sein.

3.8 Objekterkennung mit KNN

In diesem Abschnitt werden nun zwei verschiedene Netzwerktopologien gezeigt, welche sich für die Objekterkennung eignen.

3.8.1 Das Multilayer-Perceptron Netz (MLP)

Für jedes beschreibende Merkmal wird ein Eingabeneuron verwendet. Pro Klasse wird ein Ausgabeneuron eingesetzt. Während des Lernvorgangs werden dem Netz die Merkmalsvektoren von Pixeln, deren Klassen bekannt sind, präsentiert. Als Soll-Ausgabewert wird an dem Ausgabeneuron, welches für eine bestimmte Klasse zuständig ist, eine Eins angelegt, an allen anderen Ausgabeneuronen der Wert Null. m MLP modelliert werden soll, desto mehr verborgene Schichten werden benötigt. Da das vorhandene Datenmaterial häufig nicht genügend Lernstichproben zur Verfügung stellt, kann die Generalisierungsleistung bei diesen Klassifikatoren unter Umständen herabgesetzt sein. Hier hilft man sich mit sogenannten Pruning-Verfahren, die während oder nach dem Training entweder Neuronen oder Gewichte aus dem bereits trainierten Netz “kappen”.

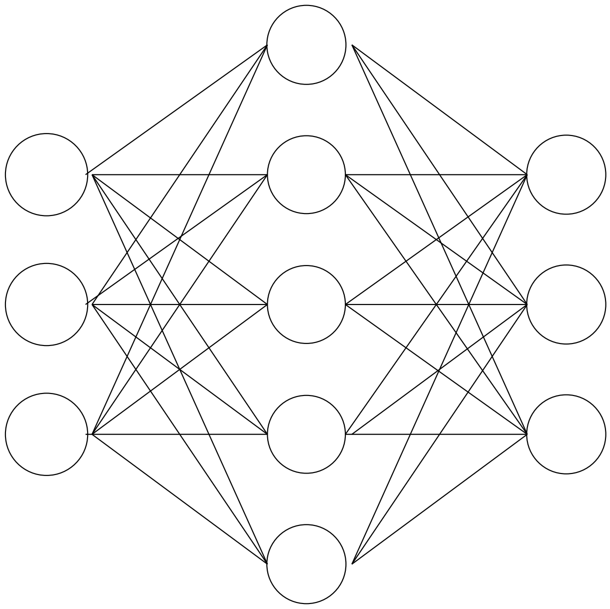


Abb. 3.8.1: Ein MLP mit 3 Eingangsneuronen, 5 Neuronen im Hidden Layer und 3 Ausgangsneuronen

Eine Faustregel besagt, daß die Anzahl der Lernstichproben je nach Aufgabenstellung etwa doppelt bis zehnmal so hoch sein sollte wie die Anzahl der vorhandenen Gewichtsverbindungen im Netz, damit das Netz während des Lernvorgangs nicht die Stichproben “auswendig” lernt, sondern die gewünschte Abbildungsvorschrift findet.
Um diesen Effekt zu vermeiden, wird beim Training des Netzes außerdem das sogenannte Cross-Validation-Verfahren eingesetzt. Dabei wird das Training selbst mit einer sogenannten Trainingsmenge von bekannten Beispielvektoren aus dem Merkmalsraum vorgenommen. Das Ergebnis des Trainings wird jedoch anhand einer separaten, nicht gelernten Testmenge überprüft. Der Lernalgorithmus, z.B. Standard-Backpropagation, wird so lange auf den Daten der Trainingsmenge angewendet, bis der Ausgabefehler bei Anlegen der Daten aus der Testmenge genügend klein ist.
MLP-Klassifikatoren werden häufig erfolgreich bei Klassifikationsproblemen eingesetzt, in denen die Klassen im Merkmalsraum nicht mehr ohne weiteres linear trennbar sind. Je komplexer jedoch die Trennungsfunktion aussieht, die mit dem MLP modelliert werden soll, desto mehr verborgene Schichten werden benötigt. Da das vorhandene Datenmaterial häufig nicht genügend Lernstichproben zur Verfügung stellt, kann die Generalisierungsleistung bei diesen Klassifikatoren unter Umständen herabgesetzt sein. Hier hilft man sich mit sogenannten Pruning-Verfahren, die während oder nach dem Training entweder Neuronen oder Gewichte aus dem bereits trainierten Netz “kappen”.

3.8.2 Das Kohonennetz

Dieses Netz wurde von Teuvo Kohonen entwickelt und wird auch selbstorganisierendes Merkmalsebenen-Netz (Selforganizing Feature Map SOFM) oder Vektor-Klassifizierer genannt. Es besteht aus einer Eingangsschicht und einer quadratischen Ausgangsmatrix. Neuronen aus der Eingangsschicht sind vollständig mittels gewichteter Verbindungen mit den Ausgangsneuronen in der Matrix verbunden. Das Netzwerk wird mittels unüberwach-

tem Lernens trainiert.
Wenn nun Trainingsdaten an den Netzeingangsneuronen anliegen, werden die Werte der Ausgangsneuronen berechnet und untereinander verglichen, um das Gewinnerneuron zu finden, welches die stärkste Ausgabe produziert. Wenn das Gewinnerneuron gefunden worden ist, werden die Gewichte bzgl. des Gewinnerneurons sowie dessen Nachbarn abgestimmt. Die Nachbarn des Gewinnerneurons sind definert als ein quadratischer Bereich, in dessen Zentrum sich das Gewinnerneuron befindet. Die Größe des Nachbarschaftsbereiches wird während des lernens sukzessive veringert. Dieser Netztypus lernt die Eingangsmuster über die Ebene zu verteilen, so daß am Ende des Lernverfahrens jeweils ein Neuron in der Ebene eine Klasse von Mustern identifiziert.

Die Netzwerkarchitektur

Das Kohonen Netzwerk besteht wie oben beschrieben aus einer Eingangsschicht, an die die Features des vorherigen Prozeßes eingegeben werden, und einer Ausgabeschicht, bestehend aus einer quadratischen Matrix aus Neuronen. Da die Quallität der Klassifizierung von der Größe dieser Matrix abhängt, sollte diese relativ groß sein. Die Nachbarschaftsmatrix sollte ungefähr halb so groß als die Ausgabematrix initialisert werden (Sie wird während des Lernens sukzessive verkleinert). Die Anzahl der Lernzyklen sollte ca. das 2- bis 10-fache der Anzahle der Eingabeneuronen betragen.

Bei der Klassifikation berechnet nun jedes Ausgabeneuron den euklidischen Abstand seines Gewichtsvektors zum Merkmalsvektor, der an der Eingabeschicht anliegt. Dasjenige Ausgabeneuron, welches den geringsten Abstand berechnet, ist "zuständig" für den Eingabevektor und feuert eine "1". Alle anderen geben den Wert "0" aus. In der Lernphase werden ebenfalls die euklidischen Abstände der Gewichtsvektoren zu den angelegten Eingabevektoren berechnet. Der Gewichtsvektor desjenigen Ausgabeneurons mit dem kleinsten Abstand zum Eingabevektor wird so modifiziert, daß er in die Richtung des Eingabevektors "gezogen" wird. Außerdem werden die Gewichtsvektoren in einer vorher definierten Nachbarschaft um das Gewinner-Neuron in gleicher Weise modifiziert:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)[m_i(t) - w_{ij}(t)]$$

Hierbei ist $\eta(t)$ der Lernfaktor zum Zeitpunkt t. Sowohl die Nachbarschaft als auch der Lernfaktor werden in der Regel während des Trainings von grosen Werten auf kleine Werte heruntergesetzt.

Nach diesem Training haben sich die Gewichte im Netzwerk stabilisiert haben und es kann mit der Beschriftung (supervised labeling) der Ausgabeneuronen begonnen werden. Dabei wird ein bekanntes Element zur Klassifizierung an das Netzwerk übergeben und das Ausgabeneuron mit der stärksten Ausgabe beschriftet.

Durch die Berücksichtigung der Nachbarschaftsbeziehung wird erreicht, daß Punktwolken, die im Merkmalsraum nebeneinander liegen, in der Ausgabeschicht von ebenfalls benachbarten Ausgabeneuronen dedektiert werden. Da eine Klasse im Merkmalsraum auch durch mehrere Punktwolken repräsentiert werden kann, wird durch die Anzahl der Ausgabeneuronen die Maximalanzahl der detektierbaren Klassen festgelegt. Es können aber durchaus mehrere Ausgabeneuronen für die gleiche Klasse zuständig sein. [?]

Kapitel 4

Projekt EMMA

In diesem Kapitel soll nun das in den vorigen Kapiteln erarbeitete Wissen praktisch angewandt werden. Dazu wurde eine Projektidee ausgearbeitet, welche eine Methode demonstrieren soll, um mit dem bereits eingangs erwähnten mobilen Agenten EMMA Objekte in einer stark vereinfachten Umgebung zu erkennen. Dabei wird ein Bild (von EMMA aufgenommen) verwendet, an welchem die verschiedenen Schritte der Objekterkennung verdeutlicht werden sollen:

1. Um ein Bild analysieren zu können, muß dieses zuerst in eine digitale Form überführt werden. Dabei verwendet EMMA eine Bildauflösung von $80x60$ Bildpunkten mit 64 Graustufen.

2. In diesem Schritt (Preprocessing) wird das nun in digitaler Form vorliegende Bild für die weitere Objekterkennung vorbereitet. Dabei wird versucht, etwaige Bildstörungen (wie etwa Bildrauschen) abzuschwächen oder ganz zu entfernen. Dazu verwendet man sehr oft den sogenannten Gaußfilter. Dieser verwischt nun das Bild, ohne neue Kanten zu bilden (wie dies bei einigen anderen Smoothing-Algorithmen der Fall wäre).

3. In diesem Schritt wird nun mit Hilfe von geeigneten Algorithmen versucht, das nun in geglätteter Form vorliegende Bild zu segmentieren. Um dies zu bewerkstelligen, werden zwei verschiedene Methoden betrachtet:

   - Die erste Möglichkeit ist, mit Hilfe eines Gradientenverfahrens die Kanten hervorzuheben, und nachträglich mit einem geeigneten Schwellwertverfahren (Thresholding) das Bild zu binärisiert. Sodann hat man ein Bild vorliegen, welches nur mehr Kanten enthält. Dann kann man die kantenbasierende Segmentation starten.
   - Die zweite Methode geht einen anderen Weg. Hier werden die Kanten nicht extra hervorgehoben, sondern es werden direkt aus den Bild die Segmente extrahiert. Dazu wird sehr oft das sogenannte "Region Growing" verwendet.

4. Nun wird versucht, die im vorigen Schritt erhaltenen Segment in einen Featurespace zu überführen. In unserem Fall von EMMA stehen mehrere Merkmale zur Verfügung, welche die in der Szene vorkommende Objekte diskriminieren:

   - Helligkeit und Textur des Segmentes
   - Form des Segmentes (rund, eckig,...)

5. Der letzte Schritt ist dann, die extrahierten Features an ein Kohonennetz weiterzugeben. Dabei hat dieses Kohonennetz genausoviele Eingabeneuronen, wie es unterschiedliche Merkmale bei den Segmenten gibt. Nach einer bestimmten Anzahl von Lernschritten, ist dieses Netz dann in der Lage Objekte auf einem Eingabebild zu klassifizieren.

Um die Arbeitsweise der verschiedenen Filter und Algorithmen zu zeigen, werden im Folgenden die Schritte der Objekterkennung anhand von Bildbeispielen gezeigt. Dabei ist das erste Bild ein künstliches Bild, welches mit einem Zeichenprogramm gezeichnet wurde. Das zweite Bild hingegen wurde direkt von EMMA aufgenommen.

4.1 Objekterkennung bei einem künstlichen Bild

Dieses künstlichen Bild wurde mit einem Zeichenprogramm erstellt. Es stellt einfache geometrische Formen in verschiedenen Farben (hier: Grauwerte) dar.

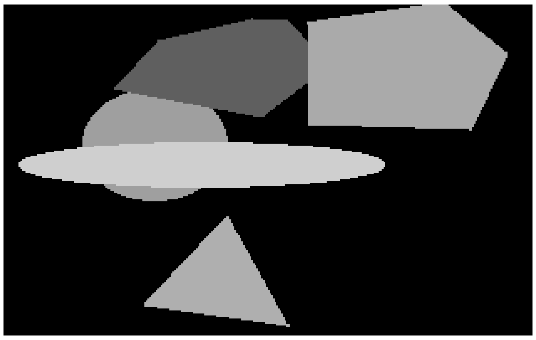


Abb. 4.1.1: Einfache geometrische, verschiedenfarbige Formen.

Dieses Bild enthält keine Bildstörungen, also benötigt man hier keinen Smoothing-Filter und kann gleich zur Kantendetektion übergehen.

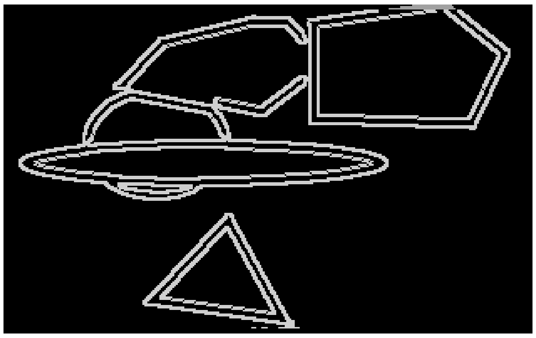


Abb. 4.1.2: Marr Hildreth.

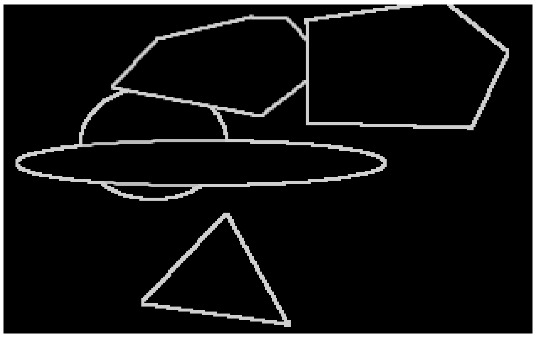


Abb. 4.1.3: Sobel Operator.

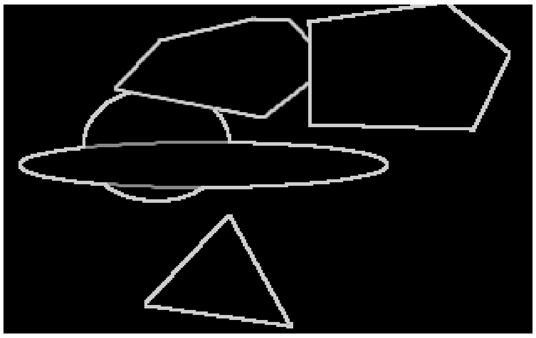


Abb. 4.1.4: Prewitt Operator.

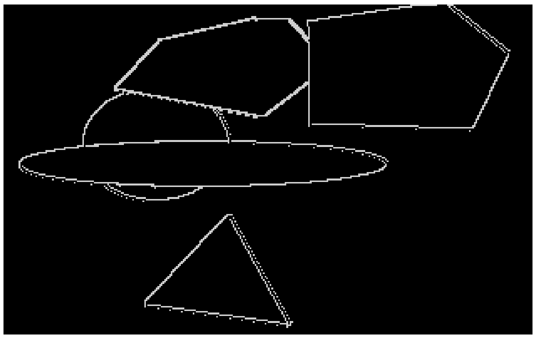


Abb. 4.1.5: Prewitt Thresholded.

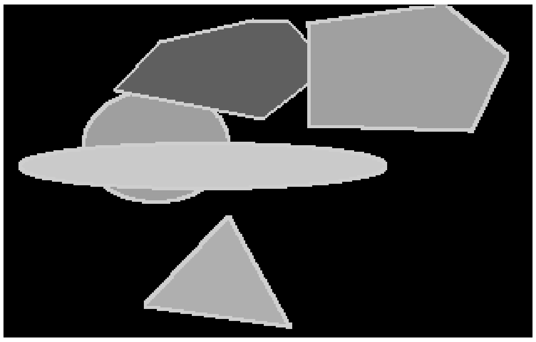


Abb. 4.1.6: Semikante.

4.2 Objekterkennung bei einem digitalisierten Bild

Dieses Bild wurde im Gegensatz zum vorigen Beispiel mit einer CCD-Kamera digitalisiert. Die Szene auf dem Bild enthält einen Hintergrund, einen Ball und einen Holzboden.

Abb. 4.2.1: Eine von EMMA aufgenommene Szene.

Hier erkennt man sofort, daß dieses Bild im Gegensatz zu Bild 4.1.1 Störungen in Form von Rauschen enthält. Hier ist es also notwendig das Bild zu glätten. Dazu wurde hier der Gaußfilter verwendet, welcher der Entstehung neuer Artefakte vorbeugen soll.

Abb. 4.2.2: Das Eingangsbild nach der Glättung.

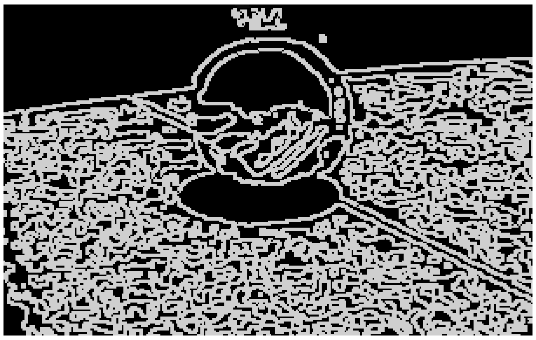


Abb. 4.2.3: Der Marr-Hildreth Filter ist hier nicht zu gebrauchen, da er ein Liniengewirr erzeugt.

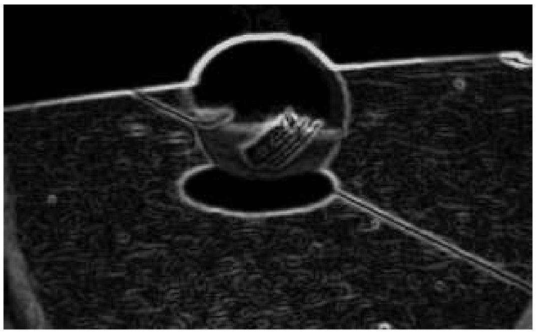


Abb. 4.2.4: Prewitt.

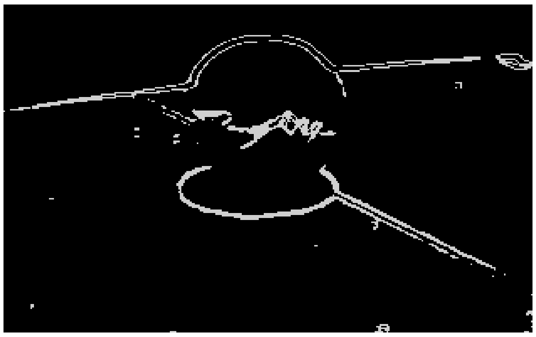


Abb. 4.2.5: Prewitt und nachfolgendes Thresholding.

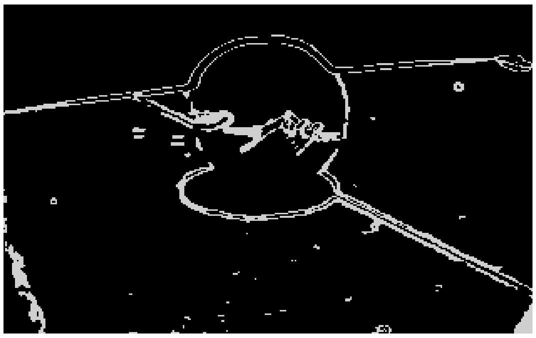


Abb. 4.2.6: Sobel und nachfolgendes Thresholding.

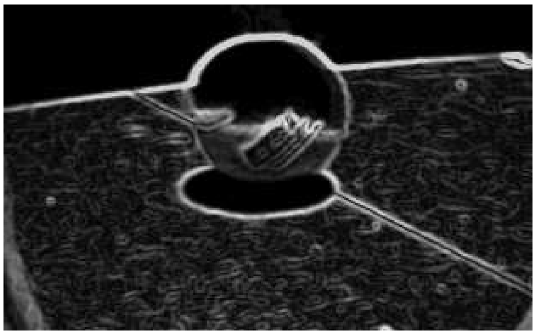


Abb. 4.2.7: Der Sobel Operator angewandt auf das Eingangsbild.

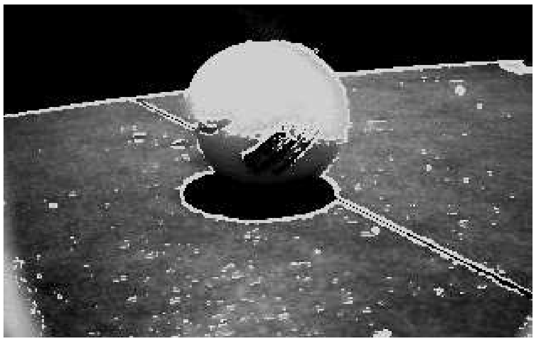


Abb. 4.2.8: Semikante (Sobeloperator mit darauffolgendem, angepaßtem Thresholding über das Originalbild gelegt).

4.3 Feature Extraction

Aus den erhaltenen Segmenten des Bildes müssen nun die Features extrahiert werden, um damit das neuronale Netz zu speisen. Für die einfache Umgebung und in Hinsicht auf die beschränkten Resourcen von EMMA wurden folgende vier Features gewählt:

- *Helligkeitswert*
- *Längen/Breitenverhältnis*
- *Segmentgröße*
- *Hurst Koeffizient*

Der so entstehende vierdimensionale Raum wird zur Klassifizierung des Segmentes mittels eines neuronalen Netzes verwendet. Aufgrund der scharfen Abgrenzung von Längen/Breitenverhältnis und Hurst Koeffizient, sowie der starken Helligkeitsunterschiede zwischen Vorder- und Hintergrund wird mit diesem Feature Space eine sichere und zuverlässige Klassifizierung möglich sein.

4.3.1 Feature Helligkeit

Das arithmetische Mittel des Helligkeitswertes eines Bildes eignet sich vorallem, die Restklasse abzugrenzen (Da der Hintergrund i.a. dunkler ist als die drei zu Klassifizierenden Objekte).

4.3.2 Feature Längen/Breitenverhältnis

Dieses Feature liefert eine relativ saubere Abgrenzung der drei Klassen:

- *Tor:* Wertebereich bei etwa 0.3...0.7
- *Ball:* Wertebereich bei etwa 0.9...1.2
- *Gegner:* Wertebereich bei etwa 1.7...2.1

4.3.3 Segmentgröße

Um kleine Segmente die durch Störungen, Spiegelungen oder Farbkontraste im Hintergrund entstehen zu eleminieren, untersuchen wir jedes Segment noch auf seine Größe. Da jedes der drei zu klassifizierenden Objekte mindestens 350 Pixel besitzen, kann so sichergestellt werden, daß keine Fehler irrtümlich als Objekte klassifiziert werden.

4.3.4 Hurst Koeffizient

Als letztes Feature wird der bereits berechnete Hurst Koeffizient verwendet. Er liefert eine Repräsentation der Objektoberfläche und eignet sich somit, die drei Objekte, welche alle eine differente Oberfläche besitzen, zu unterscheiden.

4.4 Neuronales Netz

Zur Klassifizierung der 3 verschiedenen Objekte in der EMMA-Szene werden die oben genannten Features nun in ein Neuronales Netz, genauer in ein Kohonen-Netz gespeist. Das Netzwerk wurde zuerst mit 144 Ausgangsneuronen realisiert, da sich aber nach genügend langer Lernphase keine eindeutigen Klassen ausbildeten, wurde die Ausgangsmatrix auf 4x4 = 16 Ausgangsneuronen beschränkt.
Dieses Netzwerk besitzt die Eingangsschicht bestehend aus 4 Neuronen, jeweils für ein Feature und eine quadratische Ausgangsmatrix, bestehend aus 16 Ausgangsneuronen. Zum Training sollten ca. 400 Lernzyklen ($10 \cdot Eingangsneuronen$)ausreichen, damit sich die Gewichte stabilisieren. Nach dieser Lernphase kann nun mit der Beschriftung der Ausgangsneuronen begonnen werden, dabei kann man zum Beispiel das Netz mit Features, die bei

der Vorverarbeitung eines Bildes mit einem Ball entstehen würden, speisen und anschließend das Ausgangsneuron, das die stärkste Ausgabe liefert mit B beschriten. Natürlich werden in der Ausgangsmatrix Bereiche entstehen, die jeweils für eine Objektklasse gelten.

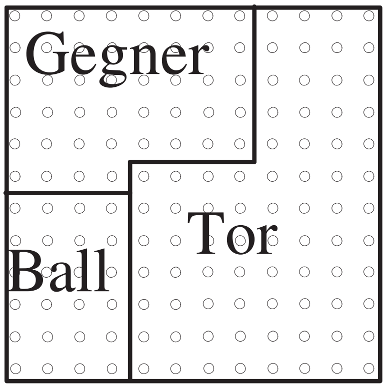


Abb. 4.4.1: Aufteilung der Ausgangsmatrix (144 Neuronen) auf die Objektklassen. Man sieht deutlich die Bereiche, auf die sich die Klassen aufteielen

Anhang A

Protokolle

A.1 10. März 1999

Zeit: 13:00 - 13:30
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, T. Gangl, M. Eckschlager und M. Liedlgruber

Behandelte Punkte:

- Einführung in die Problematik
- Besprechung der Formalitäten
- Hinweis auf weitere Quellen zum Thema

Aufgabenstellungen bis zum 17. März 1999

- Einführung zum Thema "Neuronale Netze" lesen
- Einleitung und Grundgerüst des Projektdokumentes erstellen

A.2 17. März 1999

Zeit: 13:30 - 13:45
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, T. Gangl und M. Liedlgruber

Behandelte Punkte:

- Korrektur der bereits abgegebenen Einleitung
- Besprechung der weiteren Vorgehensweise
- Festlegung des Formats des Dokuments

Aufgabenstellungen bis zum 24. März 1999

- Korrigieren der 1. Dokumentversion
- Einführung zum Thema "Objekterkennung"

A.3 24. März 1999

Zeit: 13:00 - 13:30
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, M. Eckschlager, T. Gangl und M. Liedlgruber

Behandelte Punkte:

- Korrektur der vorigen Dokumentversion
- Besprechung der weiteren Vorgehensweise
- Hinweis auf weitere Bücher und Literatur
- Dokumentumfang noch zu gering

Aufgabenstellungen bis zum 14. April 1999

- Korrigieren der 2. Dokumentversion
- Dokument umfangreicher gestalten (min. 10 Seiten)
- Neuronale Netze (Überblick, biologische Motivation, Klassen, Verarbeitungsneuronen, Synapsen, Funktionalität innerhalb eines künstlichen Neurons, Lernregeln, welche Klassen von ANN gibt es?, Transformation von Neuronen (Igor ALEKSANDER), Beispiel zu einem Kohonen Netz)
- Objekterkennung (welche Arten gibt es? (statistische Bilderkennung, ...) kurze Übersicht über die Methoden verschiedene Stufen der Objekterkennung (smoothing, kantenerkennung, ...), Feature Space, Beispiele mit ANN, wo werden diese in der Objekterkennung eingesetzt (smooting, edgedetection, ...) welche Arten von Kanten gibt es)
- Literaturverzeichnis erweitern
- Bilder zum Thema besorgen

A.4 14. April 1999

Zeit: 13:00 - 13:30
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, M. Eckschlager, T. Gangl und M. Liedlgruber

Behandelte Punkte:

- Korrektur der vorigen Dokumentversion
- Besprechung der weiteren Vorgehensweise
- Hinweis auf weitere Bücher und Literatur

Aufgabenstellungen bis zum 21. April 1999

- Korrigieren der 3. Dokumentversion
- Dokument umfangreicher gestalten (min. 10 Seiten)
- Objekterkennung (welche Arten gibt es? (statistische Bilderkennung, ...) kurze Übersicht über die Methoden verschiedene Stufen der Objekterkennung (smoothing, kantenerkennung, ...), Feature Space, Beispiele mit ANN, wo werden diese in der Objekterkennung eingesetzt (smooting, edgedetection, ...) welche Arten von Kanten gibt es)
- Bilder zum Thema besorgen

A.5 21. April 1999

Zeit: 13:00 - 13:30
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, M. Eckschlager, T. Gangl und M. Liedlgruber

Behandelte Punkte:

- Korrektur der vorigen Dokumentversion
- Besprechung der weiteren Vorgehensweise
- Hinweis auf weitere Bücher und Literatur

Aufgabenstellungen bis zum 5. Mai 1999

- Korrigieren der 4. Dokumentversion
- Thema Objekterkennung noch genauer behandeln
- 2 Wochen Zeit um nun eigenständig eine Projektidee zu entwickeln, bei welcher das bisher erworbene Wissen einfliesen soll (Zusammenhang mit EMMA wäre erwünscht).
- Bilder zum Thema besorgen
- August Mayer und Andreas Lackner (RoboLab) kontaktieren.

A.6 5. Mai 1999

Zeit: 13:00 - 13:30
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, M. Eckschlager, T. Gangl und M. Liedlgruber

Behandelte Punkte:

- Korrektur der vorigen Dokumentversion

- Besprechung der weiteren Vorgehensweise
- Hinweis auf weitere Bücher und Literatur
- Besprechung der Projektidee

Aufgabenstellungen bis zum 19. Mai 1999

- Projekt anfangen
- Bilder zum Thema besorgen
- August Mayer und Andreas Lackner (RoboLab) kontaktieren.

A.7 19. Mai 1999

Zeit: 13:00 - 13:30
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, M. Eckschlager, T. Gangl und M. Liedlgruber

Behandelte Punkte:

- Korrektur der vorigen Dokumentversion
- Besprechung der weiteren Vorgehensweise
- Hinweis auf weitere Bücher und Literatur

Aufgabenstellungen bis zum 2. Juni 1999

- Projekt weiterführen
- Bilder zum Thema besorgen und konvertieren
- Bilder mit Programm KANTEX weiterverarbeiten

A.8 2. Juni 1999

Zeit: 13:00 - 13:30
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, M. Eckschlager, T. Gangl und M. Liedlgruber

Behandelte Punkte:

- Korrektur der vorigen Dokumentversion
- Besprechung der weiteren Vorgehensweise
- Ausgabebilder von KANTEX in das Dokument einbinden
- Präsentationstermin festlegen

Aufgabenstellungen bis zum 16. Juni 1999

- Projekt weiterführen
- Bilder mit Programm KANTEX weiterverarbeiten

A.9 16. Juni 1999

Zeit: 13:00 - 13:30
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, M. Eckschlager, T. Gangl und M. Liedlgruber

Behandelte Punkte:

- Korrektur der vorigen Dokumentversion
- Besprechung der weiteren Vorgehensweise
- Ausgabebilder von KANTEX in das Dokument einbinden
- Präsentationsvorlagen

Aufgabenstellungen bis zum 30. Juni 1999

- Projekt beenden
- Bilder mit Programm KANTEX weiterverarbeiten
- Präsentation vorbereiten

A.10 30. Juni 1999

Zeit: 13:00 - 13:30
Ort: Raum 202.2, Institut für Computerwissenschaften, Universität Salzburg
Anwesende Personen: R. Schwaiger, M. Eckschlager, T. Gangl und M. Liedlgruber

Behandelte Punkte:

- Korrektur der vorigen Dokumentversion
- Ausgabebilder von KANTEX in das Dokument einbinden
- Vorbesprechung der Präsentation

Anhang B

Projekt

01/06

Schnelle Objekterkennung in der Robotik

Manfred Eckschlager
meckschl@cosy.sbg.ac.at

Thomas Gangl
tgangl@cosy.sbg.ac.at

Michael Liedlgruber
mliedl@cosy.sbg.ac.at

Gruppe 1

02/06

Beschreibung der Projektaufgabe

Ziel dieses Projektes war es,Methoden der Objekterkennung (im speziellen mittels Künstlicher Neuronaler Netze), für den am Institut vorhandenen Mobilen Agenten EMMA, zu untersuchen. Dazu mußten folgende Punkte erarbeitet werden.

- Vorverarbeitung der Bilddaten
- Feature Extraktion
- Auswahl und Entwurf eines Neuronalen Netzes
- Simulation des Neuronalen Netzes im SNNS

03/06

Beschreibung möglicher Lösungsansätze

- Ein Bild wird, nachdem es digitalisiert wurde, qualitativ verbessert. Danach wird das Bild segmentiert bzw. die Kanten extrahiert.
  - Qualitštsverbesserung (Gaussfilter, Smoothing, Kontrast)
  - Kantendetektoren (Sobel, Prewitt, Marr-Hildreth)
  - Segmentierung (Grauwert-Thresholding, Texturbasierende Segmentierung, Region-Growing)
- Für die Feature-Extraktion gibt es mehrere verschiedene Features, welche zur Auswahl stehen.
  - Seitenverhältnis der Boundingbox
  - Helligkeit und Textur (Hurst Koeffizient)
  - Anzahl der Bildpunkte im Segment
- Für den Lernvorgang gibt es mehere mögliche Netzwerktopologien, welche verwendet werden können.
  - Kohonennetz
  - Multilayer-Perceptron Netz

04/06

Beschreibung des eigenen Lösungsansatzes I

- **Die Bildvorverarbeitung**
  Für die Vorverarbeitung des Eingangsbildes haben wir und für folgende Filter entschieden:
  - *Gaussfilter*: Der Gaussfilter hat gegenüber dem einfachen Smoothing den Vorteil, daß beim Glätten des Bildes keine neuen Artefakte entstehen.
  - *Kontrast*: Die Kontrastverbesserung des Bildes bewirkt, daß Segmente im Bild besser unterscheidbar sind.
- **Die Kantenerkennung**
  Aus dem nun qualitativ verbesserten Bild, werden nun Kanten extrahiert. Wir haben uns für diese Aufgabe den Sobeloperator ausgesucht. Dies ist ein Gradientenverfahren, bei welchem zuerst die horizontalen, und dann die vertikalen Kanten extrahiert werden. Diese Daten werden dann verknüpft.

05/06

Beschreibung des eigenen Lösungsansatzes II

- **Segmentierung**
  Für die Segmentierung haben wir den Hurst Koeffizienten ausgewählt. Der Grund dafür ist der große Speicherverbrauch des Region-Growing Algorithmus und die sehr eingeschränkte Einsatzmöglichkeit des Grauwert-Thresholding.

- **Die Featureextraktion**
  Wenn das Bild nun in Segmente unterteilt ist, müssen Features extrahiert werden, um damit das neuronale Netz zu speisen.
  - *Helligkeit*
  - *Seitenverhältnis*
  - *Segmentgröße*
  - *Textur*

- **Neuronales Netz**
  Als neuronales Netz wurde das Kohonen Netz verwendet. Es ist ein überwacht lernendes Netz, welches nach der Lernphase eine Ausgangsmatrix mit segmentierten Bereichen liefert. Die Neuronen dieser Segmente müssen dann noch beschriftet werden.

06/06

Ergebnisse

Zur Simualtion wurde der Stuttgarter Neuronale Netze Simulator herangezogen. Da keine Traingsbilder zur Verfügung standen wurden die Trainigspattern mit einem C-Programm erstellt.
Der Simulator wurde mit folgenden Parametern gestartet:

1. Kohonennetz: 4 Eingangsneuronen, 4 x 4 Ausgangsmatrix
2. Lernzyklen: 400
3. Anzahl der Pattern: 8000
4. Lernrate: 0.5, Nachbarschaftsmatrix 3

Nach der Lernphase konnte das Netzwerk Tor(T), Ball(B) und Gegner(G) klassifizieren(siehe Abb.)

Literaturverzeichnis

[1] A. Zell, *Simulation Neuronaler Netze*. Addison-Wesley, 1 ed., 1994.

[2] S. Haykin, *Neural Networks - A Comprehensive Foundation*. Macmillan Publishing, 1 ed., 1994.

[3] H. G. Frey, "Modern geknotet," *Ct Magazin*, vol. 2, p. 256, 1996.

[4] D. A. Pinz, *Bildverstehen*. Springer Verlag, 1 ed., 1994.

[5] A. Watt, *The Computer Image*. Addison-Wesley, 1 ed., 1998.

[6] U. Matecki, *Klassifikation mit Neuronalen Netzen in der Bildverarbeitung*. www.uni-osnab.de, 1999.