PS: Einführung in die Bioinformatik (SS99)
Fast Object Recognition

Obermair Christoph

cober@cosy.sbg.ac.at

Schmidbauer Josef

jschmid@cosy.sbg.ac.at

Stieglbauer Gerald

gstiegl@cosy.sbg.ac.at

Version 1999-07-01

Gruppe 3

Zeit: 17.45 - 18.15

Lehrveranstaltungsleiter: Schwaiger Roland

Inhaltsverzeichnis

1 Einleitung **2**
1.1 Was ist Objekterkennung? 2
1.2 Erkennung von Objekten in Bildern 3
1.2.1 Abgrenzen der Objekte von ihrer Umwelt 3
1.2.2 Vorverarbeitung der Objekte 4
1.2.3 Klassifizieren von Objekten 10
1.3 Was sind künstliche neuronale Netze? 11
1.3.1 Grundbegriffe 11
1.3.2 Ein Neuron in einem künstlichen neuronalen Netz (ANN) 12
1.3.3 Arten von ANNs 12

2 Anwendungsbeispiel **17**
2.1 Projektziele 17
2.2 Der Stuttgarter Neuronale Netze Simulator (SNNS) 18
2.2.1 Auswahlkriterien 18
2.2.2 Begriffe im Umgang mit dem SNNS 18
2.3 Einsatz des SNNS bei der Bilderkennung 24
2.3.1 Aufnahme der Trainingsdaten 24
2.3.2 Ermittlung der gewünschten Ausgabedaten 24
2.3.3 Erzeugung der Patternfiles 24
2.3.4 Ermittlung der optimalen Netzwerkstruktur 25
2.3.5 Beispielbilder bei der Entwicklung des ANNs 26
2.4 Segmentierung 27
2.4.1 Rauschreduktion 28
2.4.2 Sobel Edge Detection Algorithmus 30
2.4.3 Kirsch Edge Detection Algorithmus 30
2.4.4 Situationsspezifische Merkmale des Bildes 31

3 Abschlußbemerkungen **33**

A Bilder **34**
A.1 Bilder der EMMA Quickcam 34

B Programme **37**
B.1 Beispielprogramm zur Feature Extraktion 37
B.2 Segmentierung 39
B.2.1 Algorithmus zur Glättung des Bildes 39

B.2.2 Sobel Edge Detector 39
B.2.3 Kirsch Edge Detector 40
B.2.4 Hough Transformation 40
B.2.5 Kantenfolgungs Algorithmus 42
B.3 Tools zur Erzeugung und Bearbeitung der Pattern Files 43
B.3.1 view 43
B.3.2 bmp242pic 45
B.3.3 pic2eps 46
B.3.4 patdiv 48
B.3.5 makepatt 49

C Protokolle **53**
C.1 09.03.1999 53
C.2 16.03.1999 54
C.3 23.03.1999 54
C.4 13.04.1999 55
C.5 20.04.1999 55
C.6 04.05.1999 56
C.7 18.05.1999 56
C.8 01.06.1999 57
C.9 15.06.1999 57
C.10 29.06.1999 58

Zusammenfassung

Ziel des Projekts ist eine ausführliche Auseinandersetzung mit dem Thema „Objekterkennung“. Dabei werden eingangs grundlegende Methoden der Objekterkennung betrachtet und verglichen. In folge soll eine intensivere Auseinandersetzung mit der Objekterkennung mit Hilfe von künstlichen neuronalen Netzen erfolgen. Daraus soll ein möglichst schnelles Verfahren zur Objekterkennung gesucht werden, welches langfristig auf dem am Institut für Computerwissenschaften der Universität Salzburg entwickelten Roboter E.M.M.A. Einsatz finden soll.

Kapitel 1

Einleitung

1.1 Was ist Objekterkennung?

Informationsverarbeitende Systeme (IVS) können ihre Umwelt nur über bestimmte Schnittstellen wahrnehmen. Diese Schnittstellen empfangen Informationen von außen (Input) und werden als Sensoren bezeichnet. Die Sensordaten können auf verschiedene Weise ausgewertet werden. Ein wichtiges Gebiet ist dabei die „Object Recognition“.
Ein Objekt ist in diesem Kontext durch seine vordefinierten Eigenschaften eindeutig bestimmt und grenzt sich dadurch von seiner Umwelt ab. Man verwendet bestimmte Eigenschaften des Objektes um es von anderen unterscheiden zu können.
Dieses Dokument beschränkt sich darauf, nur jene Eigenschaften zu verwenden, die aus Bildern bestimmt werden können, wobei ein Bild in diesem Zusammenhang eine zweidimensionale optische Abbildung der Umwelt ist.

Im weiteren verstehen wir im speziellen unter IVS Roboter. Damit diese intelligent und effizient mit ihrer Umwelt interagieren können, muß der Roboter Objekte in dieser Umwelt erkennen und lokalisieren können, was wie bereits erwähnt über Sensoren erfolgt. Diese Aufgabe läßt sich in drei Gebiete unterteilen [1]:

- Identitifikation der Gestalt des gesuchten Objektes in einer natürlichen Umgebung.
- Untersuchung des gefundenen Objektes auf seine definierenden Merkmale.
- Feststellung der Position des Objektes bezüglich einer räumlichen Abbildung der Umwelt.

Weitere wichtige Aspekte der Objekterkennung [1]:

- Welche Sensoren sind relevant, um ein bestimmtes Objekt zu erkennen?
- Wie soll man die definierenden Merkmale eines individuellen Objekt bestimmen?
- Wie können große Mengen von Klassifizierungsmerkmalen effizient gespeichert werden?
- Welche Adressierungsmethoden werden verwendet, um bestimmte Objekte aus einer solchen Datenbank ableiten zu können?

- Welche Methoden werden verwendet, um eine Beziehung zwischen Sensordaten und Objekteigenschaften herzustellen?
- Wie kann man aus Sensordaten die Position und Lage eines Objektes herleiten?
- Wie werden Eigenschaften neuer Objekte erkannt und in der Datenbank gespeichert?
- Welche Rolle spielt die Gewichtung bei der Auswahl derjenigen Sensordaten, die für das Objekt wesentlich sind?
- Welche Rolle spielt die Bewertung einer speziellen Situation bzw. die Wahrscheinlichkeit von Ereignissen bei der Erkennung?
- Welchen Beitrag leistet die Funktion oder der Zweck eines Objektes zu dessen Identifikation?
- Welche Rolle spielen die geometrischen Eigenschaften und Merkmale bei dessen Lokalisierung?

1.2 Erkennung von Objekten in Bildern

Die Erkennung von Objekten läßt sich wie in 1.1 beschrieben in zwei Gebiete einteilen:

- Abgrenzen der Objekte von ihrer Umwelt
- Vorverarbeitung von Objekten
- Klassifizieren von Objekten

1.2.1 Abgrenzen der Objekte von ihrer Umwelt

Eine wichtige und grundlegende Komponente der Objekterkennung ist das Abgrenzen der Objekte, die sogenannte Segmentierung. Hier werden einzelne Pixel in einer Bildmatrix als Teile von verschiedenen Objekten bzw. Bereichen einer Szene identifiziert.
Objekte können durch verschiedene physikalische Gesetzmäßigkeiten gekennzeichnet sein und somit aus einem Bild herausgelöst werden[2]:

- **Formkonturen**

  Kontinuierliche Kanten markieren meist die Grenze eines Objekts, das andere Gegenstände oder einen Hintergrund verdeckt. Solche Konturen hängen in der Regel mit abrupten diskontinuierlichen Tiefenänderungen zusammen. Dazu gehören aus Bewegungen resultierende, parallele Verschiebungen. Kontinuierliche Kanten, abrupte Tiefenänderungen und Bewegungsparallaxen grenzen einen Gegenstand von seiner Umgebung ab.

- **Textur**

  Die Textur einer Oberfläche ist durch Bereiche ähnlicher Helligkeitsdichten gekennzeichnet, die meist von geringfügigen Tiefenänderungen verursacht werden. Im Gegensatz zum vorherigen Punkt werden diese Tiefenänderungen aber nicht verwendet, um durch sie Bereiche voneinander abzugrenzen.

- **Helligkeit**

  Besondere Helligkeitseffekte entstehen besonders beim Zusammentreffen mehrerer Oberflächen in einer Ecke. Die Helligkeitsunterschiede auf den verschiedenen Oberflächen sind graduierlich und nicht abrupt.

- **Schatten**

  Schatten sind auch kontinuierliche Kanten, gehen jedoch nicht mit abrupten Tiefenänderungen einher.

- **Bewegung**

  Sich parallel bewegende Teile gehören meist zum gleichen Objekt.

Textur geht über Helligkeitseffekte und Schatteneffekte hinweg, nicht jedoch über Formkonturen.
Es ist schwierig die Segmentierung als einen generellen Lösungsansatz zu implementieren, da die Auswahl der angeführten Gesetzmäßigkeiten typischerweise sehr problemabhängig sind [3].

1.2.2 Vorverarbeitung der Objekte

Vor der Klassifizierung wird eine Vorverarbeitung durchgeführt. Dabei werden wichtige Eigenschaften von Objekten extrahiert und in einem n-dimensionalen Raum, den sogenannten Feature Raum (oder Eigenschafts-Raum), eingetragen [3], [4]. Methoden zum Klassifizieren von Objekten setzen u. U. auf diesen Feature Raum auf und versuchen anhand der darin enthaltenen Daten Objekte einzuteilen.
Jede verwendete Eigenschaft eines Objektes wird auf eine Achse im Feature Raum abgebildet. Je größer die Anzahl der verwendeten Eigenschaften (Features) ist, desto besser können die Objekte im allgemeinen differenziert werden (einzelne Verfahren können natürlich aufgrund der größeren Komplexität eines höherdimensionalen Featureraumes bei einer Vergrößerung der Anzahl der verwendeten Eigenschaften wieder schlechtere Ergebnisse liefern), aber die Performance (Geschwindigkeit) sinkt mit der Anzahl der verwendeten Features [5]. Durch eine vorangehende Reduzierung des Feature Raums (Veringerung der Dimension des Raumes) auf die jeweils wesentlichsten Eigenschaften kann man die Komplexität verringern und die Geschwindigkeit erhöhen. W. Siedlecki und J. Sklansky [3] beschreiben genetische Ansätze um die Dimension des Feature Space zu reduzieren.
Wesentliche Eigenschaften sind vom Anwendungsfall abhängig. So ist z.B. die Verwendung der Objekthelligkeit als Feature für die Objektklassifizierung in einer Umgebung mit konstanter Helligkeit sinnvoller als in Umgebungen mit veränderlichen Beleuchtungsverhältnissen.
Eine gängige Methode zur Extraktion von Features ist die Fourier Transformation [6]. Dabei werden aus dem Bild des Objektes Frequenz und Phaseninformationen gewonnen. Je nach Anwendungsfall kann man dadurch erreichen das z.B. beim Verwenden der Frequenzinformation und Weglassen der Phaseninformation ein Objekt auch bei einer verschobenen Abbildung die selbe Position im Feature Space hat [4]. Weitere Feature-Extraktionsverfahren sind z.B. Moment Feature Space [4], Chord Distribution Feature Space, die Hough Transformation [3], Wigner Distribution Feature Space [3], Orthogonal Transformations [4] und Gabor Funktionen [3].

Bei verschiedenen Feature-Extraktionsverfahren lassen sich künstliche neuronale Netze (Artificial Neural Networks, ANN) einsetzten. So beschreibt z.B. Kulkarni [4] Modelle von ANNs für die Feature-Extraktion via Fourier Transformation und via Walsh-Hadamard Transformation.
Im Feature Space wird nun jedem Objekt aufgrund seiner Eigenschaften ein eindeutiger Punkt zugeordnet. Je geringer die Distanz zwischen zwei benachbarten Punkten im Feature Space ist desto ähnlicher sind die Eigenschaften der Objekte die sie repräsentieren. Das bedeutet aber im allgemeinen nicht, daß benachbarte Objekte unbedingt der selben Klasse angehören müssen [5]. Die Feature Space Abbildungen einer Klasse von Objekten befinden sich in einem Teilraum des Features Space. Im Idealfall überschneiden sich die Teilräume verschiedener Objektklassen nicht und jedes Objekt lässt sich direkt zu dem entsprechenden Teilraum und damit einer eindeutigen Objektklasse zuordnen [5].
Klassifizierungstechniken unterteilen anschließend den Feature Space in diese Teilräume und schließen durch den, dem Punkt zugeteilten Teilraum auf die Klasse des Objektes (z.B. [4],[7]).

Beispiel zur Extraktion von Features

Einleitung:
Anhand des Helligkeitsschnittes eines Bildes soll eine bestimmte Kantenart erkannt werden. Es wird angenommen, daß die für die Feature Berechnung notwendigen Daten in einer geeigneten Form vorliegen. Speziell in diesem Beispiel wurde die Vorverarbeitung des Bildes bereits durchgeführt, sodaß nun ein Helligkeitsschnitt vorliegt der analysiert werden soll. Aus dem Objektabbild sollen nun Klassifizierungen in verschiedene Kantenarten ermöglicht werden. Dazu wird das Abbild des Objektes (Helligkeitsverlauf) in einen Feature Space abgebildet.
Beispieldaten:
Als Beispieldaten werden Helligkeitsabtastwerte einer Linie im Bild verwendet. Die Helligkeitswerte reichen dabei von -1 bis +1. Es wurden 256 diskrete Helligkeitswerte verwendet. Die Daten stellen ein symmetrisches Objekt mit verschieden starken Kanten dar und reichen von einem Objekt mit sehr weichen Konturen (erzeugt durch eine Cosinusfunktion) bis zu einem Objekt mit sehr harten Kanten (Rechteck). Es wurden vom Rechner jeweils 64 Objekte erzeugt deren Kanten zwischen den beiden oben genannten Typen liegen. Jede dieser Objektabbildungen wurde mit verschieden starken Rauschen versehen. Die Bandbreite reicht hier von keinem Rauschen [Abb. 1] bis zu sehr starken Rauschen [Abb. 3].
Programm:
Das für diesen Zweck geschriebene Programm (siehe B.1) erzeugt nun aus den Helligkeitswerten (Objektabbild) zwei verschiedene Features. Als erstes wird aus dem Helligkeitsverlauf ein diskretes Fourierspektrum erstellt (dargestellt werden nur die ersten 16 Fourierkoeffizienten). Daraus wird nun als erstes Feature das Verhältnis von Signal (Grundschwingung) zu Oberwellen (ganzzahlige Vielfache der Grundschwingung) berechnet. Diese Eigenschaft kann als Maß für die Weichheit einer Kante verwendet werden. Bei sinusähnlichen Schwingungen gibt es kaum Oberwellen, und der Koeffizient Grundschwingung/Oberwellen wird sehr groß. Je härter eine Kante wird, desto größer werden die Oberwellen und desto kleiner der Koeffizient. Als zweite Eigenschaft wurde die Steilheit der Kante verwendet. Zuerst wird vom Programm die maximale und die minimale Helligkeit gesucht. Das dazu verwendete Signal wurde vorher mit einem digitalen Tiefpaß gefiltert, um es zu glätten. Anschließend

wird vom ersten Punkt mit 20% der Helligkeit zum ersten Punkt mit 80% der Helligkeit die Steigung gemessen (durch eine Gerade angenähert). Diese Steigung wird ebenfalls als Feature in den Feature Space eingetragen.
Das Programm kennzeichnet verschiedene Kantenarten dabei mit verschiedenen Farben im Feature Space (grün entspricht harten Kanten, rot entspricht weichen Kanten). Die Farbe hängt dabei nur von der ursprünglichen Signalform ab. Es wird keine Teilung des Feature Space unternommen. In vertikaler Richtung wird die Signalsteilheit aufgetragen. In horizontaler Richtung das Grundschwindungs/Oberwellen Verhältnis.

Abbildungen:

- In Abbildung 1 wird eine sinusähnliche Kante verwendet. Die daraus berechneten Fourierkoeffizienten zeigen, daß es kaum höhere Frequenzen als die Grundschwingung gibt. Daraus resultiert auch das hohe Grundschwingung/Oberwellen Verhältnis (in der Abbildung als SNR bezeichnet). Als zweites Feature wurde die Steigung der Kante wie oben beschrieben gemessen. Beide Werte werden im Diagramm „Feature Space“ als Punkt dargestellt.

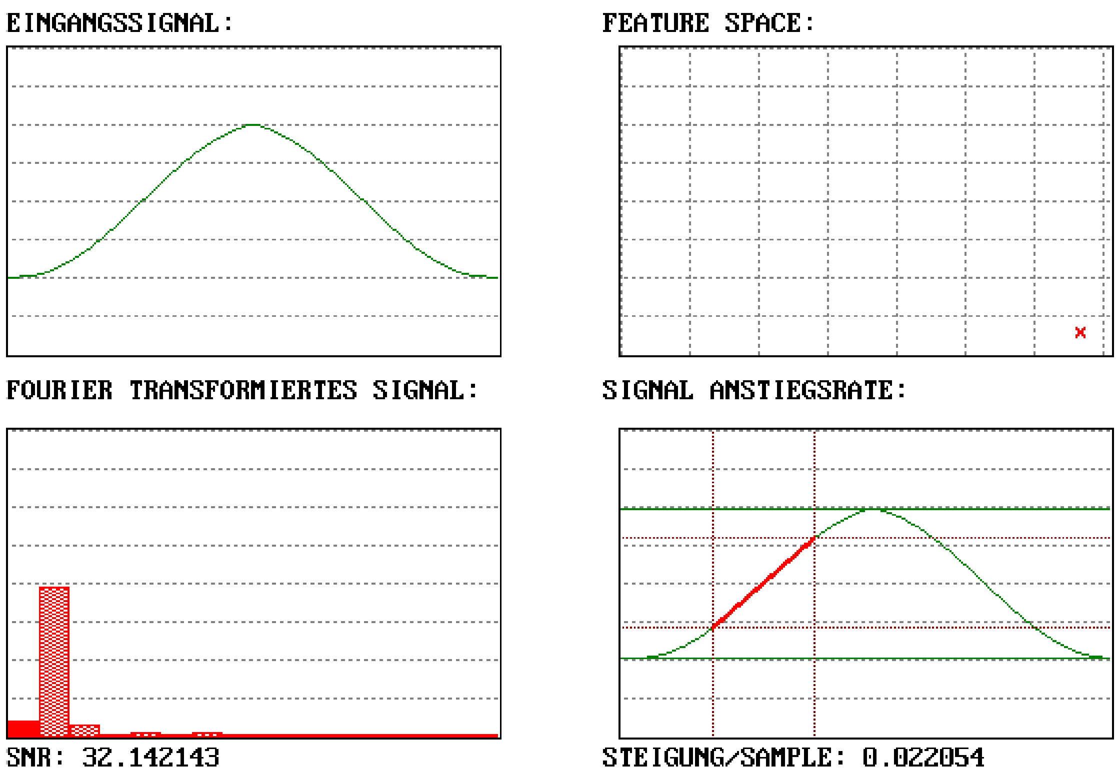


Abbildung 1

- In Abbildung 2 wird eine sehr steile Kante analysiert (Rechteck). Die Fourierkoeffizienten zeigen die erwarteten Oberwellen bei ungeraden Vielfachen der Grundschwingung. Daraus resultiert ein sehr kleines Grundschwindungs/Oberwellen Verhältnis. Die Steigung der Kante ist hier relativ groß. In der Abbildung des Feature Space sind hier bereits verschiedene Punkte von Testkanten eingetragen. Alle in dieser Abbildung eingetragenen Punkte stammen von Kanten ohne Rauschen.

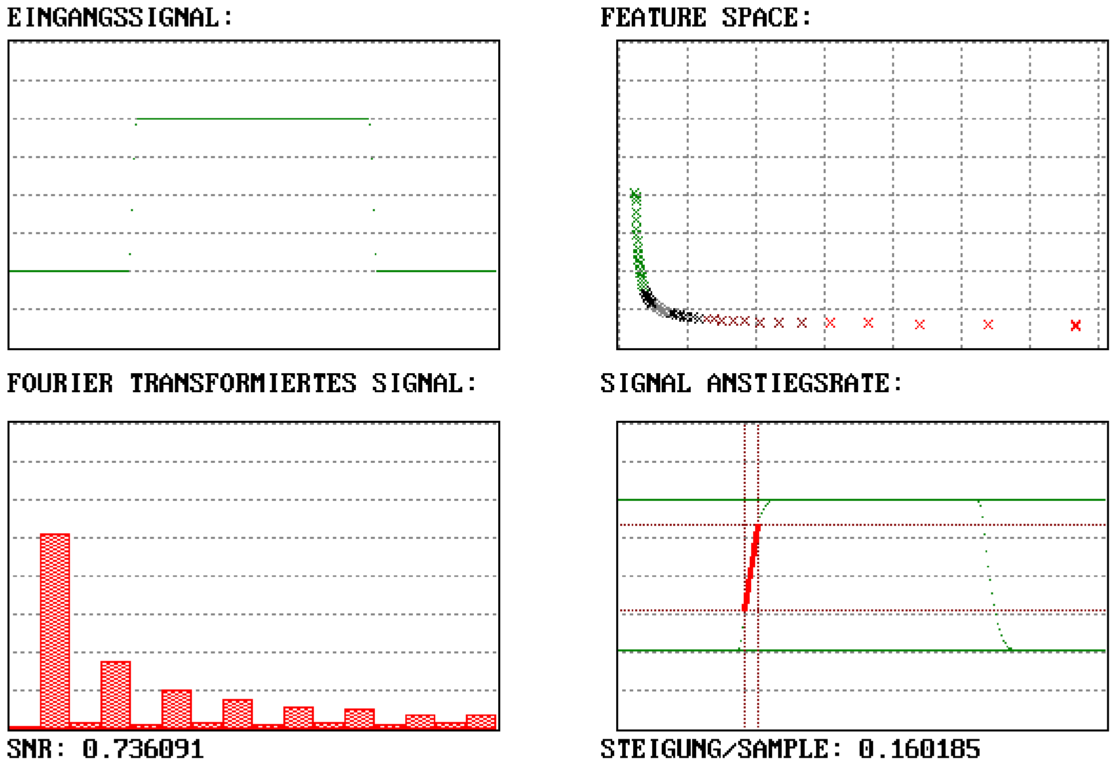


Abbildung 2

- In Abbildung 3 sieht man ein verrauschtes Rechtecksignal. Die Fourierkoeffizienten zeigen wie erwartet auch Frequenzen die nicht Vielfache der Grundschwingung sind. Auch wird die Berechnung der Signalsteigung durch das Rauschen erschwert. Allerdings kann, bedingt durch die vorhergehende Signalfilterung, auch hier die Steigung noch relativ genau berechnet werden. Im Feature Space Diagramm sieht man die Punkte von Abbildung 2 und die Punkte einiger verrauschter Kanten. Man sieht aufgrund der Farbmakierungen, daß unterschiedliche Kanten gut zu differenzieren sind und durch eine Partitionierung des Feature Spaces eine gute Einteilung in Objektklassen möglich sein sollte.

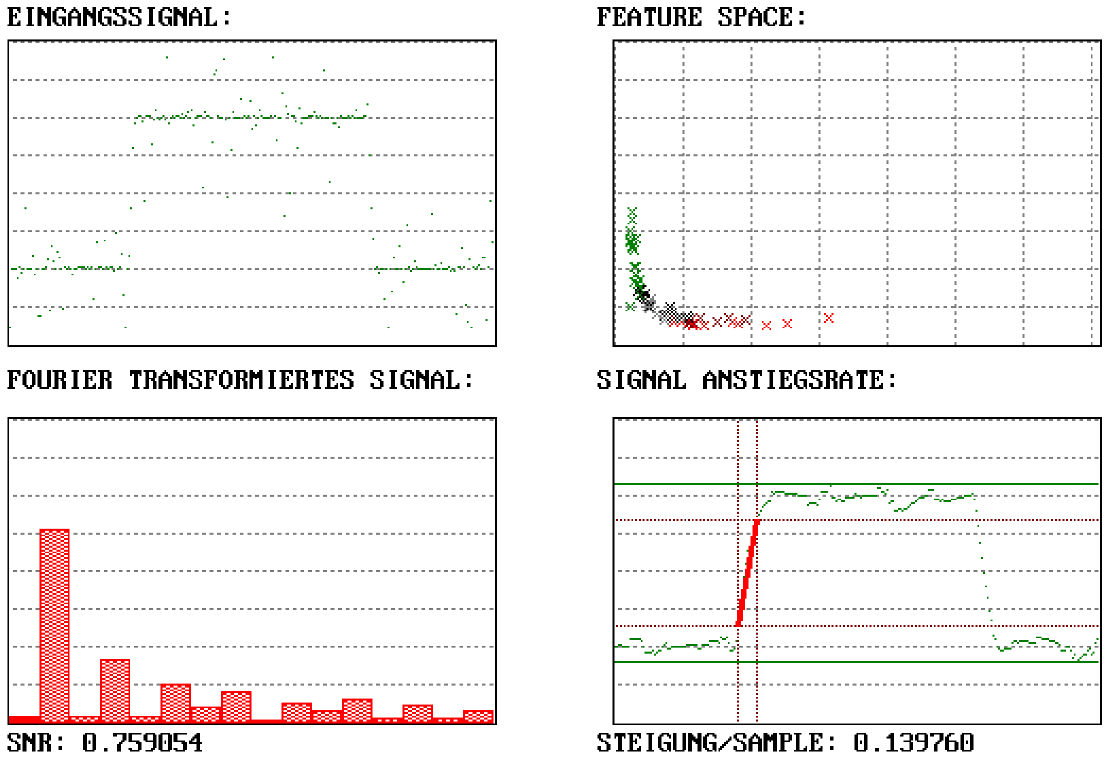


Abbildung 3

- In Abbildung 4 sieht man ein sehr stark verrauschtes Signal einer weichen Kante. Mit Hilfe der Fourierkoeffizienten erkennt man kaum noch die Art des Signales. Nur die Grundschwinung ist noch deutlich erkennbar. Auch die Steigung der Signalflanke kann, trotz Vorbearbeitung des Signales mit einem digitalen Filter, nicht mehr genau bestimmt werden. Daher sind auch die Werte im Feature Space sehr verschieden und können daher bei derart verrauschten Signalen nicht mehr eindeutig einer Art zugeordnet werden. Im Feature Space Diagramm sind hier die Punkte einiger hundert, verschieden stark verrauschter Beispielsignale dargestellt.

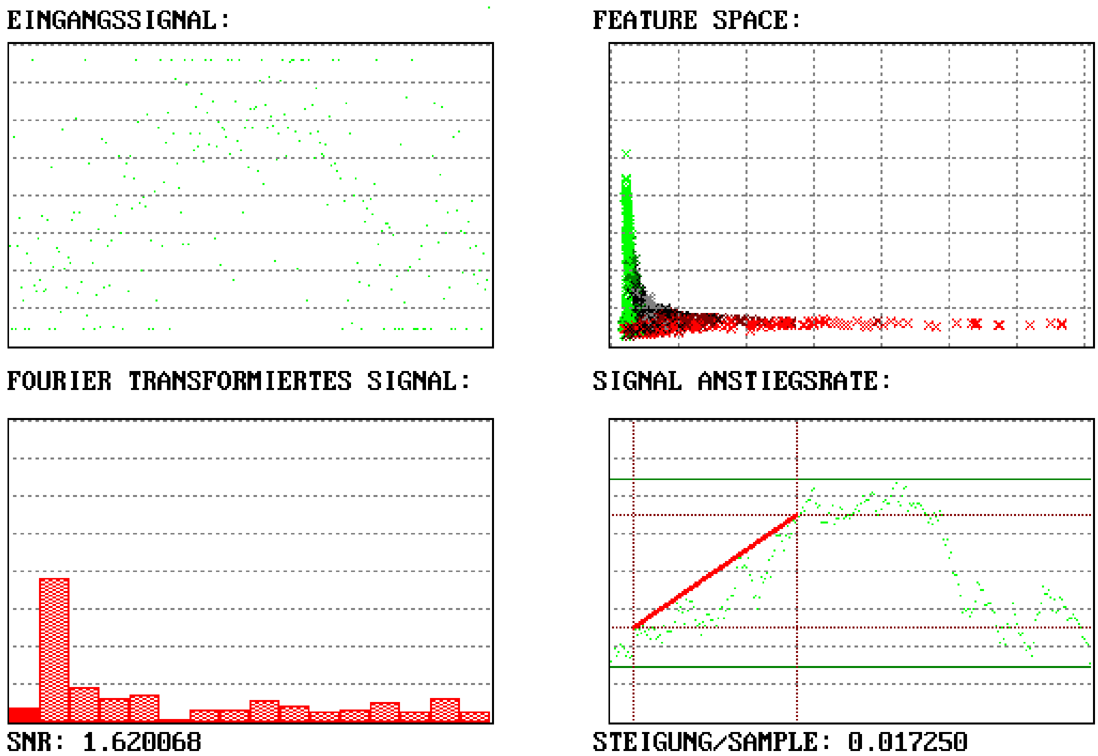


Abbildung 4

Bemerkungen:
Man sieht, daß bei unverrauschten Signalen schon durch die Verwendung eines der beiden Features eine sehr gute Unterscheidung der Objektklassen (hier mit K1 und K2 bezeichnet, wobei K1 die Klasse der Objekte mit einer harten Kante ist und K2 die Klasse der Objekte mit einer weichen Kante) möglich ist (Abbildung 5/6). Dabei würde man mit einem eindimensionalen Feature Space auskommen.

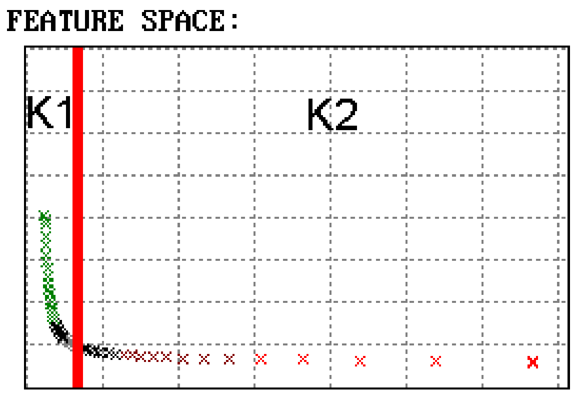


Abbildung 5

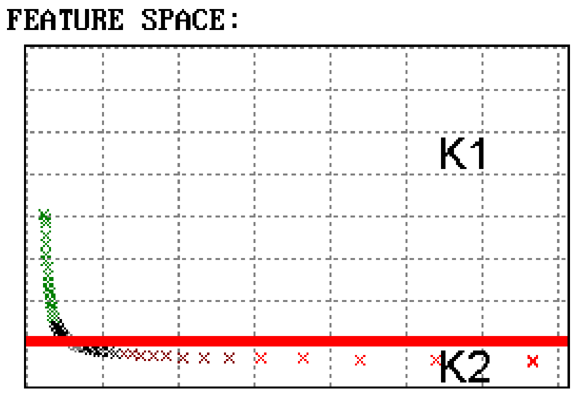


Abbildung 6

In Abbildung 7 sieht man das bei verrauschten Signalen die Verwendung von nur einem Feature keine guten Ergebnisse mehr liefert. Durch die Verwendung beider Features ist hier eine bessere Trennung der Objektklassen möglich.

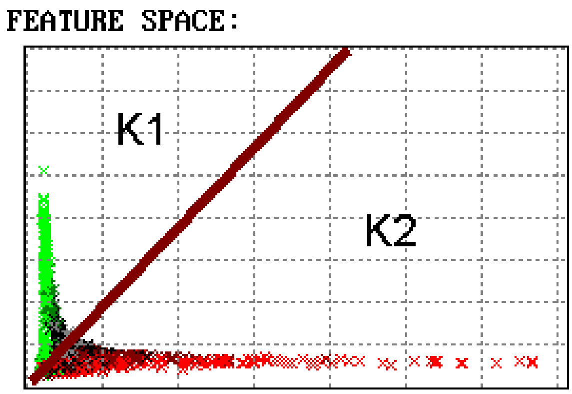


Abbildung 7

Weiters ist ersichtlich, daß durch die verwendeten Features eine sehr gute Trennung der in den Beispieldaten verwendeten Objektklassen möglich ist. Selbst bei verrauschten Signalen ist noch eine gute Zuordung zu einer der Klassen möglich. Nur bei sehr stark verrauschten Signalen wird eine Differenzierung erschwert, da sich hier Punkte im Feature Space überlappen.

1.2.3 Klassifizieren von Objekten

Die konventionellen Klassifizierungstechniken für Objekte lassen sich in Anlehnung an [4] grob in zwei Kategorien eingeteilt werden:

1. **Statistische Methoden**

   Verwendet statistische Eigenschaften im Objekt-Eigenschafts-Raum (Feature Space) um Objekte zu klassifizieren.

2. **Syntaktische Methoden**

   Verwendet eventuell wichtige, strukturelle Beziehungen zwischen Eigenschaften um Objekte zu klassifizieren.

Weiters kann man die Klassifizierungstechniken auch in

1. **Überwachte (supervised) Methoden**

   Hier steht für jede Klasse eine Anzahl von Übungsbeispielen zur Verfügung, welche verwendet werden um das System zu trainieren

2. **Nicht-überwachte (unsupervised) Methoden**

   Bei nicht überwachten Verfahren stehen keine Beispiele zur Verfügung. Das System versucht die Daten selbst zu klassifizieren.

einteilen. Beispiele für die Verwendung von überwachten und nicht-überwachten Methoden in den konventionellen Klassifizierungstechniken (nicht ANN-basierende Verfahren) finden sich in [7, 58 ff., 109 ff.]

1.3 Was sind künstliche neuronale Netze?

1.3.1 Grundbegriffe

Neuronale Netze können als technische Umsetzung der Gehirnfunktionen verstanden werden. Das Gehirn besteht im wesentlichen aus Nervenzellen (Neuronen). Der Zellkörper des Neurons trägt eine große Anzahl kurzer stark verzweigter Auswüchse, die als Dendriten bezeichnet werden. Ein weiterer Auswuchs ist das Axon, das im Vergleich mit dem Dendritenbaum sehr lang ist. Auf jedem Axon befinden sich tausende von Synapsen, die eine Verbindung mit den Dendritenbäumen anderer Neuronen haben [8].
Der Lernvorgang in einem neuronalen Netz kann nun durch zwei Phänomene erklärt werden:

- Die synaptischen Verbindungen ändern sich im Lauf der Zeit. Synapsen können wachsen, verkümmern oder sogar ganz verschwinden. Umgekehrt kann ein Axon neue Zweige mit den zugehörigen Synapsen ausbilden und dadurch mit weiteren Nervenzellen in Kontakt treten.

- Therorie von Donald Hebb [9]:

  Wenn ein Axon der Zelle A nahe genug bei der Zelle B liegt, um diese mehrmals und kontinuierlich zu erregen, so tritt ein Wachstumsprozeß oder eine Stoffwechseländerung in einer oder beiden Zellen auf, so daß die Effizenz der Zelle A Zelle B zu feuern steigt.

Wann feuert eine Nervenzelle?
Eine bestimmte Nervenzelle ist immer mit mehreren vorgeschalteten Neuronen verbunden. Feuern nun diese Zellen, so werden diese Signale über die Synapsen im Dendritenbaum der zu untersuchenden Zelle an den Zellkörper (Soma) weitergeleitet und dort aufsummiert. Überschreitet die aufsummierte Spannung einen gewissen Schwellenwert, so wird ein sogenanntes Aktionspotential ausgelöst, d.h. die Zelle feuert. Bei dem Aktionspotential handelt es sich um eine kurze Spannungspitze, die nun am Axon abgegeben wird und somit wiederum für das feuern nachgeschalteter Neuronen mitverantwortlich sein kann.

Weiters ist zu erwähnen, daß es nicht nur Synapsen gibt, die das Membranpotential erhöhen, also erregend wirken, sondern solche Synapsen existieren, die hemmend auf dieses Potential wirken. Außerdem wird die Information, die im Nervensystem verarbeitet werden nicht durch die Größe der Spannung, sondern durch die zeitlichen Abstände der Aktionspotentiale kodiert. Das Nervensystem arbeitet demnach mit Frequenzmodulation (siehe Kohonen).

1.3.2 Ein Neuron in einem künstlichen neuronalen Netz (ANN)

Ein künstliches Neuron, läßt sich nun aufgrund der oben beschriebenden Grundlagen folgendermaßen beschreiben:

- Die Eingänge des künstlichen Neurons stellen den *Dendritenbaum* dar.
- Die an den Eingängen anliegenden Signale werden nun in einer zentralen Stelle aufsummiert. Dies geschieht im biologischen Modell im Zellkörper.
- Das aufsummierte Signal wird nun wie oben bereits erwähnt mit einem Schwellenwert (Threshold) verglichen.
- Ist das aufsummierte Signal größer oder gleich dem Schwellenwert, so liegt am Ausgang (Axon) eine logische 1 an, die das Aktionspotential darstellt.
- Weiters werden hemmende Synapsen in Form eines negativen Eingangswertes (Subtraktion) bei der Aufsummierung realisiert.
- Außerdem wird die Hebb'sche insofern berücksichtigt, indem sich an den Eingängen des Neurons jeweils ein Multiplikator befindet, der das Eingangssignal um eine bestimmten Faktor ändert. Dieser Faktor wird bei jeder presynaptischen Aktivität um eine gewissen Wert erhöht und anderfalls, wenn keine Aktivität vorliegt erniedrigt. Dies hat somit zur Folge, daß bei länger andauernden Eingangssignalen die Nervenzelle leichter feuert.

1.3.3 Arten von ANNs

Farley and Clark Network

Bei diesem Netzwerk handelt es sich um das erste, das auf der Hebb'schen Theorie basiert. Dabei wird das Netzwerk in vier Quadranten unterteilt. Weiters befinden sich auf der linken Seite die Eingänge und auf der rechten Seite die Ausgänge des Netzwerkes. Die Anordnung der Verbindungen zwischen den Eingängen und den Ausgängen erfolgt zufällig [10].
Nun wird abwechselnd ein Muster im oberen und unteren linken Quadranten angelegt und der Signalweg verfolgt. Ziel ist es bei einem bestimmten Muster in einem der beiden Ausgangsquadranten eine höhere Aktitivität als in dem anderen zu bewirken. Geht nun das Signal einen für das Ergebnis günstigen Weg, so werden die einzelnen Multiplikatorenfaktoren der einzelnen auf diesem Weg liegenden Neuronen inkrementiert. Anderfalls, wenn sich der Signalweg als ungünstig erweist, werden die einzelnen Faktoren dekrementiert.

Nachteile:

- Schlechte Unterscheidung verschiedener Muster.

- Hohe Anzahl von Neuronen für eine Unterscheidung von Mustern notwendig.

Self-Assembly Network

Im Gegensatz zum Farley und Clark Netzwerk ist bei diesem Netzwerk nur eine Spalte von Neuronen vorhanden. Die Ausgänge werden nun zufällig mit den Eingängen anderer Neuronen verbunden (ein Ausgang wird mit 10 Eingängen verbunden).
Da jedes Neuron dadurch mehrere Eingänge besitzt, kann es passieren, daß dadurch die Multiplikatoren bei mehrmaliger Aktivität einen zu großen Summierungswert verursachen. Deshalb werden die Multiplikationsfaktoren bei geringer Aktivität jeden Taktzyklus veringert. Dadurch wird erreicht, daß sich alle Gewichte immer zu einem konstanten Wert aufsummieren.
Zusätzlich können Zellen bei zu hoher Aktivität „ermüden“. Dies wird durch eine Erhöhung des Schwellenwertes bewirkt.

The Original Perceptron

Dieses Netzwerk ist ein Beispiel für ein selbstständig lernendes Netzwerk. Die Original Perceptron Schaltung besteht aus vielen gleichartigen Teilschaltungen, deren Ausgänge mit einem Komparator verbunden sind.
Der Erfinder dieses Netzwerks (Rosenblatt) bezeichnet diese Schaltungen als A-Einheiten. Jede Teilschaltung erhält eine bestimmte Anzahl von binären positiven Eingängen und eine geringere Anzahl von negativen Eingängen. Wenn die Summe dieser Eingänge den Schwellenwert überschreitet, so liegt an diesen Ausgängen der Wert 1 an. Dieser Wert wird erst jetzt mit einem Faktor multipliziert, wobei dieser zwischen 0 und 1 liegt und als Wert der A-Einheit bezeichnet wird. Die negativen Eingänge bewirken eine Verbesserung der Erkennungsrate bei Teilmustern.
Der mit den Ausgängen verbundene Komparator wählt nun den größten Wert aus, um diesen durchzuschalten. Der durchgeschaltene Ausgang wird nun zur Teilschaltung zurückgeführt, um den Multiplikationsfaktor um einen bestimmten Betrag zu erhöhen. Gleichzeitig werden die anderen Faktoren um einen entsprechenden Betrag erniedrigt.
Ein neuronales Netzwerk, das einen derartigen Komparator verwendet, wird als konkurrierendes Netzwerk bezeichnet, weil die einzelnen Teilschaltungen um die Auswahl ihres Signals konkurrieren.

ADALINEs

ADALINE ist ein Akronym für Adaptive Linear Element.
Ein bestimmtes Bitmuster wird an den Eingängen angelegt. Anschließend wird jeder Eingang mit einem bestimmten Faktor multipliziert und an einem zentralen Punkt geleitet, wo alle manipulierten Eingänge aufsummiert werden. Dieser Wert liegt nun vorläufig am Ausgang an. Dieser Ausgangswert wird nun mit einem sogenannten Zielwert verglichen. Dieser Zielwert soll für jedes Eingangsmuster charakteristisch sein. Die Multiplikationsfaktoren werden nun um einen von der Differenz zwischen dem Ausgangswert und dem Zielwert abhängigen Wert inkrementiert bzw. dekrementiert. Ziel ist es, durch mehrmaliges alternierendes Anlegen verschiedener Muster dem Zielwert möglichst nahe zu kommen.

Eine Verbesserung wird weiters dadurch erreicht, daß nur jene Multiplikationsfaktoren für eine etwaige Gewichtungsänderung berücksichtigt werden, an deren Eingänge eine logische 1 anliegt.
Die allgemeine Formel zur Berechnung der Gewichtungsänderung lautet nun:

$$Gewichtungs\ddot{a}nderung = \frac{Eingangsbitwert * Fehler}{AnzahlderEing\ddot{a}nge} \tag{1.1}$$

The Classic Perceptron

Diese Teilschaltung stellt eine Kombination des Original Perceptron mit ADALINE-Elementen dar. Das so entstandene Netzwerk basiert nun ausschließlich auf Mißerfolgslernen. Das heißt die einzelnen Gewichte werden nur bei einer Mißklassifizierung verändert.
Die einzelnen Multiplikationseinheiten befinden sich hier wie ADALINE vor dem Summierungspunkt. Anstatt einer normalen Schwellenwerteinheit wird nun eine sogenannte subtrahierende Schwellenwerteinheit verwendet. Diese zieht einfach vom Eingangswert einen konstanten Wert ab. Ist das Ergebnis kleiner null, so liegt am Ausgang allerdings eine Null an. Zustätzlich sind die Eingänge nicht zufällig verschaltet, sondern die Eingänge der einzelnen Teilschaltungen erhalten jeweils das selbe Eingangsmuster.
Diese Vorraussetzungen ermöglichen nun auch nicht-boolsche Ausdrücke korrekt zu verarbeiten. Bei diesen Ausdrücken kann es sich z.B. um Seitenverhältnis, Farbe, Helligkeit, usw. von Objekten handeln. Weiters sollten sich die einzelnen Eingangswerte jeweils auf einen konstanten Wert aufaddieren, um eine verläßliche Unterscheidung von verschiedenen Eingangsmustern zu gewährleisten.
Die Korrektur der einzelnen Multiplikationsfaktoren erfolgt proportional zu dem am Ausgang ermittelten Fehler.

The Association Network of Kohonen and Anderson

Dieses Netzwerk verwendet eine Reihe analoger ADALINE Schaltkreise zur Unterscheidung zwischen verschiedenen Mustern. Kohonen und Anderson verwendeten zur Darstellung dieses Netzwerkes einfache mathematische Matrizen. Das Ziel dieses Netzwerkes ist es, bei einem bestimmten Eingangsmuster (Vektor) einen bestimmten Ausgangsvektor zu ermitteln, der mit einem für das am Eingang anliegende Muster erwünschten Vektor, möglichst übereinstimmen soll. Der Unterschied zu den vorigen Netzwerken besteht also u. a. darin, daß alle und nicht nur ein Ausgang bei der Bewertung, welches Muster gerade anliegt, herangezogen werden.
Die Gewichte für jede Teilschaltung werden als eine Reihe in der Matrix dargestellt. Die einzelnen Ausgangswerte können somit mit Hilfe des inneren Produktes zwischen der Gewichtsmatrix und dem Eingangsvektor berechnet werden.
Für die Neuberechung der Gewichte kann wiederum die zuvor besprochene leicht modifizierte Widrow-Hoff Regel angewendet werden.

The Cognitron

Diese Netzwerk von Kunihiko Fukushima soll eine Erweiterung des Original Perceptron darstellen. Wie das Original Perceptron ist dieses Netzwerk fähig on fremde Hilfe Muster zu erkennen und zu unterscheiden. Darüber hinaus weist es bei der Erkennung verschachtelter Muster wesentlich bessere Eigenschaften auf. Dies geschieht durch die zusätzliche Verwendung eines hemmenden Eingangs der zu den einzelnen Teilschaltungsknoten führt. Das Netzwerk lernt einzelne Muster, indem die durch den abschließenden Komparator ausgewählten Eingangsmultiplikatonsfaktoren durch den Wert der folgenden Formel inkrementiert werden:

$$ErhöhungDerFaktoren = \frac{PropKonst * PosSummWert}{AnzDerEingänge} \tag{1.2}$$

Für die hemmende Eingänge wir weiter folgende Formel verwendet:

$$ErhöhungDerFaktoren = \frac{MusterKonst * SummeAllerPosEingänge}{GesMusterwert} \tag{1.3}$$

Der endgültige Ausgangswert wird nicht einfach aus der Subtraktion des negativen Eingangs (h) von der Summe der positiven Eingänge (e) berechnet, sondern ergibt sich aus folgender Formel:

$$y = \frac{e - h}{1 + h} \tag{1.4}$$

Eine Erweiterung dieses Netzwerkes besteht in der Verwendung eine weiteren Verknüpfung (OR-Operation) nach dem abschließenden Komparator. Dadurch kann auch ein Objekt wiedererkannt werden, das in gewissen Zeitabständen seine Position verändert bzw. sich bewegt. Diese Erweiterung wird als das Neocognitron bezeichnet.

The Hopfield-Network

Dieses von John Hopfield entwickelte Netzwerk unterscheidet sich dadurch von den vorigen Netzwerken, indem es bidirektionale Signalwege ermöglicht. Die einzelnen Summationsknoten besitzen einen Schwellenwert von 0. Wird dieser erreicht oder überschritten, so liefert der Knoten den Wert 1. Die einzelnen Multiplikationsfaktoren können Werte zwischen -1 und 1 aufweisen. Dadurch werden excitatorische und inhibitorische Eigenschaften im Netzwerk implementiert.

Die allgemeine Idee, die dem Netzwerk lernende Eigenschaften zukommen laßt, ist die, daß Gewichte, die sich zwischen zwei aktiven Knoten (d.h. solche Knoten, die eine 1 ausgeben) befinden, inkrementiert werden, währendessen alle anderen Gewichte dekrementiert werden. Zusätzlich wird im originalen Bericht von Hopfield noch zusätzlich verlangt, daß sich alle Gewichte zu 0 aufaddieren mußsen.

Um nun eine Korrelation zwischen dem Eingangs- und dem Ausgangsmuster feststellen zu können, sind mehrere Trainingseinheiten notwendig, bei denen das Netz ein Muster lernen soll. Die Einheiten werden solange wiederholt, bis sich die einzelnen Gewichte nicht mehr oder nur mehr kaum ändern.

Der Vorteil dieses Netzwerk ist der, daß es sowohl bei der Unterscheidung von Mustern, als auch bei Verallgemeinerungen von Teilen von Mustern verwendet werden kann.

Der Nachteil ist allerdings, daß die Anzahl der Verbindungen zwischen den einzelnen Knoten exponentiell mit der Anzahl der Knoten steigt. Weiters stellen die Anforderungen auf Normierung (aufaddieren aller Gewichte auf 0), sowie die binären Eingänge Einschränkungen dar.

Reilly and Cooper's Hybrid Network

Dieses Netzwerk besteht aus zwei Layern, wobei jeder Layer eine andere Aufgabe erfüllt. Der erste Layer mit konvergenten Teilschaltungen ist zuständig für die Generalisierung, der zweit für die Spezielisierung. Dies ist eine „Fundamental Pattern Classification Strategy". Jede Klasse von Mustern hat einen bestimmten Ausgang.
Liegt am Eingang des Netzwerkes ein Muster das keinen Output erzeugt, so werden die Gewichte der aktiven Signale im ersten Layer auf einen Wert größer als Eins erhöht. Solange der Schwellenwert Eins beträgt kann man nun sicher sein, daß das Muster einen Output im ersten Layer erzeugen wird. Als nächstes wird im zweiten Layer das Eingangsgewicht des jeweiligen Signals auf Eins erhöht um somit auch hier einen Output zu erzeugen.
Wird ein Muster falsch klassifiziert, so muß das Netzwerk zurückgetrimmt werden, indem man die Gewichte bei den aktiven Signalen dahingehend verringert, daß kein Output mehr erzeugt wird. Diese Lernprozedur funktioniert nur, solange nicht allzuviele überschneidungen bei den verschiedenen Mustern auftreten.

Back-Propagation Networks

Die Strategie dieses Multi-Layer Netzwerks ist es, den Fehlerbeitrag von jedem Pfad im Netzwerk zu ermitteln und weiterzuverwenden.
Die Lernregeln für die vorderen Layer beinhalten die Gewichte der Layer, die eher weiter hinten liegen. Dadurch werden in den vorderen Layern kleinere Gewichtsänderungen bewirkt. Es wird angenommen, daß gemäß der „Fundamental Pattern Classification Strategy" die Gewichte der hinteren Layer stärkeren Einfluß auf den Fehler haben, als die der vorderen. Während hybride Netzwerke nur zwei Layer für diese Strategie benützen, sind es bei den Back-Propagation Netzwerken mehrere.
Daraus resultiert ein sehr langsames Lernen, das sogar mehere Tausend Iterationen benötigen kann, um ein paar Muster zu lernen.

Adaptive Resonance (ART) Networks

Das ART Netzwerk ist nach dem Original Perceptron die zweite Generation von Regelmäßigkeits-Detektoren. Man unterscheidet hier zwei Typen:

- ART 1 (benutzt binäre Eingangssignale)
- ART 2 (benutzt analoge Eingangssignale)

Sie benutzen die „Convergent Subcircuit Dynamic Allocation Strategy", welche dem Muster eine neue konvergente Teilschaltung zuweist, wenn keine der anderen Teilschaltungen eine hinreichende Übereinstimmung findet. Das Lernen findet also statt, wenn für ein Muster keine Übereinstimmung gefunden wird. Das Netzwerk löst in diesem Fall einige Resets aus, um verschiedene Teilschaltungen zu testen. Dies geschieht so lange, bis eine andere Schaltung gefunden wird, die es erlaubt die Gewichte in der letztendlich ausgewählten konvergenten Teilschaltung zu ändern.

Kapitel 2

Anwendungsbeispiel

2.1 Projektziele

Ziel dieses Projektes ist die prinzipielle Entwicklung einer schnellen Objekterkennung für den am Institut für Computerwissenschaften entwickelten Roboter E.M.M.A. Die Aufgabe dieses Roboters besteht in der Entwicklung einer, gegenüber bereits im Einsatz befindlichen Robotern, besseren Strategie im Fussballspielen. Dabei muß der Roboter prinzipiell zwischen drei Objekten unterscheiden:

- Dem Gegner (Zylinderform),
- dem Ball (Kugel),
- und dem Tor (Rechteck).

Diese drei Objekte unterscheiden sich im weiteren durch verschiedene Farben. Der Roboter selbst erhält ein Bild der Außenwelt durch eine Videokamera.
Aus diesem Bild ergeben sich nun folgende Aufgaben:

- Objektsegmentierung
- Objektvorverarbeitung
- Klassifizierung der Objekte
- Anwenden von ANNs zur Lösung der oben genannten Probleme

Bei der Objektsegmentierung müssen die einzelnen Objekte von anderen abgegrenzt und herausgelöst werden. Anschließend muß das Objekt in einen Feature-Raum übertragen werden, um aus diesen nun die verschiedenen Objekte klassifizieren zu können. Dies soll nun möglichst mit Hilfe eines ANNs verwirklicht werden. Allerdings ist zu überlegen, ob sich dadurch nicht eher ein Geschwindigkeitsnachteil ergibt, da sich die einzelnen Objekte durch eher einfache Merkmale unterscheiden (wie Farbe, einfache geometrische Form, etc.) Eine weitere Anwendungsmöglichkeit der ANNs würde in der Objektsegmentierung bestehen, die ansonsten sicherlich den größten Zeitanteil der gesamten Bildverarbeitung benötigen würde.
Wir wollen in diesem Projekt versuchen Aufgaben, wie Objektvorverarbeitung mit eigenen Programmen zu lösen. Bei der Anwendung der ANNs wiederum möchten wir den Stuttgarter Neuronale Netze Simulator (SNNS) verwenden.

2.2 Der Stuttgarter Neuronale Netze Simulator (SNNS)

Bevor eine Entscheidung getroffen werden kann, wie der SNNS am besten in unsere Projektziele eingebunden werden kann, ist eine genaue Auseinandersetzung mit der Funktionsweise dieses Simulators notwendig. Ein Thema, das bei der Behandlung einer optimalen Einbindung des SNNS auftritt, wäre zum Beispiel die Frage welcher Typ von ANNs für welche Anwendung eingesetzt werden soll usw.
Weiters soll dadurch auch abgeklärt werden, ob eine sinnvolle Verwendung der im Simulator entwickelten und trainierten ANNs im Hinblick auf die Effizienz (begrenzter Speicherkapazität und Rechenleistung des Roboters) überhaupt möglich ist.

2.2.1 Auswahlkriterien

Im SNNS können verschiedene Typen von ANNs simuliert werden. Daraus ergibt sich der Vorteil, daß ein idealer Typ bereits in der Simulationsphase ermittelt werden kann. Auch kann das Netz hier bereits trainiert werden und somit bereits mit „Vorwissen" auf das reale System übertragen werden. Dort muß man sich nun nicht mehr um Trainingsalgorithmen kümmern, weshalb dadurch wiederum Resourcen eingespart werden können.
Dieses Übertragen des ANNs auf das reale System (in diesem speziellen Fall auf den Roboter) benötigt allerdings eine bequeme Schnittstelle. Im SNNS wird dies dadurch gelöst, indem man eine Datei im ASCII-Format erstellen kann, die das ANN enthält. Für diese Datei steht nun wiederum ein Compiler (SNNS2C) bereit, der diese in eine C-Datei umwandelt, die nun leicht in die Programmierumgebung des Roboters eingebunden werden kann.
Der Simulator wird nun weiters klar in folgende Gebiete eingeteilt [11]:

- Simulator kernel
- Graphical user interface
- Batch execution interface: Batchman
- Networkcompiler SNNS2C

Mit dem graphischen Benutzerinterface, das auf dem Kernel aufsetzt, kann der aktuelle Zustandes des ANNs jederzeit abgelesen werden. Die Aufgabe von Batchman besteht in der Erzeugung eines Batch-Skripts des entwickelten ANNs.

2.2.2 Begriffe im Umgang mit dem SNNS

In diesem Unterkapitel soll geklärt werden, wie die zuvor in der Theorie besprochenen Begriffe zum Thema ANNs nun im Simulator gedeutet und eingesetzt werden. Dabei werden die im Simulator eingesetzten englischen Begriffe absichtlich nicht übersetzt, um später im konkreten Umgang mit dem Simulator Mißverständnisse zu vermeiden.

Allgemeines

Im SNNS besteht ein ANN prinzipiell aus zwei Komponenten:

- Units

- Links

Unter einer Unit versteht man nichts anderes als eine Nervenzelle oder ein Neuron bzw. deren Kern. Genau wie in der Biologie besitzt diese Eingänge, die mit den gewichteten Ausgängen der vorgeschalteten Zelle verbunden ist.
Die Verbindung der einzelnen Units geschieht nun mit den sogenannten Links. Wenn man sich die Units als Knoten in einem Graphen vorstellt, so kann man Links als gewichtete und gerichtete Kanten ansehen. Links stellen also nichts anderes als Axone und Synapsen dar.
Ferner wird noch unterschieden zwischen:

- Activation function und
- Output function.

Die Activation function berechnet zuerst den Eingangswert einer Unit aus den gewichteten Ausgangswerten vorgeschalteter Units. Dann berechnet sie den neuen Aktivierungswert dieses Eingangs. Im Zusammenhang mit den Eingängen ist noch der Begriff der *Sites* zu erwähnen. Dies ist nichts anderes als die Umsetzung eines Dendritenbaumes. Hiermit können Eingänge zu Gruppen zusammengefaßt und differenziert behandelt werden (jede Site besitzt eine zugehörige Funktion).
Die Output function benutzt das Ergebnis dieser Berechnung, um den Ausgangswert der betroffenen Unit zu berechnen.
Die Funktionen können als C-Programme zum Simulator-Kernel hinzugefügt werden und können weiters für jede Unit verschieden sein.

Weiters ist noch zu erwähnen, daß eine Zeiteinheit im Simulator als Berechnungschritt (update step) realisiert wird, wobei $a(t+1)$ der Schritt oder Zustand nach $a(t)$ ist.

Im folgenden wird auf die hier erwähnten Begriffe genauer eingegangen und die zugehörigen Begriffe aufgelistet.

Units

Es wird eingangs zwischen drei Arten von Unit unterschieden:

- Input Units
- Hidden Units
- Output Units

Units, die keine Eingangsverbindungen mit anderen Units besitzen werden als Input Units bezeichnet. Analog werden Units, die keine Ausgangsverbindungen haben als Output Units bezeichnet. Die Hidden Units müssen demnach zwischen den Input und den Output Units liegen und sind daher, wenn man das Netzwerk als „Black Box“ bezeichnet, von außen her unsichtbar.
Nun folgt eine Auflistung und Beschreibung wichtiger Attribute, die den Units zugeordnet werden:

- *no:* Jede Unit besitzt einen eindeutigen Schlüssel, der die Identifizierung in einem Netzwerk ermöglicht und gleichzeitig die Position im Simulator Kernel angibt.

- *name:* Jeder Unit kann ein zusätzlicher Name zugewiesen werden. Dieser Name kann Sonderzeichen und Leerzeichen enthalten, muß jedoch mit einem Buchstaben beginnen. Es empfiehlt sich, um Darstellungsprobleme zu vermeiden, kurze Namen zu verwenden, da diese im Graphical User Interface angezeigt werden.

- *io-type* oder *io:* Beschreibt die Funktion einer Unit in einem Netzwerk. Es werden folgende Werte unterschieden:

  - *input:* Input Unit
  - *output:* Output Unit
  - *dual:* Input and Output Unit
  - *hidden:* Hidden Unit
  - *special:* Gesonderte applikationsabhängige Units. Standardmäßig werden diese vom Simulator so implementiert, daß sich während dem Lernprozeß die Gewichte der Units nicht verändern.
  - *special input, special output, special hidden:* Manchmal ist es notwendig, die Position einer Unit mit dem Attribut *special* in einem Netzwerk anzugeben.

- *activation:* Der Aktivierungswert einer Unit.

- *initial activation* oder *i_act:* Initialisierungwert der einzelnen Aktivierungswerte eines Netzwerkes nach einem Reset.

- *bias:* Der Schwellenwert (Threshold) einer Unit.

- *activation function* oder *actFunc:* Funktion, die den neuen Aktivierungswert einer Unit berechnet. Allgemein kann diese Berechnung folgendermaßen dargestellt werden:

  $$a_j(t+1) = f_{act}(net_j(t), a_j(t), b_j) \tag{2.1}$$

  wobei:

  - $a_j(t)$: Aktivierungswert der Unit j im Schritt t
  - $net_j(t)$: Netzeingangwert in Unit j im Schritt t
  - b_j: Schwellenwert der Unit j

- *output function* oder *outFunc:* Diese Funktion berechnet den Ausgangswert einer Unit anhand des aktuellen Aktivierungswertes derselbigen (es ist also zwischen Aktivierungswert und Ausgangswert einer Unit zu unterscheiden). Also:

  $$o_j(t) = f_{out}(a_j(t)) \tag{2.2}$$

  Eine mögliche Output Function wäre zum Beispiel eine Funktion, die den Ausgangswert auf das Intervall $[0, 1]$ beschränkt.

- *f-type:* Hiermit kann ein sogenannter Prototyp einer Unit definiert werden, der Daten wie Activation Function, Output Function, ob Sites vorhanden sind usw.

- *position:* Gibt die Koordinaten einer Unit in einem zwei- oder dreidimensionalen ANN an.
- *subnet no:* Zusammenfassung mehrer Units zu einer einzigen.
- *layers:* Zuordnen von Units zu einem Layer, um bei der visuellen Betrachtung des Netzes die Übersicht zu erhöhen.
- *frozen:* Die Parameter von Units die dieses Attribut besitzen werden während der Simulation nicht verändert.

Links und Sites

Bei *Links* handelt es sich im allgemeinen um gerichtete Verbindungen, die den Signalverlauf innerhalb eines Netzwerkes bestimmen. Die Unit von der eine Verbindung beginnt wird *Source Unit* genannt, während die Verbindung, zu der der Link führt, *Target Unit* heißt. Weiters sind rekursive (d.h Source und Target Unit sind identisch) Links erlaubt. Jedem Link wird eine Gewichtung (*Weight*) zugeordnet. Diese bestimmt ob es sich bei einem bestimmten Link um eine excitatorische (Weight ist positiv) oder um eine inhibitorische (Weight ist negativ) Verbingung handelt.
Eine Unit mit Sites hat keine direkten Eingänge mehr. Alle gewichteten Ausgangssignale vorgeschalteter Units werden durch eine benutzerdefinierte Site-Funktion verändert, woraus sich eine neues Eingangssignal ergibt. Bei Sites ist es im Gegensatz zu einer normalen Verbindung erlaubt, daß ein Ausgang einer Unit auf verschiedene Sites derselben nachgeschalteten Unit geführt wird.

Update Modi

Im folgenden wird nun erläutert wie die einzelnen Aktivierungswerte der Units neu berechnet werden. Im allgemeinen gibt es zwei Arten von Update Modi:

- Synchrones Update der Units
- Asynchrones Update der Units (hier gibt es vier Unterarten)

Insgesamt wird also zwischen fünf Update Modi unterschieden:

- *Synchronous:* Hier werden in jedem Schritt alle Units neu berechnet. Dazu werden zuerst alle Aktivierungswerte der Units anhand der Input-Werte ermittelt und erst anschließend die Output-Werte berechnet. In der Realität würde das bedeuten, daß alle Units gleichzeitig feuern.
- *Random Permutation:* Hier erfolgt die Berechnung der Aktivierungs- und Output-Werte sequentiell, wobei die Reihenfolge der gewählten Units zufällig ist. Es wird allerdings garantiert, daß jede Unit in jedem Schritt genau einmal behandelt wird.
- *Random:* Berechnung erfolgt wie bei *Random Permutation*. Allerdings können Units hier auch mehrere Male (oder auch gar nicht) während eines Schrittes zur Berechnung herangezogen werden.

- *Serial:* Die Reihenfolge der Units wird durch die interne Unit Nummer (no) festgelegt. Hierbei handelt es sich um den schnellsten Update Modus.

- *Topological:* Der Kernel sortiert die Units anhand ihrer Topologie, um die Reihenfolge der Berechung zu ermitteln. Wenn es sich bei dem verwendeten ANN um ein rein unidirektionales Netzwerk handelt, so erfolgen diese Berechnungen sehr schnell, da sich viele Output-Werte der Units nicht mehr ändern. Außerdem kann dieser Modus auch durchaus mit dem in der Natur verwendeten Modus verglichen werden.

Es gibt noch weitere 11 verschiedene Update Modi für einige spezielle Netzwerktheorien, die hier allerdings nicht besprochen werden.

Lernen in ANNs

Eine wesentliche Fragen im Zusammenhang mit ANNs ist die, wie die Gewichte der einzelnen Links so verändert bzw. trainiert werden, sodaß das Netzwerk das gewünschte Verhalten zeigt. Im Allgemeinen wird dabei wiederum, die im vorigen Kapitel besprochene Hebbsche Regel verwendet. Diese lautet nun allgemein:

$$\Delta w_{ij} = g(a_j(t), t_j)h(o_i(t), w_{ij}) \tag{2.3}$$

wobei:

- Δw_{ij} : Gewichtung des Links von Unit i zu Unit j.

- $a_j(t)$: Aktivierung von Unit j in Schritt t.

- t_j : Der gewünschte Output-Wert der Unit j.

- $o_i(t)$: Output-Wert der Unit i in Schritt t.

Beim Lernvorgang werden nun zwei Lernphasen unterschieden:

- *forward propagation phase:* Hier wird ein Muster (Pattern) an das Netzwerk angelegt und abgewartet bis die Aktvierung den Output-Layer erreicht hat.

- *backward propagation phase:* Nun wird die Output-Wert mit dem gewünschten Wert (*Teaching Input* t_i) verglichen und nach obiger Formel eine Gewichtungsänderung für die entsprechenden Links berechnet. Für die Berechnung der Gewichungsänderung zwischen Hidden-Layers wird eine spezielle Formel (siehe weiter unten) verwendet, die jeweils den Fehler (Differenz zwischen Output-Wert und Teaching Input) der nachfolgenden Verbindung benützt.

Ferner wird noch unterschieden zwischen:

- *Online Learning*: Die Gewichtungsänderung Δw_{ij} wird nach jedem Pattern sofort auf das Netz angewendet.

- *Offline Learning* oder *Batch Learning*: Die Gewichtungsänderungen werden in einer Trainings-Datei gesammelt und erst dann als Summe auf des Netzwerk angewendet.

Der am meisten verwendete Algorithmus ist der *Backpropagation* oder *generalized delta-rule*-Algorithmus. Dieser wird durch folgende Formel beschrieben:

$$\Delta w_{ij} = \eta \delta_j o_j$$

$$\delta_j = \begin{cases} f_j(net_j)(t_j - o_i) & \text{wenn Unit } j \text{ eine Output Unit,} \\ f_j(net_j) \sum_k \delta_k w_{jk} & \text{wenn Unit } j \text{ eine Hidden Unit ist.} \end{cases}$$

wobei:

- η: Lernfaktor Eta (Konstante)
- δ_j: Fehler von Unit j
- o_i: Output-Wert der vorgeschalteten Unit j
- i: Index der vorgeschalteten Unit i
- j: Index der aktuellen Unit j
- k: Index der nachgeschalteten Unit k
- w_{ij}: Gewichtung des Links von Unit i zu Unit j
- w_{jk}: Gewichtung des Links von Unit j zu Unit k

Generalisierung in einem ANN

Einer der Hauptvorteile eines neuronalen Netzes ist das Zuordnen von Mustern zu einer Klasse, wobei das entsprechende Muster vorher noch nie mit dem ANN in Berührung gekommen ist. Vorher muß dazu natürlich das Netz auf eine solche Klasse mit entsprechenden, zu der jeweiligen Klasse zugehörigen Mustern, trainiert werden.
Im SNNS wird dieser Vorgang in drei Einheiten unterteilt:

1. *Training Set*: Hier wird das Netz mit Mustern trainiert und so der Fehler der Output-Werte minimiert.
2. *Validation Set*: Bestimmt die Fehlerrate, von Mustern mit denen das Netzwerk noch nichts zu tun hatte, die sich aber sehrwohl einer Klasse zuordnen lassen.
3. *Test Set*: Hier wird das gesamte Netzwerk noch einer allumfassenden Prüfung unterzogen.

Es ist weiters zu beachten, daß das Lernen gestoppt werden soll, sobald der Fehler im Validation Set ein Minimum erreicht hat. Sonst kann es passieren, daß das Netzwerk „übertrainiert“ wird, und die Erkennrate im Validation Set wieder abnimmt.
Validation Set und Training Set werden parallel ausgeführt. Sind beide abgeschlossen so soll das Netzwerk mit dem Test Set nocheinmal überprüft werden.

2.3 Einsatz des SNNS bei der Bilderkennung

Wir haben uns nun entschieden den SNNS konkret bei der Positionsbestimmung des Balles im Sichtfeld der EMMA zu verwenden. Wir wollen den Einsatz von ANNs auf die Ballerkennung beschrnken, da die anderen Probleme (wie Tor und Gegnererkennung) durch heuristische Methoden leichter lösbar sind.
Ziel dieses Versuchsaufbaus zur Ballerkennung ist ferner herauszufinden wie sich ein ANN mit einer groen Anzahl von Input-Neuronen verhlt bzw. sich überhaupt trainieren läßt und wie es sich bei untrainierten Input Patterns verhält.

2.3.1 Aufnahme der Trainingsdaten

Als Trainingsdaten wurden ca. 340 Bilder mit Hilfe der eingebauten Kamera der EMMA aufgenommen. Dabei wurden möglichst viele verschiedene Spielsituationen simuliert. Im allgemeinen wird zwischen Aufnahmen ohne Ball und mit Ball, ohne Gegner und mit Gegner, ohne Tor und mit Tor sowie Kombinationen davon unterschieden. Ferner wurde auch ständig die Beleuchtung geändert, um möglichst viele Streulichtzustände zu simulieren.Auerdem wurden alle Testbilder noch einmal gespiegelt, um dadurch die Anzahl der Spielsituationen noch einmal zu erhöhen und um zu verhindern, daß das Netzwerk zu „einseitig“ lernt. Die Gesamtanzahl der der Eingabebilder beträgt somit 680 Bilder.
Diese können nun einerseits zum Trainiern des Netzes und andererseits zum Testen von unbekannten Spielsituatione verwendet werden.

2.3.2 Ermittlung der gewünschten Ausgabedaten

Die nächsten Aufgabe besteht darin die gesammelten Daten entsprechend auszuwerten (sprich Ermittlung der gewünschten Output-Patterns). Als Output wurde die x-Position des Balles bestimmt. D.h. Pixel die den Ball darstellen werden senkrecht auf die x-Achse projiziert und stellen im Output-Pattern eine 1 dar, während die restlichen Pixel mit einer 0 markiert werden. Neben dem Winkel zwischen Emma und Ball kann somit auch der Abstand des Balles von der EMMA näherungsweise ermittel werden, indem man die Anzahl der 1er im Output-Pattern als Ma für die Entfernung des Balles nimmt. Es ist allerdings darauf zu achten, daß dabei ein Spezialfall zu untersuchen ist: Befindet sich nämlich der Ball nicht vollständig im Sichtfeld der EMMA so wird die Abstand und Winkelberechnung verfälscht. Dieser Fehler muß anschließend im Anwendungsprogramm durch entsprechende Plausibilitätsabfragen korrigiert werden.
Die Auflösung der EMMA-Bilder beträgt minimal 80x60 mit 64 Graustufen. Um die Eingabedaten etwas zu minimieren wurde die Auflösung auf 40x30 reduziert.

2.3.3 Erzeugung der Patternfiles

Um ein Netzwerk im SNNS trainieren zu können müssen Testdaten in einem speziellen Patterfile zusammengefaßt werden. Dabei handelt es sich um ein standardisiertes Dateiformat das hier anhand eines Beispiels erlautert wird:

```
SNNS pattern definition file V3.2
generated at Mon Apr 25 18:08:50 1994
```

```
No. of patterns : 26
No. of input units : 35
No. of output units : 26

# Input pattern 1:
0 1 1 1 0 1 0 0 0 1
1 0 0 0 1 1 1 1 1 1
1 0 0 0 1 1 0 0 0 1
1 0 0 0 1
# Output pattern 1:
1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0
# Input pattern 2:
1 1 1 1 0 1 0 0 0 1
1 0 0 0 1 1 1 1 1 0
1 0 0 0 1 1 0 0 0 1
1 1 1 1 0
# Output pattern 2:
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0

etc.
```

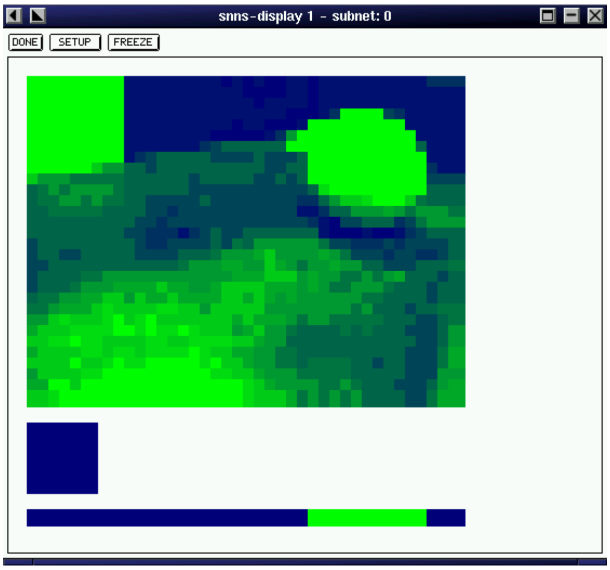
Es ist darauf zu achten, daß die Anzahl der Zeichen im Header-Teil unverändert bleibt (natürlich mit Ausnahme der Zeichenlänge der darin vorkommenden Zahlen). Weiters muß auf ein Input-Pattern immer das entsprechende Output-Pattern folgen. Die einzelnen Pattern sind weiters mit einem `#` zu trennen und der Text nach einem solchen Zeichen wird als Kommentar ausgewertet.
Um den Zeitaufwand der Auswertung der großen Anzahl der Bilder zu minimieren, haben wir ein Programm entwickelt, daß das Erzeugen einer solchen Pattern-Datei halbautomatisiert. Dabei werden die einzelnen Kamerabilder am Bildschirm eines PCs dargestellt. Mit einem Cursor, kann man sich nun bequem innerhalb des Bildes herumbewegen und den Anfang und das Ende der zu setzenden Pixel (wie oben beschrieben) entsprechend markieren. Als Output kann nun ein Teil eines Patternfiles in eine Datei gespeichert werden. Durch ein zusätzliches Programm werden nun diese Dateien nun wiederum zusammengefaßt und in ein entgültiges Patternfile geschrieben, das die oben genannten Konventionen erfült. Diese Datei kann somit im SNNS geladen werden, um damit ein Netzwerk zu trainieren.

2.3.4 Ermittlung der optimalen Netzwerkstruktur

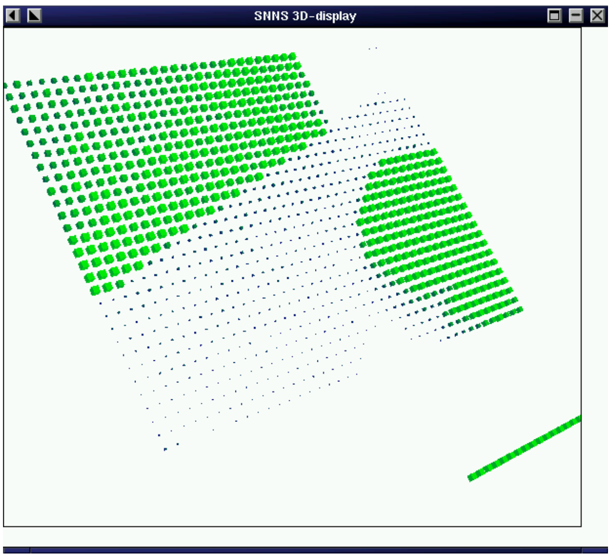
Bei derart großen ANNs, wie wir sie aufgrund der großen Input-Layer-Schicht verwenden, ist eine große Rechenleistung notwendig um ein solches Netzwerk zu trainieren. Die Anzahl der Inputneuronen beträgt versuchsbedingt 1200 Neuronen; die Anzahl der Outputneuro-

nen jedoch nur 40.
Es gilt nun die optimale Anzahl der Hidden-Layer-Schichten, sowie die Anzahl der Neuronen pro Schicht zu ermitteln. Auerdem kann man die Anzahl der Links reduzieren, was einen zusätzlichen Zeitgewinn bei den Berechnungen darstellt. Das reduzieren den Anzahl von Neuronen hat noch den zusätzlichen Vorteil, daß bei einem fertig trainiertem ANN die Berechnung der Ouput-Neuronen anhand des Eingangszustandes beschleunigt wird (siehe Zielsetzung einer Fast Object Recognition)
Die Berechnung dieser Optimierung erfolgt über das Programm NETGEN, das aus einem vorgegebenen Patternfile eine optimale Netzstruktur evolviert. Diese Berechnungen sind allerdings ebenfalls einigermaßen zeitaufwendig und werden daher auf unserem Institut auf mehrere Rechner verteilt, um das Netzwerk somit zum frühestmöglichen Zeitpunkt verwenden zu könnenn.

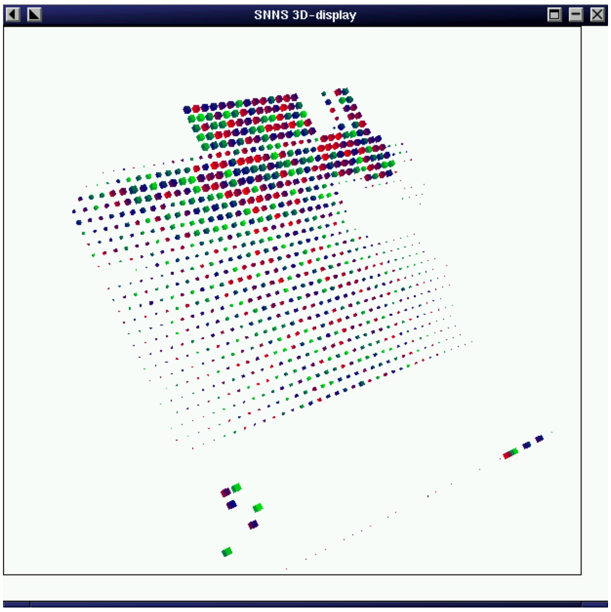
2.3.5 Beispielbilder bei der Entwicklung des ANNs



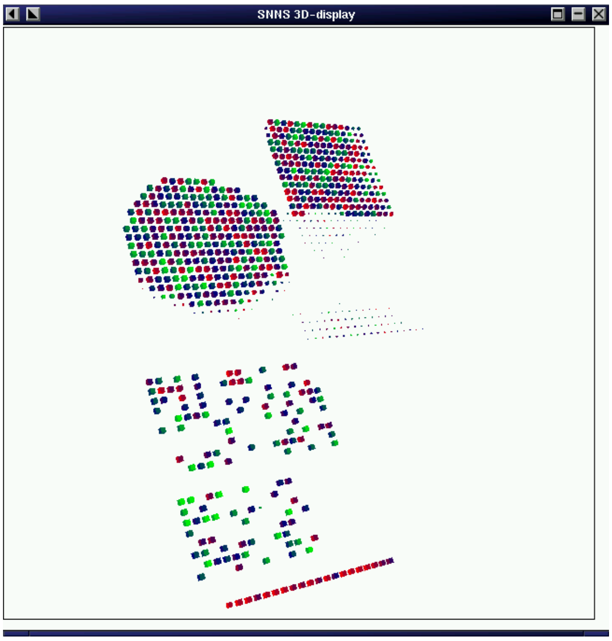
ANN zur Erkennung des Balls



3D Ansicht des ANN im SNNS



Ansicht des trainierten ANNs



Ansicht des trainierten ANNs

2.4 Segmentierung

Bei der Segmentierung sollten die einzelnen Objekte von ihrer Umwelt abgegrenzt werden. Dazu müssen sich die Objekte von der Umgebung unterscheiden. Es können dazu verschiedene Kriterien verwendet werden (siehe 1.2.1).
Eine ist das Erkennen von Objekten durch ihre Konturen: Zu diesem Zweck wurde ein einfaches Programm in der Programmiersprache C geschrieben. Die Funktion `follow_border` (siehe B.2.5) verfolgt die Kanten eines Objekts in dem als Parameter übergebenen Bild `pic`.
Die Funktion wird das erste Mal aufgerufen, wenn in pic ein Bildpunkt gefunden wird, der

sich von der Hintergrundfarbe (`BG_COLOR`) unterscheidet. Es werden dann die acht benachbarten Punkte auf ihren Farbwert hin überprueft. Besitzen alle acht Punkte den gleichen Farbwert, so wird abgebrochen, da es sich dann um keinen Punkt auf einer Objektkante handeln kann. Ansonsten wird der Punkt in der Matrix `u` als Kantenpunkt markiert und die Funktion für die acht benachbarten Punkte rekursiv aufegrufen.
Auf diese Weise erhält man nun die Kanten von allen sich im Bild befindlichen Objekten. Dieses Verfahren benötigt eindeutig erkennbare Objekte mit einem sehr großen Kontrastunterschied. Das heißt das sich Objekte vom Hintergrund durch ihre Helligkeit stark abheben. In der Praxis ist dies aber kaum der Fall (siehe A.1). Aus diesem Grund müßen die Kanten vorher verstärkt werden. Zu diesem Zweck wurden folgende Verfahren getestet:

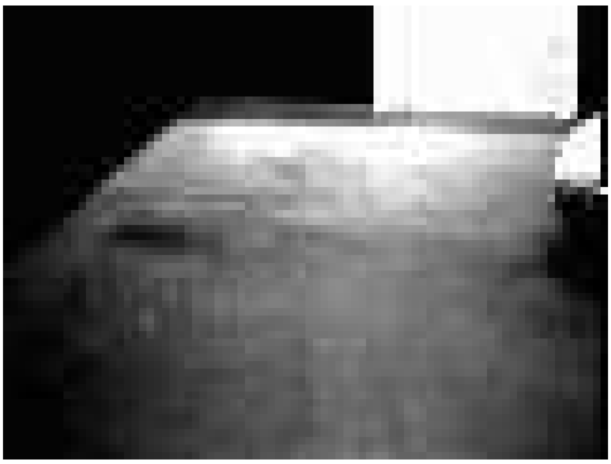
- *Sobel Edge Detection Algorithmus* Dieser Algorithmus verstärkt die Kanten innerhalb des Bildes.

- *Kirsch Edge Detection Algorithmus* Wurde verwendet um vertikale (oder horizontale) Kanten im Bild zu erkennen.

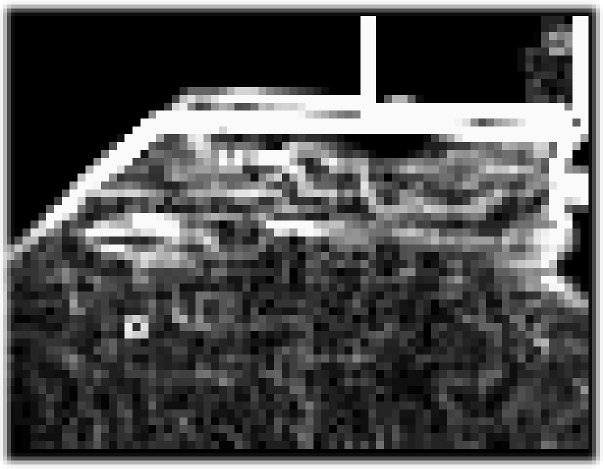
- *Hough Transformation* Transformiert das Bild in einen Parameter Raum (Polardarstellung). Für diesen Raum wird nun einen Schwellwert festgelegt und dann die Rücktransformation durchgeführt [12]. Mit dieser Methode konnten allerdings bei den verwendeten Testbildern (siehe A.1) keine brauchbaren Informationen gewonnen werden.

2.4.1 Rauschreduktion

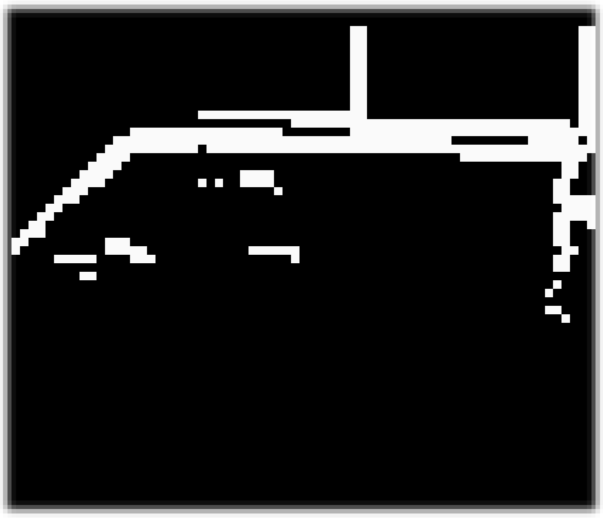
Vor Verwendung eines Kanten Erkennungs Algorithmus ist es oft sinnvoll die Bilder zu glätten. Besonders bei Bildern von Kameras ist das Bild oft verrauscht. Das Rauschen wird durch den verwendeten Algorithmus oft als Kante erkannt. Durch eine vorhergehende Mittelwertberechnung (siehe rauschfilterprog) der umgebenden Bildpunkte jedes Punktes kann das Rauschen stark reduziert werden und somit das Ergebnis der Kantenerkennung verbessert werden.



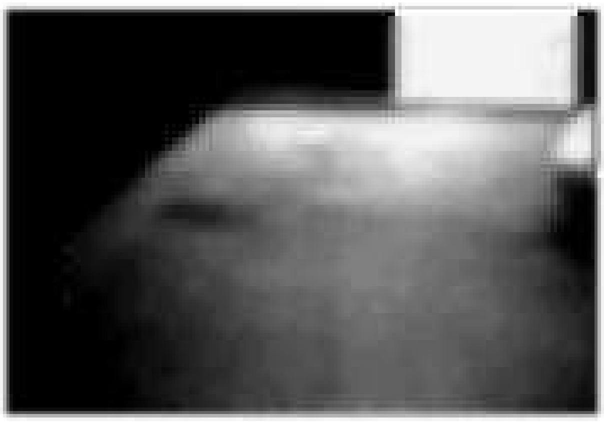
Ungefiltertes Bild



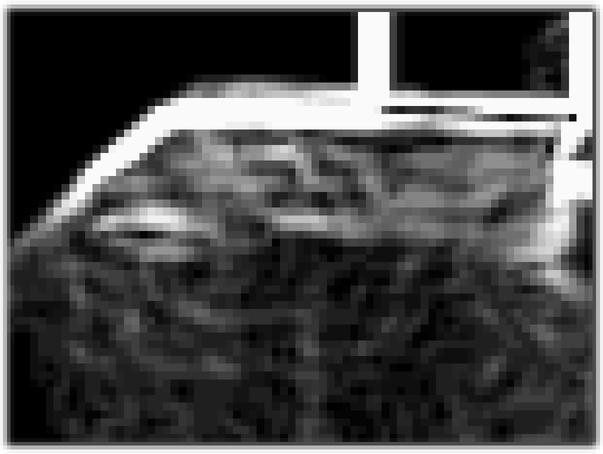
Ungefiltertes Bild nach dem Sobel Algorithmus



Ungefiltertes Bild nach dem Sobel Algorithmus und nach Schwellwert Triggerung
Deutlich erkennbar sind Teile der Bodentextur die fälschlicherweise als Kanten erkannt wurden.



Gefiltertes Bild



Gefiltertes Bild nach dem Sobel Algorithmus

Gefiltertes Bild nach dem Sobel Algorithmus und nach Schwellwert Triggerung
Im Gegensatz zum ungefilterten Bild entfallen hier die Artefakte der Bodentextur

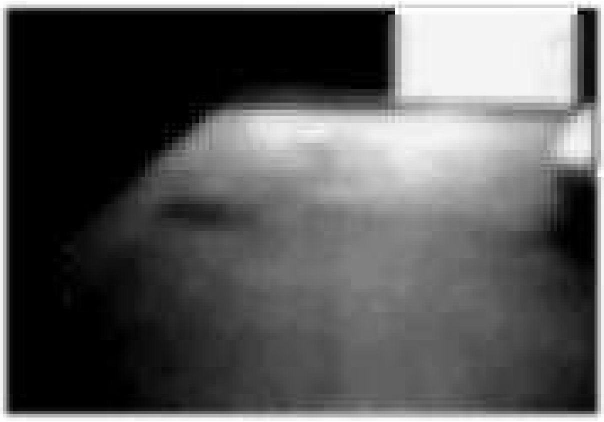
Am obigen Beispiel kann man erkennen das eine Filterung (Rauschreduktion) des Bildes für die folgenden Verarbeitungschritte durchaus Vorteile bringt. So werden z.B. beim ungefilterten Bild Kanten aus der Bodentextur extrahiert.

2.4.2 Sobel Edge Detection Algorithmus

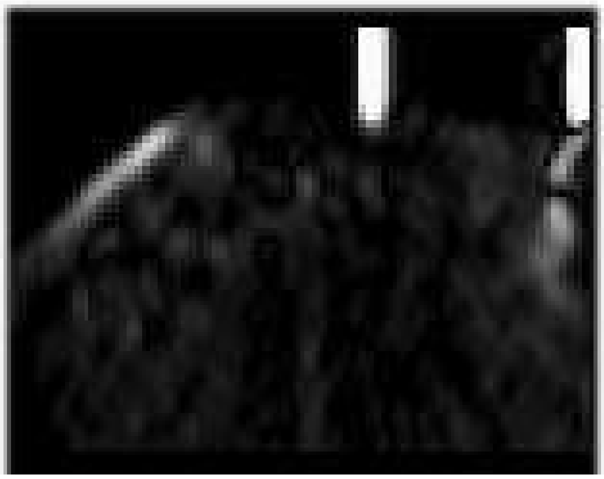
Dieser Algorithmus verwendet zwei 3x3 Masken um das Bild zu filtern. Dabei werden die Kanten im Bild stark hervorgehoben und andere Bildelemente unterdrückt [12]. Beispiele für die Kantenerkennung via Sobel Edge Detection siehe 2.4.1. Die verwendete Implementierung ist in B.2.2 abgedruckt.

2.4.3 Kirsch Edge Detection Algorithmus

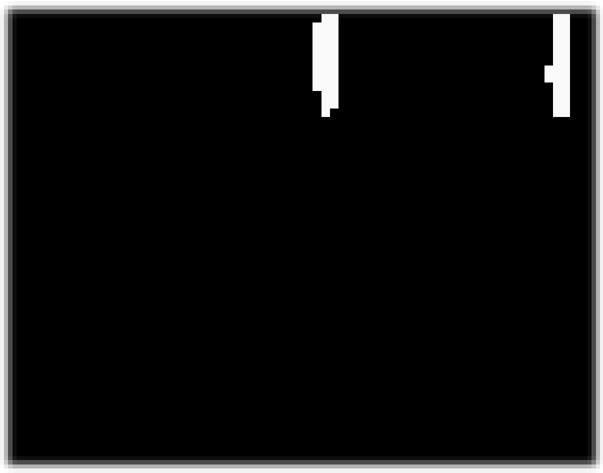
Die Kirsch templates wurden verwendet um gerichtete Kanten im Bild zu finden. In unserem Fall wurde nach vertikalen Kanten gesucht [12]. Das Tor wird hierbei beispielsweise durch zwei parallele vertikale Kanten dargestellt. Das für die folgenden Bilder verwendete Programm ist in B.2.3 abgedruckt.



Gefiltertes Bild



Gefiltertes Bild nach dem Kirsch Algorithmus



Gefiltertes Bild nach dem Kirsch Algorithmus und nach der Schwellenwert Triggerung

2.4.4 Situationsspezifische Merkmale des Bildes

In dem verwendeten Beispielszenario der Bilderkennung für die EMMA können verschiedene Situationsspezifische Merkmale verwendet werden um eine schnelle Bilderkennung zu ermöglichen.

In diesem Szenario müssen nur drei verschiedene Objekte erkannt werden.

- Ball,
- Tor und
- Gegner

Folgende Situationsspezifische Merkmale ergeben sich:

- Das **Tor** befindet sich immer am oberen Bildrand. Dies wird durch die Kammeraeinstellung sichergestellt. Die Farbe des Tors ist Grau. Da der Hintergrund sehr dunkel ist und die Kammera eine automatische Helligkeitsregelung verwendet ergeben sich sehr hohe Helligkeitswerte für das Tor.
- Der **Gegner** befindet sich ebenfalls immer am oberen Bildrand. Auch dies wird durch den Einstellwinkel der Kammera sichergestellt.

Es ergeben sich unter anderem folgende Möglichkeit um diese spezifischen Merkmale für eine schnelle Objekterkennung zu nutzen:
Man summiert für jede x Koordinate des Bildes die Helligkeit aller zugehörigen Pixel der Spalte auf. Man erhält so einen horizontalen Helligkeitsverlauf über das Bild. An den Stellen an denen sich das Tor, der Ball oder der Gegner befindet erhält man sehr hohe Werte. Wenn man nun nicht die ganzen Spalten aufsummiert sondern nur den oberen Bereich des Bildes verwendet kann man auf diese weise relativ gut das Tor erkennen.

Kapitel 3

Abschlußbemerkungen

Die Verwendung des gesamten Bildes in einem Netz mit entsprechend vielen Input Neuronen erscheint realisierbar. Die Aufteilung der Erkennungsaufgabe in mehrere Teilnetze kann für zukünftige Versuche sinnvoll sein. Eine mögliche Aufteilung wäre hierbei ein Teilnetz das nur Bilder mit Ball lernt und daraus die Richtung des Balles bestimmen kann und ein Netz das nur lernt ob im Bild der Ball vorhanden ist oder nicht. Durch Kombination dieser Netzte sollte die Fehlerwarscheinlichkeit verringert werden koennen.
Die Möglichkeit einer Vorverarbeitung wie z.B. eine Kantenextraktion vor der Analyse durch das Netz sollte ebenfalls noch genauer untersucht werden.
Am praxistauglichsten dürfte aber dennoch eine Objekterkennung sein welche die spezifischen Eigenschaften der Situation verwendet. Diese dürfte in der Praxis auch am schnellsten sein, da fertig trainierte ANNs zwar schnell durchgerechnet werden können, aber bei einer sehr hohen Zahl an Verbindungen die benötigte Rechenzeit, sowie der benötigte Speicherbefarf trotzdem stark ansteigt.

Anhang A

Bilder

A.1 Bilder der EMMA Quickcam

Bild 1

Bild 2

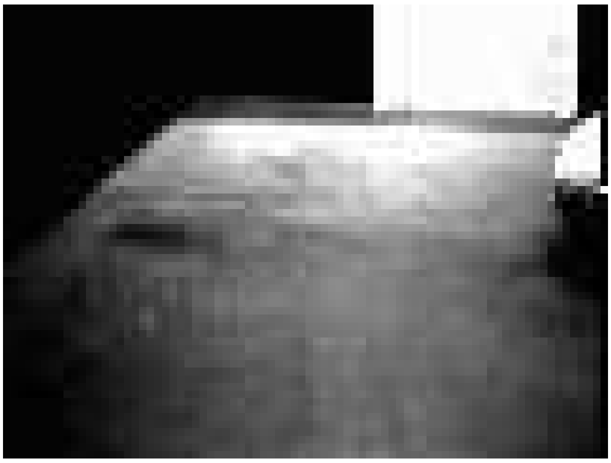


Bild 3

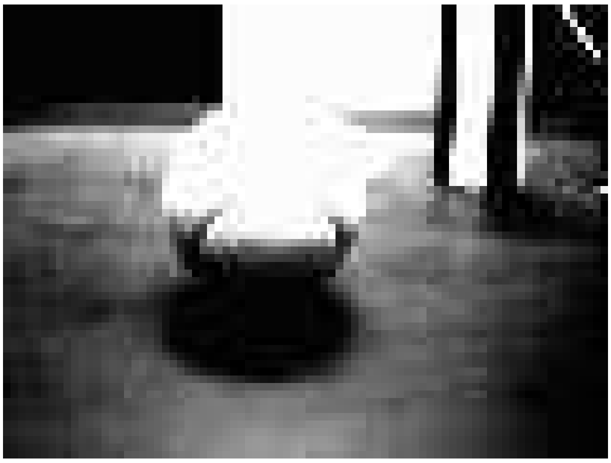


Bild 4

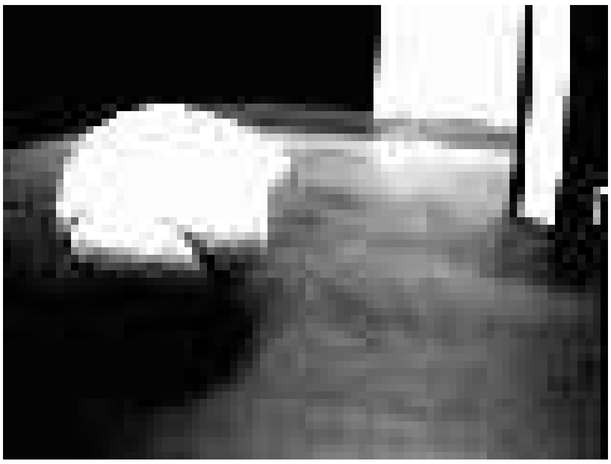


Bild 5

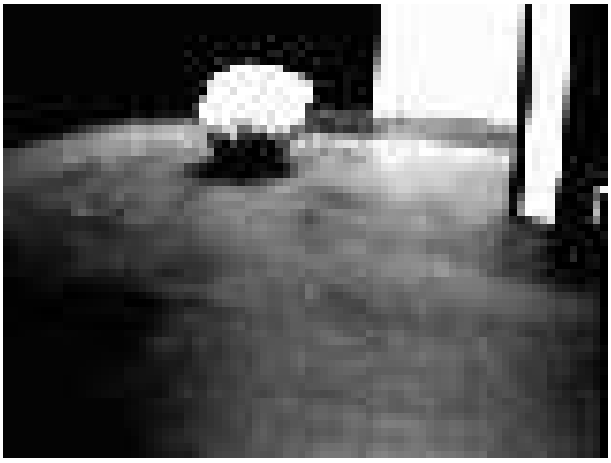


Bild 6

Anhang B

Programme

B.1 Beispielprogramm zur Feature Extraktion

```
/*Berechnung der Steigung der Funktion (von 10% der Amplitude auf 90% der
  Amplitude)
  Eingangsdaten in input[] */

  // Glaettung der Kurve mittels eines einfachen (nicht exakt brechneten)
  // digitalen Tiefpassfilters

  for(i=256;i<512;i++)
  {
    setcolor(12);

    // Filterkoeffizienten sind gegebenenfalls einer speziellen
    // Charakteristik anzupassen
      output[i%256]=(input[(i-0)%256]*0.14+
      input[(i-1)%256]*0.13+
      input[(i-2)%256]*0.12+
      input[(i-3)%256]*0.11+
      input[(i-4)%256]*0.10+
      input[(i-5)%256]*0.09+
      input[(i-6)%256]*0.08+
      input[(i-7)%256]*0.07+
      input[(i-8)%256]*0.06+
      input[(i-9)%256]*0.05+
      input[(i-10)%256]*0.04+
      input[(i-11)%256]*0.03+
      input[(i-12)%256]*0.02+
      input[(i-13)%256]*0.02+
      input[(i-14)%256]*0.01+
      input[(i-15)%256]*0.01)/1.1;

    // Zeichnen der geglaetteten Kurve
    putpixel(i%256+321,300-output[i%256]*40,10);
```

```
    // Suchen des min und max der Kurve
    if(output[i%256]<mi) mi=output[i%256];
    if(output[i%256]>ma) ma=output[i%256];
  }

  state=0;
  for(i=8;i<256;i++)
  {
    // 10 % Wert suchen
    if(output[i]>(mi+(ma-mi)*0.1) && state==0)
    {
      // Merken und einzeichnen
      state=1;
      i10=i;
      setcolor(4);
      setlinestyle(4,0xAAAA,1);

      line(321,300-output[i]*40,321+255,300-output[i]*40);
      line(321+i,300-80,321+i,300+80);
    }
    // 90% Wert suchen
    if(output[i]>(mi+(ma-mi)*0.9) && state==1)
    {
      // Merken und einzeichnen
      state=2;
      i90=i;
      setcolor(4);
      setlinestyle(4,0xAAAA,1);

      line(321,300-output[i]*40,321+255,300-output[i]*40);
      line(321+i,300-80,321+i,300+80);
    }
  }
  // Steigung einzeichnen
  setlinestyle(0,0,3);
  setcolor(12);
  line(321+i10,300-output[i10]*40,321+i90,300-output[i90]*40);

  // Steigung berechnen und ausgeben
  gotoxy(40,25);
  printf("STEIGUNG/SAMPLE: %f  \n",((output[i90]-output[i10])/(i90-i10)));

/*************************************************************************/

/* Diskrete Fourier Transformation
   Eingangsdaten = Samplingwerte
   Ausgangsdaten = Amplitude von Cn
```

```
*/
void dft(float *in, float *out, int n)
{
  int m=n,k,i;
  float pm=(3.1415926*2)/(float)m;
  float p,s=0,t=0;

  // Diskrete Fourier Transformation
  for(k=0;k<m;k++)
  {
    for(i=0;i<m-1;i++)
    {
      p=i*k*pm;
      s+=in[i]*cos(p);
      t+=in[i]*sin(p);
    }
    // Brechnung der Amplitude aus Sinus und Cosinus Anteil der Cn.
    out[k]=sqrt((s/(float)m)*(s/(float)m)+(t/(float)m)*(t/(float)m));
    s=t=0;
  }
}
```

B.2 Segmentierung

B.2.1 Algorithmus zur Glättung des Bildes

```
void noisefilter(image bild)
{
 for(y=1;y<IMAGEY-1;y++)
    for(x=1;x<IMAGEX-1;x++)
      bild[x][y]=(bild[x-1][y-1]+bild[x-1][y]+bild[x-1][y+1]+
  bild[x  ][y-1]+bild[x  ][y]+bild[x  ][y+1]+
  bild[x+1][y-1]+bild[x+1][y]+bild[x+1][y+1])/9;
}
```

B.2.2 Sobel Edge Detector

```
int sobel( image a, image b, int N1, int M1, int N2, int M2)
{
  int i,j,c,cc;

  for(i=N1+1; i<N2-1; i++)
    for(j=M1+1;j<M2-1;j++)
    {
      cc =-a[i-1][j-1] -2*a[i-1][j] - a[i-1][j+1];
      cc+= a[i+1][j-1] +2*a[i+1][j] + a[i+1][j+1];
```

```
      c=abs(cc);
      cc =-a[i-1][j-1] -2*a[i][j-1] - a[i+1][j-1];
      cc+= a[i-1][j+1] +2*a[i][j+1] + a[i+1][j+1];
      c+=abs(cc);
      if(c>63) c=63; // 64 Graustufen
      b[i][j]=c;
    }
}
```

B.2.3 Kirsch Edge Detector

```
/* Kirsch template fuer vertikale Kanten */
int kirsch( image a, image b, int N1, int M1, int N2, int M2)
{
  int i,j,c,cc;

  for(i=N1+1; i<N2-1; i++)
    for(j=M1+1;j<M2-1;j++)
    {
      cc =-a[i-1][j-1] - a[i-1][j] - a[i-1][j+1];
      cc+= a[i+1][j-1] + a[i+1][j] + a[i+1][j+1];
      c=abs(cc);
      if(c>63) c=63;
      b[i][j]=c;
    }
}
```

B.2.4 Hough Transformation

```
/* Hough Transformation nach "Digital Image Processing Algorithms,
   Ioannis Pitas, Prentice Hall, 1993" */

int hough(image xin,imatrix p,int n,int m,int N1, int N2)
{
  int k,l,i,j,kk,ll;
  float r,b;
  float SQRTD=sqrt((float)N1*(float)N1+(float)N2*(float)N2);

  for(kk=0;kk<n;kk++)
    for(ll=0;ll<m;ll++)
      p[kk][ll]=0;

  for(k=0;k<N1;k++)
    for(l=0;l<N2;l++)
    {
      if(xin[k][l]==1)
      {
```

```
for(i=0;i<n;i++)
{
  r=k*COS[i]+l*SIN[i]; b=SQRTD;
  r+=b; r/=(SQRTD*2.0); r*=(m-1); r+=0.5;
  j=floor(r); p[i][j]++;
}
      }
    }
  return 0;
}

int ihough(image xin, image xout, imatrix p, int N1, int N2, int n, int m)
{
  int k,l,i,j;
  float r,y;
  float SQRTD=sqrt((float)N1*(float)N1+(float)N2*(float)N2);

  for(i=0;i<N1;i++)
    for(j=0;j<N2;j++)
      xout[i][y]=0;

  for(k=0;k<n;k++)
    for(l=0;l<m;l++)
      {
y=(float)0.0;
if(p[k][l]==1)
{
  for(i=0;i<N1;i++)
  {
    r=(float) l *2.0 *SQRTD/(m-1) -SQRTD;
    if(SIN[k]==0.0)
      y++;
    else
      y=(r-(float)i*COS[k])/SIN[k];
    y+=0.5; j=floor(y);
    if (j>=0 && j<N2)
      if(xin[i][j]==1) xout[i][j]=50;
  }
}
     }
}

int look_up_table(int m)
{
  int i;
  float th;
  float R_TO_D=0.017453;
```

```
  for(i=0;i<m;i++)
  {
    th=(float)i*180.0/(m-1)-90.0; th=th*R_TO_D;
    COS[i]=(double)cos((double)th);
    SIN[i]=(double)sin((double)th);
  }
}
```

B.2.5 Kantenfolgungs Algorithmus

```
int follow_border(int pic[PIC_SIZE][PIC_SIZE], char u[PIC_SIZE][PIC_SIZE],
                  int x, int y)
{
   int i;

   if (u[x][y]==1 || pic[x][y]==BG_COLOR) return 0;

   if (x>0 && x<(PIC_SIZE-1) && (y>0) && y<(PIC_SIZE-1))
      { i= (pic[x-1][y-1]!=BG_COLOR)?0:1;
        i+=(pic[x-1][y  ]!=BG_COLOR)?0:1;
        i+=(pic[x-1][y+1]!=BG_COLOR)?0:1;
        i+=(pic[x  ][y-1]!=BG_COLOR)?0:1;
        i+=(pic[x  ][y+1]!=BG_COLOR)?0:1;
        i+=(pic[x+1][y-1]!=BG_COLOR)?0:1;
        i+=(pic[x+1][y  ]!=BG_COLOR)?0:1;
        i+=(pic[x+1][y+1]!=BG_COLOR)?0:1;

        if (i==8 || i==0) return 0;

        putpixel(PIC_POS_X+x,PIC_POS_Y+y,10);
        u[x][y]=1;

        if (u[x-1][y-1]!=1 || pic[x-1][y-1]!=BG_COLOR)
           follow_border(pic,u,x-1,y-1);
        if (u[x-1][y  ]!=1 || pic[x-1][y  ]!=BG_COLOR)
           follow_border(pic,u,x-1,y  );
        if (u[x-1][y+1]!=1 || pic[x-1][y+1]!=BG_COLOR)
           follow_border(pic,u,x-1,y+1);
        if (u[x  ][y-1]!=1 || pic[x  ][y-1]!=BG_COLOR)
           follow_border(pic,u,x  ,y-1);
        if (u[x  ][y+1]!=1 || pic[x  ][y+1]!=BG_COLOR)
           follow_border(pic,u,x  ,y+1);
        if (u[x+1][y-1]!=1 || pic[x+1][y-1]!=BG_COLOR)
           follow_border(pic,u,x+1,y-1);
        if (u[x+1][y  ]!=1 || pic[x+1][y  ]!=BG_COLOR)
```

```
          follow_border(pic,u,x+1,y  );
        if (u[x+1][y+1]!=1 || pic[x+1][y+1]!=BG_COLOR)
          follow_border(pic,u,x+1,y+1);

        return 1;
      }

   else return 0;
}
```

B.3 Tools zur Erzeugung und Bearbeitung der Pattern Files

B.3.1 view

Programm zum generieren der gewünschten Output Patterns

```
#include <stdlib.h>
#include <dos.h>
#include <stdio.h>
#include <conio.h>

void main(int argc, char *argv[])
{
   int i, j, c, a, e, mrk1, mrk2, cnt;
   int image[80][60];
   unsigned far char *screen = (unsigned char *) MK_FP(0xA000, 0);
   FILE *data;

   if (argc==1) exit(1);
   else data=fopen(argv[1] ,"rb");

   for (j=0; j<60; j++)
      { for (i=0; i<80; i++)
{ image[i][j]=getc(data);
}
      }
   fclose(data);

   mrk1=0;
   do { mrk1++; } while(argv[1][mrk1] != '.');

   argv[1][mrk1+1]='o';
   argv[1][mrk1+2]='u';
   argv[1][mrk1+3]='t';
```

```
   asm{ mov ax, 0x13
int 0x10
      }

   for (i=128; i<256; i++)
      { outp(0x3c8, i);
outp(0x3c9, i/2);
outp(0x3c9, i/2);
outp(0x3c9, i/2);
      }

   for (i=0; i<80; i++)
      { for (j=0; j<60; j++)
{ screen[i+320*j]=128+image[i][j]/2;
  if ((i%2==0) && (j%2==0)) screen[120+i/2+320*j/2]=128+image[i][j]/2;
}
      }

   mrk1=mrk2=a=e=0;
   i=j=1;

   gotoxy(16,6);
   printf("%2d  %2d", i, j);
   gotoxy(23,1);
   printf("Anfang: %2d", a);
   gotoxy(23,2);
   printf("Ende:   %2d", e);

   screen[120]=255-screen[120]+128;

   do { c=getch();
if ((c=='h') && (i>1)) { mrk1=i; mrk2=j; i-=1; }
else if ((c=='k') && (i<40)) { mrk1=i; mrk2=j; i+=1; }
else if ((c=='u') && (j>1)) { mrk1=i; mrk2=j; j-=1; }
else if ((c=='j') && (j<30)) { mrk1=i; mrk2=j; j+=1;}
else if (c=='a') { a=i; }
else if (c=='e') { e=i; }
else if ((c=='w'))
   { data=fopen(argv[1] ,"w");
     for (cnt=1; cnt<=40; cnt++)
{ if ((cnt>=a) && (cnt<=e)) putc('1', data);
  else putc('0', data);
}
     fclose(data);
     gotoxy(23,4);
     printf("*");
     delay(500);
```

```
      gotoxy(23,4);
      printf(" ");
    }

if (mrk1 || mrk2)
    { screen[120+(mrk1-1)+320*(mrk2-1)]=255-screen[120+(mrk1-1)+320*(mrk2-1)]+128;
      screen[120+(i-1)+320*(j-1)]=255-screen[120+(i-1)+320*(j-1)]+128;
      mrk1=mrk2=0;
    }
gotoxy(16,6);
printf("%2d  %2d", i, j);
gotoxy(31,1);
printf("%2d", a);
gotoxy(31,2);
printf("%2d", e);
      } while (c != 27);

   asm{ mov ax, 0x03
int 0x10
      }
}
```

B.3.2 bmp242pic

Wandelt eine Truecolor BMP Datei in das intern verwendete pic Format um. Dieses Format ist ein Bitstream in dem jedes Pixel des Bildes durch einen 8 Bit Helligkeitswert dargestellt wird. Die Auflösung beträgt 80x60 Pixel.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
 FILE *in,*out;
 int i,j,k;
 unsigned char r,g,b,c;

 unsigned char buffer[255];
 unsigned char pic[100][100];

 printf("BMP 24 bit => pic converter\n");

 if(argc<2) exit(1);

 strcpy(buffer,argv[1]);
 printf("in: %s - ",buffer);

 in=fopen(buffer,"rb");
```

```
 buffer[strlen(buffer)-3]='p';
 buffer[strlen(buffer)-2]='i';
 buffer[strlen(buffer)-1]='c';
 printf("out: %s\n",buffer);
 out=fopen(buffer,"wb");

 //printf("reading header...\n");
 for(i=0;i<54;i++) fgetc(in);

 //printf("image...\n");
 for(i=0;i<60;i++)
 {
   for(j=0;j<80;j++)
   {
     r=fgetc(in);
     g=fgetc(in);
     b=fgetc(in);
     pic[79-j][59-i]=r;
   }
 }

 for(i=0;i<60;i++)
  for(j=0;j<80;j++)
   fputc(pic[j][i],out);

 fclose(in);
 fclose(out);

}
```

B.3.3 pic2eps

Wandelt die intern verwendeten pic Dateien in das eps Format um. Damit können diese leicht in anderen Programmen (z.B. LaTeX) verwendet werden.

```
#include <stdio.h>
#include <string.h>

char *epsheader[]={"%!PS-Adobe-1.0 EPSF-1.2",
                   "%%BoundingBox: 0 0 80 60",
                   "%%Creator: pic2eps",
                   "%%Title: converted 80x60 grayscale picture",
                   "%%CreationDate: 0",
                   "%%EndComments",
                   "/width 80 def",
```

```
                    "/height 60 def",
                    "/pixwidth 80 def",
                    "/pixheight 60 def",
                    "/picstr width string def",
                    "/emmapic {",
                    "gsave width height 8",
                    "[width 0 0 height 0 height neg]",
                    "{currentfile picstr readhexstring pop}",
                    "image grestore } def",
                    "0 height neg translate pixwidth pixheight scale",
                    "emmapic",
                    "" };

void main(int argc, char *argv[])
{
  FILE *in, *out;

  int x,y,i;
  unsigned char c;
  char buffer[100];
  unsigned char bild[80][60];

  printf("pic2eps converter, josef schmidbauer, 1999\n");
  if(argc<2 || argc>3)
  {
    printf("usage: %s picfile [epsfile]\n",argv[0]);
    exit(1);
  }

  printf("in: %s ",argv[1]);
  in=fopen(argv[1],"rb");

  if(argc==2)
  {
    strcpy(buffer,argv[1]);
    i=strlen(argv[1]);
    if(buffer[i-4]=='.' && buffer[i-3]=='p' && buffer[i-2]=='i' && buffer[i-1]=='c')
    {
      buffer[i-3]='e';
      buffer[i-2]='p';
      buffer[i-1]='s';
    }
    else
    {
      strcat(buffer,".eps");
    }
  }
```

```
  else
    strcpy(buffer,argv[2]);

  printf("out: %s \n",buffer);

  out=fopen(buffer,"wa");


  i=0;
  while(strlen(epsheader[i])>0) fprintf(out,"%s\n",epsheader[i++]);

  for(y=0;y<60;y++)
    for(x=0;x<80;x++)
      bild[x][y]=getc(in);

  for(y=0;y<60;y++)
  {
    for(x=0;x<80;x++)
    {
      c=bild[x][59-y];
      sprintf(buffer,"000%X",c);
      fprintf(out,"%s",&buffer[strlen(buffer)-2]);
    }
    fprintf(out,"\n");
    if(y%2) printf(".");
  }

  fprintf(out,"%%Trailer\n");
  fclose(out);
  fclose(in);
  printf("done\n");
}
```

B.3.4 patdiv

Teilt die lst Datei für das `makepatt` Programm in 3 Teile ein. Damit kann man aus einer lst Datei die lst Dateien fuer 3 verschiednde Patternsets erzeugen um so unabhängige Trainings-, Test- und Validierungssets zu erstellen.

```
#include <stdio.h>

void main()
{
  FILE *in, *out1, *out2, *out3;

  int i;
  char buffer[255];
```

```
  printf("\"patterns geteilt durch 3\"(tm) programm\n");
  printf("input: patterns.lst output: pattern1.lst pattern2.lst pattern3.lst\n");


  in=fopen("patterns.lst","ra");
  if(in==NULL)
  {
    fprintf(stderr,"error: patterns.lst not found\n");
    exit(1);
  }

  printf("dividing patterns...\n");
  out1=fopen("pattern1.lst","wa");
  out2=fopen("pattern2.lst","wa");
  out3=fopen("pattern3.lst","wa");

  i=0;
  while(fgets(buffer,255,in))
  {
    i++;
    if(i%3==0) fputs(buffer,out1);
    if(i%3==1) fputs(buffer,out2);
    if(i%3==2) fputs(buffer,out3);
    putchar('.');
  }
  printf("\ndone\n");


}
```

B.3.5 makepatt

Erzeugt aus einer lst Datei welche die Dateinamen der zu verwendenden Patterns enthält, und den jeweiligen pic und out Dateien ein Patternsset für den SNNS.

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#define VALUES_PER_LINE 15

void main(int argc, int *argv[])
{
   int i, j, k, l;
   int value;
   int mirror=0;
```

```
   int number=0;
   static int image[80][60];
   int out[40];
   char pic[50], pat[50];
   FILE *list, *input, *output;

   if ( argc>1 && (unsigned char)argv[1][0]==(unsigned char)'s')
   {
     mirror=1;
   }

   output=fopen("emma.pat", "wb");

   list=fopen("patterns.lst", "r");
   if (list == NULL)
      { printf("\nDatei \"patterns.lst\" nicht gefunden!\n\n");
exit(1);
      }
   while (fgets(pic, 50, list) != 0) number++;
   fclose(list);

   list=fopen("patterns.lst", "r");

   fprintf(output, "SNNS pattern definition file V3.2\n");
   fprintf(output, "generated at Sat Oct 29 23:04:00 1977\n\n\n");
   if (mirror) fprintf(output, "No. of patterns : %d\n", 2*number);
   else fprintf(output, "No. of patterns : %d\n", number);
   fprintf(output, "No. of input units : 1200\n");
   fprintf(output, "No. of output units : 40\n\n");

   i=1;
   printf("\n\n\nGenerating SNNS Pattern Definition File \"emma.pat\"...\n\n");
   while (fgets(pic, 50, list) != 0)
      { fprintf(output, "# Input pattern %d:\n", i);

j=0;
do { j++; } while(pic[j] != '.');
pic[j+4]='\0';

input=fopen(pic, "rb");
if (input == NULL)
   { printf("\nDatei \"%s\" nicht gefunden!\n\n", pic);
     exit(2);
   }
else printf("Processing \"%s\"...\n", pic);

for (k=0; k<60; k++)
```

```
   { for (j=0; j<80; j++)
{ image[j][k]=getc(input);
}
   }

l=0;
for (k=0; k<30; k++)
   { for (j=0; j<40; j++)
{ if (l==VALUES_PER_LINE) { l=0; fprintf(output, "\n"); }
  fprintf(output, "%1.2f ", (float)(image[j*2][k*2])/256);
  l++;
}
   }

fclose(input);
fprintf(output, "\n");

fprintf(output, "# Output pattern %d:\n", i);
j=0;
        strcpy(pat, pic);
do { j++; } while(pat[j] != '.');

pat[j+1]='o';
pat[j+2]='u';
pat[j+3]='t';

input=fopen(pat, "rt");
if (input == NULL)
   { printf("\nDatei \"%s\" nicht gefunden!\n\n", pat);
     exit(3);
   }
else printf("Processing \"%s\"...\n", pat);

for (j=0; j<40; j++)
   { out[j]=getc(input); }
l=0;
for (j=0; j<40; j++)
   { if (l==VALUES_PER_LINE*2) { l=0; putc('\n', output); }
     putc(out[j], output);
     putc(' ', output);
     l++;
   }

fclose(input);
fprintf(output, "\n");

i++;
```

```
if (mirror)
   { fprintf(output, "# Input pattern %d:\n", i);

     input=fopen(pic, "rb");
     printf("Processing mirrored \"%s\"...\n", pic);

     l=0;
     for (k=0; k<30; k++)
{ for (j=39; j>=0; j--)
     { if (l==VALUES_PER_LINE) { l=0; fprintf(output, "\n"); }
       fprintf(output, "%1.2f ", (float)(image[j*2][k*2])/256);
       l++;
     }
}

     fclose(input);
     fprintf(output, "\n");


     fprintf(output, "# Output pattern %d:\n", i);

     input=fopen(pat, "rt");
     printf("Processing mirrored \"%s\"...\n", pat);

     l=0;
     for (j=39; j>=0; j--)
{ if (l==VALUES_PER_LINE*2) { l=0; putc('\n', output); }
  putc(out[j], output);
  putc(' ', output);
  l++;
}

     fclose(input);
     fprintf(output, "\n");

     i++;
   }
      }
   fclose(list);
   fclose(output);
}
```

Anhang C

Protokolle

C.1 09.03.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Stieglbauer Gerald
Ort: Büro von Schwaiger Roland
Zeit: 17.55 - 18.10

Einführende Besprechung über das Projekt und erste Aufgabenstellung

Spezifikation des abzugebenden Projektberichtes:

- Zu verwendendes Satzsystem: LaTeX
- Eigene Titelseite
- Abstract (Übersicht)
- Einleitung (Um was geht es?)
- Literaturübersicht
- Literaturverzeichnis

Abgabe der Projektberichte jeweils bis Montag 13.00 Uhr

Erste Fragestellung:

- Was ist Object Recognition?

C.2 16.03.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Schmidbauer Josef
Ort: Büro von Schwaiger Roland
Zeit: 17.45 - 18.05

Vorzunehmende Verbesserungen am Paper:

- Angaben auf der Titelseite um Lehrveranstaltung, Version, Zeit, Gruppe, E-Mail-Adressen und Lehrveranstaltungsleiter ergänzen
- Abstract hinzufügen
- Einführung von Abkürzungen für häufig verwendete Begriffe
- Angaben im Protokoll um Zeit, Ort und teilnehmende Personen ergänzen
- Abgabe via E-Mail mit Subject „Bioinformatik, Gruppe 3"

Aufgaben bis zum 23.03.1999:

- Suche nach Beschreibung für „Object Recognition" in anderen Büchern
- Suche nach weiteren Verfahren zur Objekterkennung
- Ausführlichere Beschreibung der statistischen Objekterkennung
- Warum ist die statistische Objekterkennung so wichtig?
- Stufen der Vorverarbeitung (Pre-Processing) eines Bildes,
  Analyse in einem anderen Abbildungsraum

C.3 23.03.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Stieglbauer Gerald
Ort: Büro von Schwaiger Roland
Zeit: 17.45 - 18.10

Vorzunehmende Verbesserungen am Paper:

- E-Mail-Adressen auf Titelseite unter die Namen schreiben

- Datum als Versionsnummer

Aufgaben bis zum 13.04.1999:

- Abgrenzung der Objekte von der Umwelt, Image Segmentation
- Was sind überwachte Methoden bei dem statistischen Verfahren? (Beispiel)
- Ansätze für die Transformation in den Feature-Raum
- Was sind wichtige Eigenschaften bei der Ermittlung der Features von Objekten?
- Was bedeuten Nachbarschaftsbeziehungen im Feature-Raum?

C.4 13.04.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Stieglbauer Gerald
Ort: Büro von Schwaiger Roland
Zeit: 17.45 - 18.10

Vorzunehmende Verbesserungen am Paper:

- Auflistung im Punkt „Erkennung von Objekten in Bildern“ unvollständig
- Textur im Punkt „Abgrenzen der Objekte von ihrer Umwelt“ ungenau

Aufgaben bis zum 20.04.1999:

- Was sind überwachte und unüberwachte Methoden bei dem statistischen Verfahren? (Beispiel)
- Beispiel für Feature Space und Extraktion
- Übersicht über Neuronale Netze

C.5 20.04.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Obermair Christoph
Ort: Büro von Schwaiger Roland
Zeit: 17.45 - 17.55

Aufgaben bis zum 04.05.1999:

- Weitere Beispiele für neuronale Netze
- Beispiel für Objekterkennung entwerfen und durcharbeiten

Am 27.04.1999 findet, aufgrund einer Videovorführung im Rahmen des Pro-Seminars keine Besprechung statt.

C.6 04.05.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Schmidbauer Josef
Ort: Büro von Schwaiger Roland
Zeit: 17.45 - 17.55

Die Besprechungen finden bis zum Semsterende nur noch alle zwei Wochen statt.
Termine: 18.5., 1.6., 15.6. und 29.6.
In der 1. Juliwoche wird von jeder Pro-Seminar-Gruppe ein Vortrag über ihre Arbeit gehalten (ca. 15 Minuten, 5 Folien).

Aufgabe bis zum Ende des Semesters:

- Realisieren des in 2.1 angegebenen Beispiels

C.7 18.05.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Stieglbauer Gerald
Ort: Büro von Schwaiger Roland
Zeit: 17.45 - 17.55

Für das Testen und Anwenden der Programme stehen nun Testbilder in Form von Spielsituationen der EMMA zur Verfügung.

Aufgaben bis 01.06.1999:

- Vertiefung der Begriffe im Umgang mit SNNS.
- Verbesserung der Programme zur Extraktion von Objekten.
- Ermittlung von Features für Objekte.

C.8 01.06.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Obermair Christoph
Ort: Büro von Schwaiger Roland
Zeit: 17.45 - 17.55

Es sollte die Möglichkeit in Betracht gezogen werden, die Bilder selbst in ein neuronales Netz zu stecken.

Aufgaben bis 15.06.1999:

- Gammaverlauf anschauen
- Bilder rastern und als Input für neuronales Netz verwenden
- Geeigneten Output des Netzes überlegen

C.9 15.06.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Schmidbauer Josef
Ort: Büro von Schwaiger Roland
Zeit: 17.45 - 17.55

Es soll ein neuronales Netz entworfen werden, welches als Output die Richtung des Balls angibt.

Aufgaben bis 29.06.1999:

- EMMA-Bilder in neuronalem Netz verwenden
- Aufnehmen von weiteren EMMA-Bildern

C.10 29.06.1999

Teilnehmende Personen: Schwaiger Roland
Obermair Christoph
Schmidbauer Josef
Stieglbauer Gerald

Protokollführer: Schmidbauer Josef
Ort: Büro von Schwaiger Roland
Zeit: 18.00 - 18.15

Besprechung der Präsentation

Literaturverzeichnis

[1] W. E. L. Grimson, *Object Recognition by Computer: The Role of Geometric Constraints.* The MIT Press, 1990.

[2] G. Kleiter, *Skriptum zur VL Allgemeine Psychologie I.* Institut fuer Psychologie, Universitaet Salzburg, 1998.

[3] C. H. Chen, L. F. Pau, and P. S. P. Wang, *Handbook of Pattern Recognition and Computer Vision.* World Scientific Publishing, 1993.

[4] A. D. Kulkarni, *Artificial Neural Networks for Image Understanding.* Van Nostrand Reinhold, 1994.

[5] C. M. Bishop, *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, fourth ed., 1997.

[6] E. Herter and W. Loercher, *Nachrichtentechnik.* Carl Hanser Verlag, seventh ed., 1994.

[7] R. Schalkoff, *Pattern Recognition: Statistical, Structural and Neural Approaches.* John Wiley and Sons Inc., 1992.

[8] N. Hofmann, *Kleines Handbuch Neuronale Netze.* Vieweg, 1993.

[9] D. O. Hebb, *The Organization of Behavior.* John Wiley and Sons Inc., 1949.

[10] B. Farley and W. Clark, *Simulation of Self-Organizing Systems by Digital Computers.* IRE Transactions on Information Theory, 1954.

[11] A. Zell, G. Mamier, M. Vogt, and N. Mache, *Stuttgart Neuronal Network Simulator, User Manual Version 4.2.* Univeristy of Stuttgart, University of Tuebingen.

[12] I. Pitas, *Digital Image Processing Algorithms.* Prentice Hall, 1993.