Vergleich neuronaler und konventioneller Sakkadensteuerungen unter Realweltbedingungen

Diplomarbeit
am Lehrstuhl für Kognitive Systeme
des Instituts für Informatik und Praktische Mathematik
der Christian–Albrechts–Universität zu Kiel

vorgelegt von
Lars Riehn

Betreuer: Dipl. Inform. J. Bruske
Dipl. Phys. M. Hansen
Prof. Dr. G. Sommer

August 1996

Zusammenfassung

Gegenstand dieser Arbeit ist die Realisierung einer *Sakkadensteuerung* auf einem *binokularen Kamerakopf*. Dabei werden *monokulare* und *binokulare* Sakkaden untersucht.
Um die Sakkadensteuerung in *Echtzeit* zu implementieren, kann die Sakkade nicht auf *visuellem Feedback* basieren. In dieser Arbeit wird daher das Problem der *inversen Kinematik* sowohl analytisch als auch durch einen Lernansatz gelöst.

Diese Arbeit vergleicht eine konventionelle Sakkadensteuerung mittels *kalibrierter* Kamera, sowie *neuronale* Steuerungsansätze. Im Rahmen der neuronalen Ansätze wurden *Kohonenkarten* und *Dynamische Zell-Strukturen* (DCS) verwendet und verglichen.

Die Vorteile der Dynamischen Zell-Strukturen gegenüber den Kohonenkarten bestehen in dem selbständigen Aufbau der *Topologie* während einer *inkrementellen Wachstumsphase*. Die Topologie wird an den für die Funktionsapproximation relevanten Unterraum angepasst. Durch das Einfügen von Neuronen in Regionen hohen Fehlers wird ein kleinstmögliches Netzwerk erzeugt, dessen Approximationsfehler unter einer vorgegeben Toleranz liegt.
Diese Eigenschaft von Dynamischen Zell-Strukturen, eine fehlerabhängige Neuronenverteilung zu erzeugen, entspricht genau der Forderung, die J. Mayhew und P. Dean in ihrer Arbeit über die Implementation von neuronalen Sakkaden an eine Weiterentwicklung stellen [May91].

In der Lernphase der neuronalen Netze wird die Methode des biologisch motivierten *Error-Feedback-Lernens* zur Anwendung kommen. Sie wurde von M. Kawato vorgeschlagen und auf eine Reihe von Anwendungen erfolgreich angewendet [Kaw90a] [Kaw90b].

Das Training der neuronalen Netze findet in zwei Schritten statt. In der Off-Line Phase wird ein Kameramodell erlernt, welches grob dem verwendeten Kamerakopf Bisight der Firma TRC entspricht.
Es wird experimentell nachgewiesen, daß die Dynamischen Zell-Strukturen dabei eine Neuronenverteilung entsprechend der Minimierung des Fixationsfehlers der Sakkaden zeigen, sie bilden nicht wie Kohonenkarten eine Wahrscheilichkeitsverteilung der Eingabe.

In der On-Line Phase werden die vortrainierten Netzwerke dem realen System angepaßt. Dabei wird nur die Ausgabeschicht weiter adaptiert, die Eingabeschicht ist in der Off-Line Phase bereits fertig ausgebildet worden. Durch das Off-Line Vortraining der Netze wird die teure On-Line Trainingszeit stark reduziert. Die Ergebnisse zeigen, daß neuronale Netze in Verbindung mit Error-Feedback-Lernen hervorragend für das Sakkadenlernen geeignet sind.
Der Vergleich des neuronalen mit dem konventionellen Ansatz ergibt trotz

sorgfältiger Kalibrierung der Kameras im konventionellen Ansatz eine höhere Genauigkeit der Sakkaden mit neuronalen Netzen.

Neben dem Kamerakopf wird ein Bildverarbeitungssystem Datacube und ein Stäubli RX90 Roboter genutzt. Zusammen bilden sie ein System zum *automatisierten* Erlernen neuronaler Sakkaden.

Inhaltsverzeichnis

Kapitel 1

Einführung

In der verhaltensbezogenen Informationsverarbeitung wurden in neuester Zeit vermehrt Wege untersucht, die weg von den in der Technik vorherrschenden statischen Kameras hin zu einem *aktiven Sehen* führen (*active vision*). Aktives Sehen im Sinne von Bajcsy [Baj88] bedeutet die Steuerung des *visuellen Sensors* während der Datenaufnahme in einer für die Aufgabe geeigneten Weise. Dabei führt die gesteuerte Bewegung dazu, bestimmte Probleme der Informationsverarbeitung zu vereinfachen und eindeutige oder stabilere Lösungen zu erreichen, die für einen statischen Beobachter unterbestimmt oder instabil sind. Als Beispiele seien hier genannt: Form aus Schattierung (shape from shading), Form aus Konturen (form from contour), Form aus Texturen (form from texture) und Struktur aus Bewegung (structure from motion).
Vor diesem Hintergrund wurden die *Augenbewegungen* bei Primaten und beim Menschen seit längerer Zeit intensiv untersucht. Eine wesentliche Komponente dieser Augenbewegungen ist die *Sakkade*, die dazu dient einen interessanten Szenenpunkt in der *Fovea* der Augen zu zentrieren.
Sakkaden bilden einen Teil der ersten Stufe des aktiven Sehens, die aus der kontrollierten Steuerung der Augen- bzw. Kamerabewegungen besteht. Augengeschwindigkeiten von bis zu $600°/s$ führen dazu, daß Sakkaden zu schnell sind, um auf *visuellem Feedback* zu basieren [Car88]. Zur Berechnung schneller exakter Sakkaden ist deshalb die Lösung des *inversen Kinematik Problems* notwendig. Die Implementation von zeitoptimierten Sakkaden auf einem Kamerakopf erfolgt deshalb über die Berechnung der inversen Kinematik.

Konventionelle Systeme beschreiben hier den Weg der mathematischen Lösung des inversen Kinematik-Problems anhand eines Kameramodelles. Bei einer modellhaften Kamera werden Annahmen gemacht, die sich mit der Realität nicht exakt decken. Außerdem ist immer eine *Kalibrierung* der realen Kamera notwendig. Auf Veränderungen während der Betriebszeit kann nur mit einer Neukalibrierung reagiert werden.

Eine Lösung dieser Probleme bieten *neuronale Architekturen*. Der Bereich der künstlichen neuronalen Netze (Artificial Neural Nets: ANN), welcher aus der Analyse der biologischen Informationsverarbeitung bei Mensch und Tier hervorgegangen ist, hat in jüngster Zeit Algorithmen hervorgebracht, von denen die bekanntesten vielleicht Backpropagation, Kohonenkarten und RBF-Netze sind. Diese bieten im Gegensatz zu konventionellen informationsverarbeitenden Systemen die Möglichkeit, durch Rückkopplung eine *strukturelle Adaption* vorzunehmen. So bietet sich die Möglichkeit, mit der Zeit die gewünschten Anforderungen zu *erlernen*, ohne a priori Wissen mit einzubringen.
Die Berechnung von Sakkaden mit ANN's bietet die Möglichkeit, durch Lernen eine Kalibrierung der Kamera überflüssig zu machen. Lern- und Arbeitsphase brauchen dabei nicht entkoppelt zu sein, so daß durch den Einsatz neuronaler Netze ständig auf Veränderungen der Kamerageometrie mit einer Adaption des Netzwerkes reagiert wird.

Die Basis für das Lernen der inversen Kinematik ist *Error-Feedback-Lernen*, welches von M. Kawato vorgeschlagen wurde [Kaw90a]. Das Prinzip des Error-Feedback-Lernens wurde von Kawato erfolgreich auf eine Reihe von Aufgaben in der Robotik angewendet [Kaw90b]. Wir implementieren diese Idee, indem durch einen Proportional-Regler aus dem retinalen Fehler der Sakkade ein Fehlersignal generiert wird. Dieses wird zur Adaption der neuronalen Netze genutzt.

Ähnliche Arbeiten auf dem Gebiet der Implementation von Sakkaden auf einem binokularen Kamerakopf mit neuronalen Ansätzen wurden von J. Mayhew erstellt [May92]. Im Gegensatz zu der dort angewendeten neuronalen PILUT-Architektur werden in dieser Arbeit Kohonenkarten und Dynamische Zell-Strukturen verwendet.
Kohonenkarten wurden bei einer Vielzahl von Aufgabenstellungen erfolgreich angewendet (siehe [Rit91], [Koh95]). Dynamische Zell-Strukturen sind ein noch neuerer Ansatz, der durch den selbständigen Aufbau der Topologie in der Trainingsphase versucht, Nachteile der Kohonenkarten auszugleichen. Dynamische Zell-Strukturen erzeugen ein Netzwerk minimaler Größe, in dem der Fehler der Funktionsapproximation unter einer vorgegebenen Toleranz liegt.
Genau diese Eigenschaft wurde von J. Mayhew und P. Dean in ihrem Artikel über neuronale Sakkaden als eine zukünftige Entwicklung gefordert [May91].

Um die Bedeutung von *Sakkaden* beim menschlichen Sehen besser einschätzen zu können, wird im folgenden ein Überblick über die verschiedenen *Augenbewegungen* und ihre Eigenschaften gegeben.

1.1 Das biologische Augenmodell

Der visuelle Apparat des Menschen ist ein faszinierendes und komplexes System. Erst das unmerkliche Zusammenspiel verschiedener Mechanismen der Augenbewegungen ermöglicht uns, die Welt so exakt wahrzunehmen, wie wir es tun. Dabei werden Augenbewegungen ausgezeichnet durch hohe Funktionalität, Schnelligkeit und Anpassungsfähigkeit.
Betrachten wir hierzu die Anatomie des Augenbulbus etwas näher. Der im Fettgewebe der Augenhöhle (Orbita) eingebette Augapfel ist nach allen Richtungen beweglich. Die Bewegung wird durch sechs äußere quergestreifte Augenmuskeln ermöglicht, die in drei Paaren angeordnet sind und antagonistisch von drei Hirnnerven innerviert werden.
Die Augenbewegung gehört zu den schnellsten in der menschlichen Natur. Da der Augapfel eine kleine konstante Masse darstellt, sind Trägheitseffekte bei der Augenbewegung vernachlässigbar. Außerdem ist der Bewegungsapparat stark gedämpft, so daß Schwingbewegungen des Augapfels nicht auftreten. Man kann hier von einer direkten Umsetzung der Nervensignale in Bewegung sprechen, welche auch die hohen Bewegungsgeschwindigkeiten ermöglicht [Spe86].
Bedingt durch die hohe Geschwindigkeit kann sich das Auge ca. 95 % der Zeit in Fixationsperioden befinden [Yar67].
Das Gesichtsfeld beträgt horizontal etwa 200° und vertikal etwa 130°, dabei besitzt der Punkt des schärfsten Sehens, die *Fovea centralis* aber nur einen Durchmesser von 2°.
Man unterscheidet fünf Grundtypen der Augenbewegung, von denen wir am ausführlichsten die Sakkaden betrachten wollen [Rit91] [Töl92] :

1. Sakkaden:
   Sakkaden sind *ruckartige, konjugierte Augenbewegungen*, welche die Blickrichtung beider Augen von einem Fixationspunkt zum nächsten lenken. Sie unterliegen der willentlichen Kontrolle und können auch im Dunkeln oder mit geschlossenen Augen ausgeführt werden. Obwohl das Gesichtsfeld weit größer ist, besitzen 99 % aller Sakkaden eine Amplitude zwischen 4 Bogenminuten und 15° [Yar67]. Größere Bewegungen werden fast immer von einer Rotation des Kopfes begleitet. Sakkaden sind sehr schnelle Bewegungen, deren mittlere Winkelgeschwindigkeit mit der Amplitude ansteigt und bis zu $200-400°/s$ beträgt. Die absolute Geschwindigkeit kann bis zu $600°/s$ betragen. Die Sakkadendauer wächst näherungsweise proportional mit der Amplitude an und variiert von 30 ms bis 100 ms. Bedingt durch die hohen Geschwindigkeiten ist während der Bewegung keine Verarbeitung der visuellen Reize möglich. Zwichen dem Einsetzen des visuellen Reizes und dem Beginn der Sakkade vergeht eine *Latenzzeit* von ca. 200 ms. Ist eine Sakkade erst einmal initiiert, so wird sie zu der ursprünglich bestimmten

Zielposition ausgeführt, auch wenn sich der Reiz zwischenzeitlich bewegt hat. Nach einer Sakkade vergeht eine *Refraktärzeit* von etwa 200 ms, bevor eine neue Sakkade ausgeführt werden kann. Die horizontale und vertikale Komponente einer schräg verlaufenden Sakkade sind physiologisch voneinander unabhängig [Spe86].
Im *Superior Colliculus*, einer mehrlagigen Neuronenschicht im oberen Bereich des Hirnstammes wurden Neuronen nachgewiesen, die in ihrer topologischen Anordnung den Orten der Lichtrezeptoren der Retina entsprechen, sie bilden eine *topologische Karte* der Retina. In der benachbarten Schicht des Superior Colliculus wurden Neuronen gefunden, deren Erregungsmuster und Entladungsraten in eindeutigem Zusammenhang mit den sakkadischen Augenbewegungen stehen, sie bilden eine *motorische Karte* [Spa87].

2. Glatte Folgebewegung:
   Glatte Folgebewegungen treten bei bewegten visuellen Reizen geringer Größe auf. Sie halten diesen innerhalb eines Bereichs von ca. 2° um die Fovea centralis. Glatte Folgebewegungen können bis zu einer Objektgeschwindigkeit von $80°/s$ aufrecht gehalten werden, bei höheren Winkelgeschwindigkeiten ist die Augengeschwindigkeit deutlich langsamer, als die des Objektes, die dann auftretende *retinale Bildverschiebung* wird durch zusätzliche Sakkaden ausgeglichen.

3. Vestibular-okularer Reflex:
   Der vestibular-okulare Reflex (VOR) sorgt für eine stabilisierende und unverändert bleibende Blickrichtung der Augen bei Veränderung der Kopfposition. Die Eingangssignale dieses Reflexes entstammen aber nicht, wie vielleicht vermutet, dem visuellen System, sondern vielmehr dem *Vestibularapparat* des Innenohres.

4. Optokinetische Augenbewegungen:
   Das optokinetische Bewegungssystem stellt eine Unterstützung des VOR dar, in dem es bei Eigenbewegung zu einer visuell gestützten Stabilisierung des Fixationspunktes führt.
   Der *optokinetische Nystagmus* (OKN) ist ein periodischer Wechsel von schnellen, sakkadischen Bewegungen und langsamen Folgebewegungen, der auftritt bei der Fixation eines gleichmäßig bewegten, periodischen Reizmusters. Der OKN tritt beispielsweise bei der Betrachtung von Objekten aus einem fahrenden Eisenbahnwagon heraus auf.

5. Vergenzbewegung:
   Objekte, die sich auf einen Beobachter zubewegen oder entfernen, werden durch Vergenzbewegungen in der Fovea beider Augen gehalten. Vergenzbewegungen sind *disjunktive Augenbewegungen*, die bei einer Dauer von fast

konstant 500 ms relativ langsam sind. Die Vergenzbewegungen sind Voraussetzung für die Tiefenwahrnehmung durch *Stereopsis*.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt: Im zweiten Kapitel wird ein mathematisches Modell des verwendeten Kamerasystems eingeführt, mit dem im dritten Kapitel mathematische Lösungen der Inversen Kinematik beschrieben werden. Das vierte Kapitel ist der konventionellen Kamerakalibrierung gewidmet und zeigt erste Ergebnisse der Übertragung des mathematischen Modells auf das reale System. Das fünfte Kapitel beschäftigt sich allgemein mit neuronalen Strukturen, woraufhin im sechsten und siebenten Kapitel Kohonenkarten und Dynamische Zellstrukturen beschrieben werden. Das achte Kapitel faßt Vergleiche beim Anlernen eines Kameramodells zusammen, worauf aufbauend im neunten Kapitel Ergebnisse unter Realweltbedingungen dargestellt werden. Das letzte Kapitel widmet sich zusammenfassenden Vergleichen, der Diskussion und dem Ausblick auf die Integration dieser Arbeit.

Kapitel 2

Das Kamerasystem TRC Bisight

In diesem Abschnitt soll das verwendete *binokulare Kamerasystem* Bisight der Firma TRC beschrieben werden, sowie die direkte Abbildung vom *Referenzkoordinatensystem* der Kameraplattform zum *Bildkoordinatensystem.*

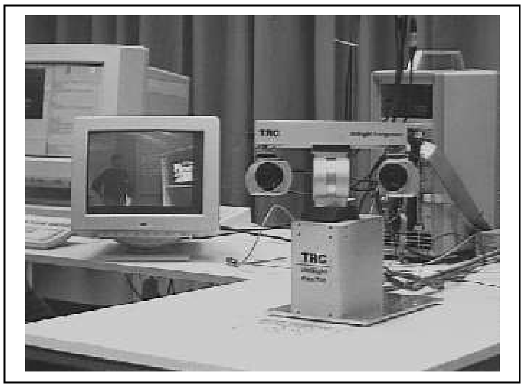


Abbildung 2.1: Das Kamerasystem TRC Bisight.

Das Sehsystem besteht neben der Kameraplattform Bisight (Abb. 2.1) aus einem Pipelinerechner MaxVideo200 der Firma Datacube, der Motorensteuerkarte PMAC der Firma DeltaTau und einer Sun Workstation (Abb. 2.2). Das Digitalisierungsverfahren liefert 512×512 Bildpunkte.

2.1 Geometrie

Das Kamerasystem verfügt über vier *mechanische Freiheitsgrade*, beschrieben durch den Schwenkwinkel χ um den Hals des Systems, den Neigungswinkel ϕ

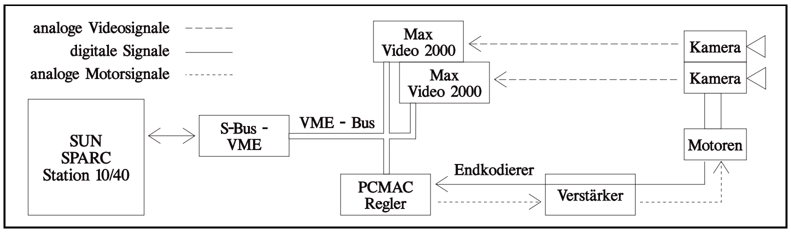


Abbildung 2.2: Architektur des Bildverarbeitungssystems (aus [DA95]).

und die Vergenzwinkel θ_l und θ_r (siehe Abb. 2.3). Die maximalen Stellgrößen betragen für den Schwenkwinkel $\pm 160°$, für den Neigungswinkel $\pm 90°$ und für die Vergenzwinkel $\pm 45°$. Auf der Neigungsachse sind die beiden CCD-Kameras montiert. Die Stereobasis wird mit b notiert und beträgt 25cm. Weiterhin verfügt das System über *optische Freiheitsgrade* wie Zoom (11.5 - 69 mm) und Iris (Blickwinkel diagonal $7 - 51°$). Die Genauigkeit der Winkelansteuerung liegt bei $\pm 0.0225°$ für den Schwenk- und den Neigungswinkel, und bei $\pm 0.006°$ für die Vergenzwinkel.

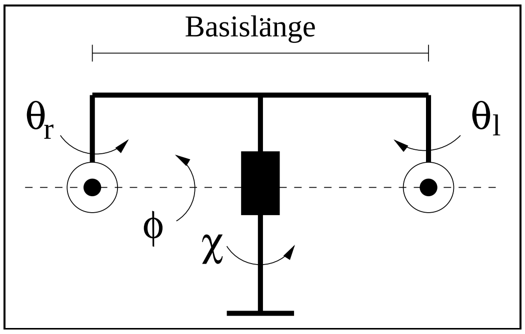


Abbildung 2.3: Geometrische Darstellung des Kamerasystems.

2.2 Kinematik

Im folgenden wird ein mathematisches Modell der Kameraplattform vorgestellt. Anhand dieses Modelles wird das inverse Kinematik-Problem gelöst, und damit die Basis für das Off-Line Training der neuronalen Netze und den Vergleich mit einem konventionellen Ansatz gelegt.

Betrachten wir die Projektion eines Weltpunktes $\vec{p}_w$ in das *Bildkoordinatensystem*: Zuerst wird der Weltpunkt in das *Kamerakoordinatensysten* der linken bzw. rechten Kamera abgebildet, um dann weiter in das Bildkoordinatensystem der jeweiligen Kamera abgebildet zu werden. Der Ursprung des Weltkoordinatensystems liegt im Schnittpunkt von Schwenk- und Neigungsachse. Die Ursprünge der beiden Kamerakoordinatensysteme liegen dabei im Schnittpunkt der Neigungsachse mit der entsprechenden Vergenzachse (Abb. 2.4).

- Abbildung in Kamerakoordinaten:
  Ein Vektor $\vec{p}_w = (x_w, y_w, z_w, 1)$ in Weltkoordinaten wird in Kamerakoordinaten $\vec{p}_c = (x_c, y_c, z_c, 1)$ abgebildet durch:

  $$\vec{p}_{c_{l/r}} = T^{-1}_{\theta_{l/r}} T^{-1}_{\phi} T^{-1}_{\chi} \vec{p}_w$$

  Dabei sind die Rotationsmatrizen gegeben durch:

  $$T_\chi = \begin{pmatrix} \cos\chi & 0 & -\sin\chi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\chi & 0 & \cos\chi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  $$T_\phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  $$T_{\theta_l} = \begin{pmatrix} \cos\theta_l & 0 & \sin\theta_l & -b/2 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_l & 0 & \cos\theta_l & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  $$T_{\theta_r} = \begin{pmatrix} \cos\theta_r & 0 & -\sin\theta_r & +b/2 \\ 0 & 1 & 0 & 0 \\ \sin\theta_r & 0 & \cos\theta_r & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Abbildung in Bildkoordinaten:
  Ein Vektor $\vec{p}_c = (x_c, y_c, z_c)$ in Kamerakoordinaten wird in Bildkoordinaten $\vec{p}_i = (x_i, y_i, 1)$ abgebildet durch:

  $$\vec{p}_i = P_c \left(\frac{\vec{p}_c}{z_c} \right)$$

Dabei ist:

$$P_c = \begin{pmatrix} \alpha_{xc} & 0 & x_{0c} \\ 0 & \alpha_{yc} & y_{0c} \\ 0 & 0 & 1 \end{pmatrix}$$

Die Parameter α_{xc}, α_{yc}, x_{0c} und y_{0c} bezeichnet man als *intrinsische Kameraparameter*. Dabei beziehen sich α_{xc} und α_{yc} auf die Brennweite der Kamera, relativ zur horizontalen bzw. vertikalen Größe der Pixel auf dem CCD-Chip. Die Parameter x_{0c} und y_{0c} beziehen sich auf die Bildmitte im Verhältnis zur Lage des CCD-Chips in der Kamera.
Neben den intrinsischen Kameraparametern existieren noch die *extrinsischen Kameraparameter*. Es sind dies die variablen Parameter der Kamerawinkel und der konstante Parameter der Basislänge.

Betrachten wir jetzt die Rückprojektion vom Bild- ins Kamerakoordinatensystem: Ist ein Vektor $\vec{p}_i$ gegeben, so kann dieser nicht eindeutig in einen Punkt im Kamerakoordinatensystem und damit auch nicht in einen eindeutigen Weltpunkt umgerechnet werden:

$$\vec{p}_c = \lambda P_c^{-1} \vec{p}_i \ ; \ \lambda \in \mathbb{R}$$

Wir führen deshalb den Begriff eines *Sehstrahls* $\lambda\ (x_w, y_w, 1)$ ein, der durch einen Punkt in der Ebene $Z = 1$ beschrieben wird.
Führt man diese Transformationen für beide Kameras durch, so schneiden sich beide Sehstrahlen im Weltpunkt $\vec{p}_w$.

2.3 Modell vs. Reale Kamera

Ein Kameramodell kann immer nur versuchen, eine reale Kamera anzunähern. Zu viele schwer zu bestimmende Parameter wären notwendig, um eine reale Kamera perfekt zu modellieren. Das vorgestellte Kameramodell macht deshalb eine Reihe von Annahmen, die in der Praxis nur annäherungsweise gegeben sind:

- Die Neigungsachse und die Vergenzachsen schneiden sich jeweils im *optischen Zentrum* der Kamera.
- Die CCD-Chips der Kameras sind optimal ausgerichtet.
- Die Basislänge stimmt exakt.

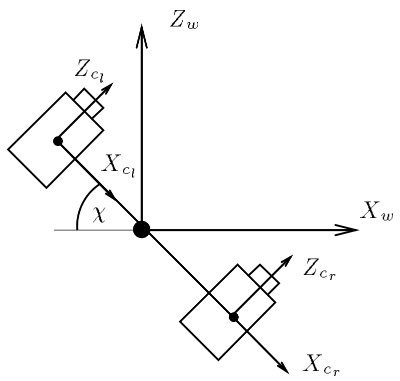


Abbildung 2.4: Blick auf die X-Z-Ebene des Weltkoordinatensystemes. Die X-Z-Ebene der Kamerakoordinatensysteme liegt nur für $\phi = 0$ in der X-Z-Ebene des Weltkoordinatensystems.

- Die intrinsischen Kameraparameter α_{xc}, α_{yc}, x_{0c} und y_{0c} sind genau bestimmt.

Damit sich Neigungs- und Vergenzachse im optischen Zentrum der Kamera schneiden, ist insbesonders die richtige Einstellung des *optischen Freiheitsgrades* Zoom zu beachten. Eine Verstellung dieses Freiheitsgrades bringt eine Verschiebung des optischen Zentrums mit sich und verletzt die gemachten Annahmen.

Kapitel 3

Inverse Kinematik

Auf der Basis des beschriebenen Kameramodells wird das Sakkadenproblem zuerst mathematisch gelöst werden. Dies bietet die Grundlage für eine genauere Analyse des Problems und die Möglichkeit, konventionelle Sakkadensteuerung mittels *kalibrierter* Kamera und neuronale Ansätze zu vergleichen.
Für ein Off-Line Training der neuronalen Netze sind diese Berechnungen nicht von Bedeutung, da wir uns hier anderer, noch vorzustellender Ansätze bedienen werden.
Bei der Berechnung der inversen Kinematik werden der monokulare und der binokulare Fall unterschieden werden. Um eine eindeutige Lösbarkeit zu erhalten, sind in beiden Fällen Einschränkungen zu machen. Im monokularen Fall wird eine Sakkade durch eindeutige Korrektur von Neigungs- und Vergenzwinkel ausgeführt. Die Einschränkung besteht in einem konstanten Schwenkwinkel. Im binokularen Fall ist eine eindeutige Aufgabenstellung gegeben, wenn nach der Sakkade symmetrische Vergenz verlangt wird, d.h. die Vergenzwinkel gleich sind.
Die beiden Aufgabenstellungen werden beschrieben durch die vor der Sakkade gegebenen Kamerawinkel ω und Bildpunkte p, sowie die gesuchten Kamerawinkel ω^*, unter denen der interessante Szenepunkt im Zentrum des Kamerabildes liegt:

- Monokularer Fall:
  Gegeben: $(\omega, p) = (\chi, \phi, \theta_{l/r}, x_{i_{l/r}}, y_{i_{l/r}})$
  Gesucht: $\omega^* = (\chi^*, \phi^*, \theta^*_{l/r})$
  Bedingung: $\chi = \chi^*$ (Keine Bewegung des Schwenkwinkels)

- Binokularer Fall:
  Gegeben: $(\omega, p) = (\chi, \phi, \theta_l, \theta_r, x_{i_l}, y_{i_l}, x_{i_r}, y_{i_r})$
  Gesucht: $\omega^* = (\chi^*, \phi^*, \theta^*_l, \theta^*_r)$
  Bedingung: $\theta^*_l = \theta^*_r$ (Symmetrische Vergenz)

3.1 Monokularer Fall

Bei der Verwendung nur einer Kamera stellt sich das Problem wie folgt:
Sei ein Parametersatz

$$(\phi, \theta_{l/r}, x_{i_{l/r}}, y_{i_{l/r}})$$

gegeben. Gesucht ist ein neuer Parametersatz

$$(\phi^*, \theta^*_{l/r}) \ ,$$

so daß gilt:

$$\begin{pmatrix} 0 \\ 0 \\ \lambda \\ 1 \end{pmatrix} = T^{-1}_{\theta^*_{l/r}} T^{-1}_{\phi^*} \begin{pmatrix} x_w \\ y_w \\ 1 \\ 1 \end{pmatrix} \quad ; \quad \lambda \in \mathbb{R}$$

Unter den neuen Winkeln $(\phi^*, \theta^*_{l/r})$ wird der Sehstrahl $\vec{p}_w = (x_w, y_w, 1)$ dann in die Bildmitte abgebildet.
Dabei ist zu bemerken, daß bei der Verwendung nur einer Kamera die Betrachtung der Stereobasis b nicht benötigt wird, da der Schwenkwinkel nicht bewegt wird. In den Transformationsmatrizen T_{θ_l} und T_{θ_r} wird deshalb $b = 0$ gesetzt, und der Ursprung des Weltkoordinatensystems ist gleich dem Ursprung des Kamerakoordinatensystems.

Die Lösung dieser Aufgabe teilt sich in zwei Abschnitte, die Berechnung des Sehstrahles und die Berechnung der neuen Winkel:

3.1.1 Berechnung des Sehstrahls

Der Sehstrahl $\lambda(x_w, y_w, 1)$ berechnet sich mit $\lambda\vec{p}_c = P_c^{-1}\vec{p}_i$ aus:

$$\begin{pmatrix} x_w \\ y_w \\ 1 \\ 1 \end{pmatrix} = T_\phi T_{\theta_{l/r}} \begin{pmatrix} \lambda x_c \\ \lambda y_c \\ \lambda \\ 1 \end{pmatrix} \quad ; \quad \lambda \in \mathbb{R}$$

Es folgen die Lösungen getrennt für linke und rechte Kamera (entgegengesetzte Vorzeichen der Vergenzwinkel).

- Linke Kamera:

$$x_{w_l} := \frac{-x_c \cos\theta_l \; - \sin\theta_l}{x_c \cos\phi \; \sin\theta_l \; - y_c \sin\phi \; - \cos\phi \; \cos\theta_l}$$

$$y_{w_l} := \frac{\sin\phi \; \cos\theta_l \; - x_c \sin\phi \; \sin\theta_l \; - y_c \cos\phi}{x_c \cos\phi \; \sin\theta_l \; - y_c \sin\phi \; - \cos\phi \; \cos\theta_l}$$

- Rechte Kamera:

$$x_{w_r} := \frac{x_c \cos\theta_r \; - \sin\theta_r}{x_c \cos\phi \; \sin\theta_r \; + y_c \sin\phi \; + \cos\phi \; \cos\theta_r}$$

$$y_{w_r} := \frac{y_c \cos\phi \; - \sin\phi \; \cos\theta_r \; - x_c \sin\phi \; \sin\theta_r}{x_c \cos\phi \; \sin\theta_r \; + y_c \sin\phi \; + \cos\phi \; \cos\theta_r}$$

3.1.2 Berechnung der Winkel

Wird ein Punkt genau in die Bildmitte abgebildet, entspricht dies dem Vektor $\vec{p}_c = (0, 0, \lambda, 1)$ in Kamerakoordinaten, wobei λ beliebig ist. Es ist also folgendes Gleichungssystem zu lösen:

$$\begin{pmatrix} 0 \\ 0 \\ \lambda \\ 1 \end{pmatrix} = T_{\theta^*}^{-1} T_{\phi^*}^{-1} \begin{pmatrix} x_w \\ y_w \\ 1 \\ 1 \end{pmatrix} \quad ; \quad \lambda \in \mathbb{R}$$

Es ergeben sich die Lösungen:

- Linke Kamera:

$$\phi^* := \arctan(\, -y_w \,)$$

$$\theta_l^* := \arctan\left(\frac{x_{w_l}}{-y_{w_l} \sin\phi^* \; + \cos\phi^*} \right)$$

- Rechte Kamera:

$$\phi^* := \arctan(\, -y_w \,)$$

$$\theta_r^* := \arctan\left(\frac{-x_{w_r}}{-y_{w_r} \sin\phi^* \; + \cos\phi^*} \right)$$

Eine andere Darstellung des neuen Vergenzwinkels nach der Sakkade ist gegeben durch:

$$\theta^*_{l/r} := +/- \arctan\left(\frac{x_{w_{l/r}}}{\sqrt{1 + y^2_{w_{l/r}}}}\right)$$

Durch einfache Umformungen wird sie aus den obigen Gleichungen gewonnen. Es zeigt sich die Unabhängigkeit der Berechnung von z_w, die schon in Abschnitt 2.2 angesprochen wurde.

3.2 Binokularer Fall

Im binokularen Fall erhält man folgendes Problem: Sei ein Parametersatz

$$(\chi, \phi, \theta_l, \theta_r, x_{i_l}, y_{i_l}, x_{i_r}, y_{i_r})$$

gegeben. Gesucht ist ein neuer Parametersatz

$$(\chi^*, \phi^*, \theta_l^*, \theta_r^*) \ ,$$

so daß gilt:

1. $$\theta_l^* = \theta_r^*$$

2. $$\begin{pmatrix} 0 \\ 0 \\ \lambda_l' \\ 1 \end{pmatrix} = T_{\theta_l^*}^{-1} T_{\phi^*}^{-1} T_{\chi^*}^{-1} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \quad ; \quad \lambda_l' \in \mathbb{R}$$

3. $$\begin{pmatrix} 0 \\ 0 \\ \lambda_r' \\ 1 \end{pmatrix} = T_{\theta_r^*}^{-1} T_{\phi^*}^{-1} T_{\chi^*}^{-1} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \quad ; \quad \lambda_r' \in \mathbb{R}$$

Der Lösungsweg teilt sich wieder in zwei Abschnitte:

3.2.1 Berechnung des Weltpunktes

Der Weltpunkt ergibt sich aus dem Schnittpunkt, der für linke und rechte Kamera berechneten Sehstrahlen:

$$\vec{p}_{w_l} = \lambda_l T_\chi T_\phi T_{\theta_l} \vec{p}_{c_l} \quad ; \quad \lambda_l \in \mathbb{R}$$

$$\vec{p}_{w_r} = \lambda_r T_\chi T_\phi T_{\theta_r} \vec{p}_{c_r} \quad ; \quad \lambda_r \in \mathbb{R}$$

Durch Gleichsetzen folgt:

$$T_\chi T_\phi T_{\theta_r} \begin{pmatrix} \lambda_r x_{c_r} \\ \lambda_r y_{c_r} \\ \lambda_r \\ 1 \end{pmatrix} = T_\chi T_\phi T_{\theta_l} \begin{pmatrix} \lambda_l x_{c_l} \\ \lambda_l y_{c_l} \\ \lambda_l \\ 1 \end{pmatrix} \quad ; \quad \lambda_l, \lambda_r \in \mathbb{R}$$

Dieses Gleichungssystem ist eindeutig lösbar (siehe Anhang A). Wir erhalten den Weltpunkt $\vec{p}_w = (x_w, y_w, z_w, 1)$.

3.2.2 Berechnung der Winkel

Zu lösen sind folgende Gleichungssysteme:

$$\begin{pmatrix} 0 \\ 0 \\ \lambda_l' \\ 1 \end{pmatrix} = T_{\theta_l^*}^{-1} T_{\phi^*}^{-1} T_{\chi^*}^{-1} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \quad ; \quad \lambda_l' \in \mathbb{R}$$

$$\begin{pmatrix} 0 \\ 0 \\ \lambda_r' \\ 1 \end{pmatrix} = T_{\theta_r^*}^{-1} T_{\phi^*}^{-1} T_{\chi^*}^{-1} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \quad ; \quad \lambda_r' \in \mathbb{R}$$

Es folgen die eindeutigen Lösungen:

$$\chi^* := \arctan\left(-\frac{x_w}{z_w}\right)$$

$$\phi^* := \arctan\left(\frac{y_w}{x_w \sin\chi^* - z_w \cos\chi^*}\right)$$

$$\theta_l^* = \arctan\left(\frac{-\frac{1}{2}\,b - x_w \cos\chi^* - z_w \sin\chi^*}{x_w \cos\phi^* \sin\chi^* + y_w \sin\phi^* - z_w \cos\phi^* \cos\chi^*}\right)$$

$$\theta_r^* := \arctan\left(\frac{-\frac{1}{2}\,b + x_w \cos\chi^* + z_w \sin\chi^*}{x_w \cos\phi^* \sin\chi^* + y_w \sin\phi^* - z_w \cos\phi^* \cos\chi^*}\right)$$

Die beiden Formeln zur Berechnung von θ_l^* und θ_r^* ergeben das gleiche Ergebnis, d.h. *symmetrische Vergenz*, unter dem Schwenkwinkel χ^*. Es sei darauf hingewiesen, daß die Möglichkeit besteht, einen beliebigen Winkel χ^* vorzugeben. Die Ergebnisse ϕ^*, θ_l^* und θ_r^* führen auch dann dazu, daß der Szenepunkt im Zentrum beider Kameras liegt. Die Berechnungen lassen hier die Möglichkeit offen, vom momentanen Zustand über mehrere Schritte, die den Szenepunkt immer im Zentrum der Kameras halten, zur symmetrischen Vergenz zu gelangen.

3.3 Abhängigkeiten

Nachdem das Sakkadenproblem mittels inverser Kinematik anhand des beschriebene Kameramodells gelöst worden ist, soll darauf aufbauend die Sicht auf die Aufgabenstellung ein wenig verändert werden. Gegeben sind wiederum die Kamerawinkel ω und die Bildpunkte p, gesucht sind diesmal die Winkeldifferenzen $\Delta\omega = \omega^* - \omega$ von den momentanen Winkeln $\omega = (\chi, \phi, \theta_l, \theta_r)$ zu den Winkeln nach der Sakkade $\omega^* = (\chi^*, \phi^*, \theta_l^*, \theta_r^*)$.
Die im folgenden beschriebenen Ergebnisse werden bei der Entwicklung neuronaler Netze eine wichtige Rolle spielen.

3.3.1 Monokularer Fall

Betrachten wir die Winkeldifferenzen

$$\Delta\phi = \phi^* - \phi \quad und \quad \Delta\theta = \theta_{l/r}^* - \theta_{l/r} \ ,$$

so ergeben sich folgende Abhängigkeiten (Beweis siehe Anhang B):

- $\Delta\phi$ ist abhängig von $\theta_{l/r}$, sowie $x_{i_{l/r}}$ und $y_{i_{l/r}}$, aber unabhängig von ϕ.
- $\Delta\theta_{l/r}$ ist abhängig von $\Delta\phi$ und $\theta_{l/r}$, sowie $x_{i_{l/r}}$ und $y_{i_{l/r}}$, aber unabhängig von ϕ.

Zur Berechnung eines korrekten Winkeloffsets $\Delta\omega$ für eine Sakkade ist die Kenntnis des momentanen Neigungswinkels ϕ folglich nicht notwendig. Die Berechnung des korrekten Winkeloffsets der Sakkade kann deshalb durch eine Funktion $f()$ mit folgenden Parametern ausgedrückt werden:

$$\begin{pmatrix} \phi^* \\ \theta_{l/r}^* \end{pmatrix} = \begin{pmatrix} \phi \\ \theta_{l/r} \end{pmatrix} + f(\theta_{l/r}\, , \, x_{i_{l/r}}\, , \, y_{i_{l/r}})$$

3.3.2 Binokularer Fall

Für alle Winkeldifferenzen

$$\Delta\chi = \chi^* - \chi \quad , \quad \Delta\phi = \phi^* - \phi \quad , \quad \Delta\theta_l = \theta_l^* - \theta_l \quad und \quad \Delta\theta_r = \theta_r^* - \theta_r$$

ergibt sich die Unabhängigkeit von χ (Der Beweis hierzu folgt analog dem Beweis im monokularen Fall).

Wie schon im monokularen Fall ist ein Winkel unabhängig. Die Kenntnis des Schwenkwinkels χ ist nicht notwendig zur Berechnung des korrekten Winkeloffsets. Die Berechnung des Winkeloffsets der korrekten Sakkade ist deshalb darstellbar durch eine Funktion mit folgenden Parametern:

$$\begin{pmatrix} \chi^* \\ \phi^* \\ \theta_l^* \\ \theta_r^* \end{pmatrix} = \begin{pmatrix} \chi \\ \phi \\ \theta_l \\ \theta_r \end{pmatrix} + f(\phi\,,\, \theta_l\,,\, \theta_r\,,\, x_{i_l}\,,\, y_{i_l}\,,\, x_{i_r}\,,\, y_{i_r})$$

3.4 Kameratoleranzen

Jedes reale System besitzt *Toleranzen*, d.h. wir dürfen unsere Ergebnisse auch nur im Sinne der Toleranz bewerten. Für einen Kamerakopf sind die Toleranzen gegeben in der Genauigkeit, mit der sich die Freiheitsgrade ansteuern lassen. Der TRC Binocular Kamerakopf besitzt eine Genauigkeit von $\pm 0.0225°$ für Schwenk- und Neigungswinkel, sowie $\pm 0.006°$ für die Vergenzwinkel.
Um einen Eindruck von den Auswirkungen dieser Toleranzen zu bekommen, benutzen wir das beschriebene Kameramodell. Es werden verschiedene Weltpunkte betrachtet werden, und deren Projektion in die Bildebene. Der Toleranzfehler ergibt sich jeweils aus der Abweichung im Bild, die durch die Veränderung der Kamerawinkel im Toleranzbereich entsteht.
Dabei werden für die *intrinsischen Kameraparameter* Werte angenommen, die ungefähr denen der realen Kamera entsprechen.
Wiederum werden monokularer und binokularer Fall unterschieden.

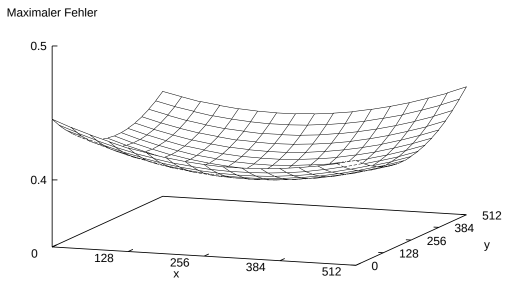


Abbildung 3.1: Toleranz bei Verwendung einer Kamera

3.4.1 Monokularer Fall

Im monokularen Fall brauchen nur die Toleranzen von Neigungs- und Vergenzwinkel betrachtet zu werden, da der Schwenkwinkel konstant bleibt. Für eine unterschiedlichen Stimulus in der Bildebene (x_i, y_i) und unterschiedliche Kamerawinkel (ϕ, θ) wird die Kameratoleranz verschieden große Abweichungen in Pixel erzeugen. Dabei interessiert nur die maximal auftretende Abweichung, die dadurch veranschaulicht wird, daß für einen festen Bildpunkt $(x_i, y_i) \in [0..512, 0..512]$ die maximale Abweichung unter der gegebenen Toleranz für Vergenzwinkel θ aus dem Intervall $[-40°..40°]$ bestimmt wird. Aufgrund der in Abschnitt 3.3 gewonnenen

Erkenntnisse ist der Neigungswinkel ϕ unabhängig, d.h für beliebige Neigungswinkel erhält man das gleiche Ergebnis.

Abbildung 3.1 zeigt die Projektion des maximalen Fehlers in die Bildebene. Zu erkennen ist der leicht höhere Fehler am Bildrand, insgesamt ist der maximale Fehler im Bereich von $[0.4, 0.5]$ Pixel aber fast konstant und vernachlässigbar.

3.4.2 Binokularer Fall

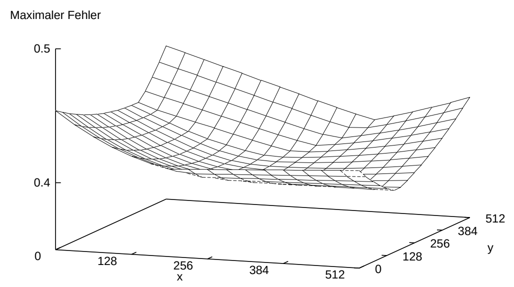


Abbildung 3.2: Toleranz bei Verwendung beider Kameras

Bei der Verwendung beider Kameras müssen die Toleranzen der Kamerawinkel $(\chi, \phi, \theta_l, \theta_r)$ beachtet werden. Es genügt, einen Vergenzwinkel θ_l oder θ_r zu betrachten, da aufgrund der symmetrischen Kamerageometrie das gleiche Ergebnis zu erwarten ist. Wir wählen hier exemplarisch θ_l aus.
Wieder ist die Abweichung in Abhängigkeit von der Lage des Szenepunktes im Kamerabild veranschaulicht (Abbildung 3.2). Dabei ist die maximale Abweichung dargestellt, die für verschiedene Kamerawinkel (ϕ, θ_l) auftritt. Als Winkelbereich ist für den Neigungswinkel ϕ das Intervall $[-30°..30°]$ und für den Vergenzwinkel θ_l das Intervall $[0°..10°]$ gewählt. Der Schwenkwinkel ist entsprechend dem Neigungswinkel im monokularen Fall unabhängig, und braucht nicht berücksichtigt zu werden (Abschnitt 3.3).
Auch bei der Verwendung beider Kameras liegt die durch Kameratoleranzen maximal auftretende Abweichung im *Subpixelbereich*, und ist damit vernachlässigbar klein.

3.5 Linearitäten

Dieser Abschnitt dient der weiteren Veranschaulichung der inversen Kinematik. Dabei sollen, wie im vorherigen Abschnitt, wieder die zur korrekten Sakkade zu berechnenden Winkeloffsets betrachtet werden. Diese hängen hauptsächlich von den Bildkoordinaten ab. Im folgenden wird näher untersucht, wie *linear* die Beziehung zwischen Bildkoordinaten und Winkeloffsets ist. Dabei wird insbesondere auf den Einfluß der momentanen Kamerawinkel auf diese Linearität eingegangen werden.

3.5.1 Monokularer Fall

Sei exemplarisch die linke Kamera ausgewählt. Im Abschnitt 3.3 wurde gezeigt, daß die Aufgabe, $\Delta\omega$ zu berechnen, unabhängig vom Neigungswinkel ϕ ist. Der Winkeloffset $\Delta\omega$ ist deshalb aus dem Vektor $(\theta_{l/r}, x_{i_{l/r}}, y_{i_{l/r}})$ zu berechnen.
Die Beziehung zwischen (ω, p) und $\Delta\omega$ ist in Abbildung 3.5.1 verdeutlicht, indem zuerst ein fest vorgegebener Vergenzwinkel θ_l ausgewählt wird. Die Stimuluspunkte, die den Reiz der Sakkade bestimmen, liegen auf einem äquidistanten Gitter über die ganze Bildebene verteilt. Jede Teilabbildung in 3.5.1 zeigt für einen festen Vergenzwinkel und jeden Stimuluspunkt aus dem Gitter den *Sakkadenoffset* $\Delta\omega$.

procedure
  Wähle Vergenzwinkel θ_l aus $\{0°, 20°, 40°, 60°\}$.
  for $x_{i_l} = 0$ **to** 512 **step** *step*
    for $y_{i_l} = 0$ **to** 512 **step** *step*
      Berechne Sakkadenoffset $(\Delta\phi, \Delta\theta_l)$ aus $(\theta_l, x_{i_l}, y_{i_l})$;
      plot $\Delta\phi, \Delta\theta_l$;
    end
  end
/* Die einzelnen Punkte werden entsprechend des äquidistanten Gitters in der Bildebene verbunden */

Man erkennt, daß mit größerem Vergenzwinkel θ_l die Beziehung zwischen Winkeloffset $\Delta\omega$ und Bildpunkten p immer nichtlinearer wird, während für kleine Vergenzwinkel θ_l diese näherungsweise linear aus den Bildkoordinaten (x_{i_l}, y_{i_l}) zu berechnen ist.

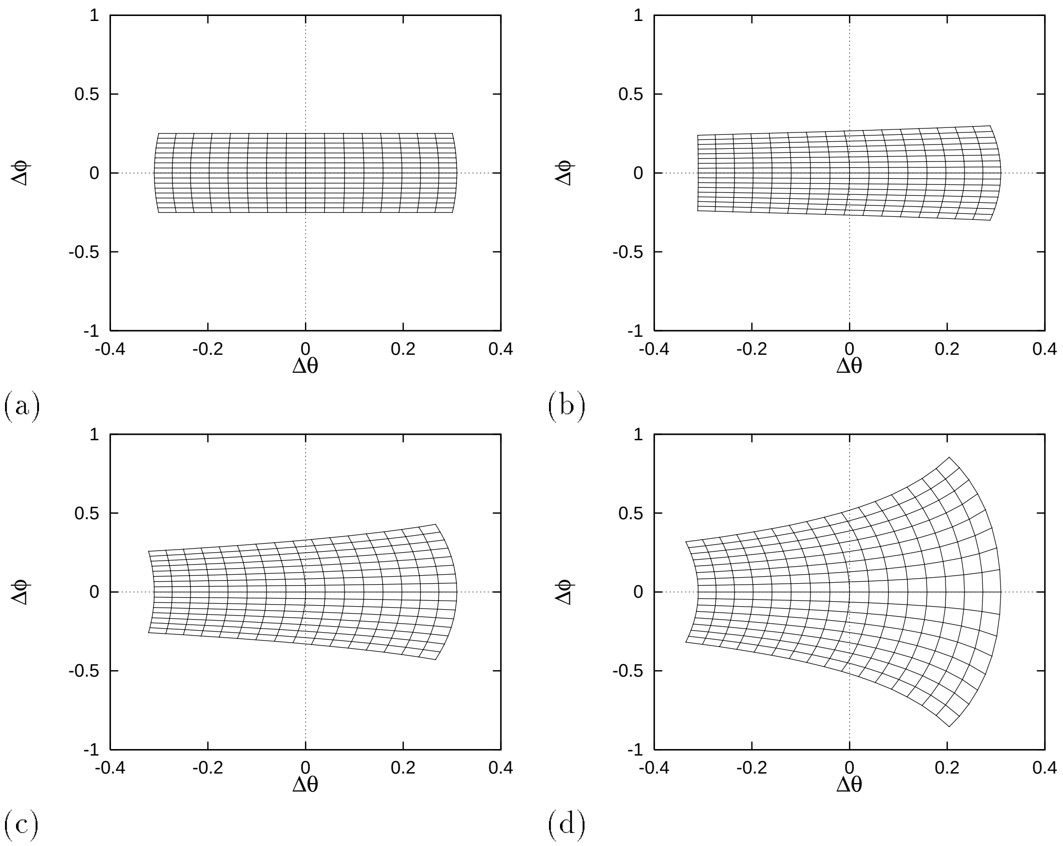


Abbildung 3.3: Abbildung von Bildkoordinaten $(x_i,\, y_i)$ auf Winkeloffset $(\Delta\phi,\, \Delta\theta)$ in Abhängigkeit vom momentanen Vergenzwinkel θ: $(x_i,\, y_i)|_\theta \rightarrow (\Delta\phi,\, \Delta\theta)$. Die Graphik zeigt die Abbildung des Bildkoordinatengitters $(x_i,\, y_i) \in [0, 512]^2$ auf Winkeloffsets für die Vergenzwinkel (a) 0°, (b) 20°, (c) 40°, (d) 60°.

3.5.2 Binokularer Fall

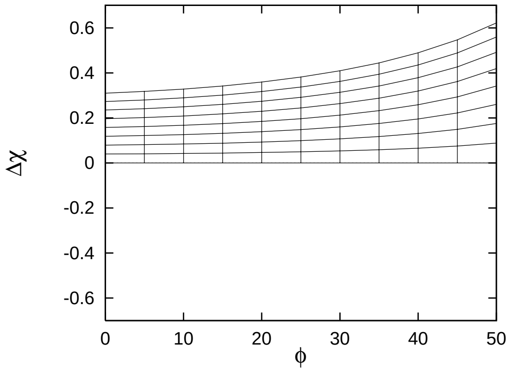
Unter Betrachtung beider Kameras ist die Aufgabe, Linearitäten zwischen Kamerakoordinaten p_i und Winkeloffsets $\Delta\omega$ in Abhängigkeit zu den momentanen Winkeln ω aufzuzeigen, etwas komplizierter. Die *hohe Dimensionalität* der Aufgabenstellung verlangt, *exemplarische Situationen* herauszugreifen und diese in Abhängigkeit der Winkel ω zu betrachten.
Im folgenden wollen wir uns darauf beschränken, exemplarische Situationen in Bezug zum Neigungswinkel ϕ aufzuzeigen.
Die Graphiken der Abbildung 3.4 sind durch folgenden Pseudoalgorithmus generiert worden:

procedure
  Setze $y_{i_l} = y_{i_r} = 0$.
  Setze $\theta_l = \theta_r = 1°$.
  Wähle Bildkoordinate x_{i_l} aus {0,128,256,384,512}.
  for $x_{i_r} = 0$ **to** 512 **step** *step*
    for $\phi = 0°$ **to** $50°$ **step** *step*
      Berechne Sakkadenoffset $(\Delta\chi, \Delta\phi, \Delta\theta)$
      aus $(\phi, \theta_l, \theta_r, x_{i_l}, y_{i_l}, x_{i_r})$;
      plot $\phi, \Delta\chi$;
    end
  end
/* Die einzelnen Punkte wurden entsprechend der Nachbarschaften im Eingaberaum verbunden */

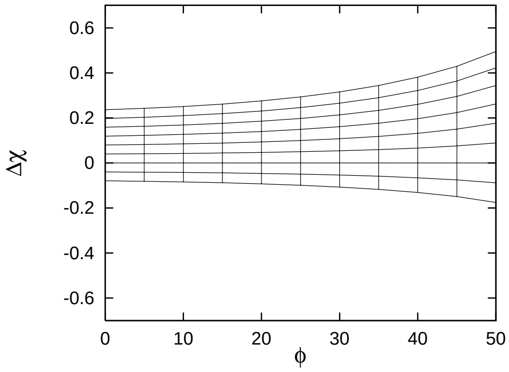
Die Graphiken der Abbildung 3.4 mögen auf den ersten Blick wenig aussagekräftig erscheinen, sie enthalten aber doch eine wesentliche Erkenntnis: Aufbauend auf einer Grundsituation aus $(\theta_l, \theta_r, x_{i_l}, y_{i_l}, x_{i_r})$ ist die Veränderung für den Offset des Schwenkwinkels $\Delta\chi$ unter Variation des Neigungswinkels *stärker als linear*. Zur Berechnung eines korrekten Winkeloffsets $\Delta\omega$ ist folglich insbesondere bei großen Neigungswinkeln die genaue Kenntnis des momentanen Neigungswinkels erforderlich. Selbst kleine Abweichungen können hier zu einem beachtlichen Fehler der Sakkade führen.

3.5.3 Folgerungen

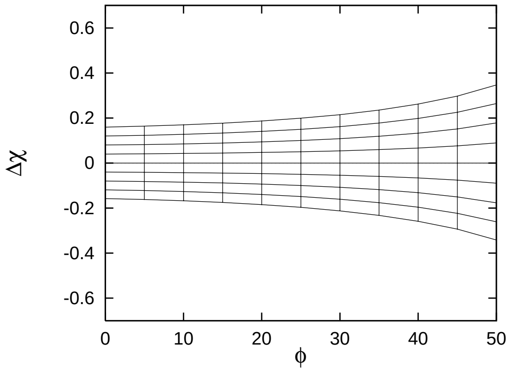
Besonders im Fall monokularer Sakkaden ist deutlich zu erkennen, daß für kleine Vergenzwinkel die Winkeloffsets fast linear aus den Bildkoordinaten zu berechnen sind. Im binokularen Fall wurde nur ein Teil der gesamten Aufgabe betrachtet, aber auch hier ist eine starke Linearität zwischen Bildkoordinaten und Winkeloffset zu erkennen.



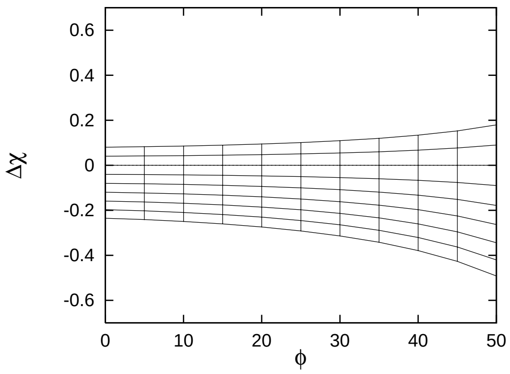
(a)



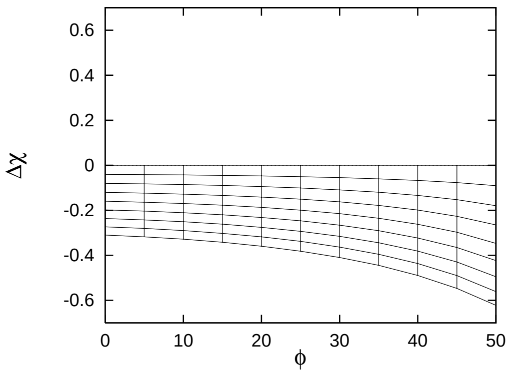
(b)



(c)



(d)



(e)

Abbildung 3.4: Die Einzelgraphiken zeigen den Schwenkwinkeloffset in Abhängigkeit vom momentanen Neigungswinkel ϕ und der Bildkoordinate x_{i_r}: Für jede Graphik gilt: $y_{i_l} = y_{i_r} = 0$ und $\theta_l = \theta_r = 1°$. In jeder Graphik wird die X-Koordinate der rechten Kamera x_{i_r} von 0 bis 512 variiert. Die einzelnen Graphiken unterscheiden sich in der X-Koordinate der linken Kamera x_{i_l}: (a) 0, (b) 128, (c) 256, (d) 384, (e) 512. Der Neigungswinkel ϕ variiert jeweils zwischen 0° und 50°. Die einzelnen Punkte $(\phi, \Delta\chi)$ sind entsprechend der Nachbarschaft im Eingaberaum verbunden.

Im Hinblick auf die Berechnung von Sakkaden mit neuronalen Netzen kann man den Schluß ziehen, daß es sinnvoll ist, einem Neuron eine Matrix zuzuordnen und damit durch Multiplikation mit dem Vektor der Bildkoordinaten einen Winkeloffset zu berechnen.
Unter dieser Annahme kann man folgern, daß die Inverse-Kinematik-Funktion in Regionen höherer Nichtlinearität schlechter zu approximieren ist, als in stark linearen Regionen. Im monokularen Fall finden sich Bereiche hoher Nichtlinearität bei großen Vergenzwinkeln entgegen starker Linearität nahe $0°$ Vergenz. Im binokularen Fall sind die Nichtlinearitäten bei hohen Neigungswinkeln, und die Funktion ist fast linear für kleine Neigungswinkel.

Kapitel 4

Konventionelle Kamerakalibrierung

Dieses Kapitel soll einen kurzen Einblick in die Aufgabe des *Kalibrierens* geben, d.h. die Bestimmung der *intrinsischen Kameraparameter* (Siehe Abschnitt 2.2). Das vorgestellte Kameramodell, initialisiert mit intrinsischen Kameraparametern, ist ein grobes mathematisches Modell der Kameraplattform TRC Bisight. Wie in Kapitel 3 gezeigt, werden auf diesem Modell durch die Lösung der inversen Kinematik Sakkaden berechnet.
Es wird beschrieben werden, welche Genauigkeit bei der Ausführung von Sakkaden unter Realweltbedingungen mit diesem Modell zu erreichen ist.
Der zur Kamerakalibrierung verwendete Ansatz folgt Arbeiten von O.Faugeras [Fau93].

4.1 Methoden

In der Literatur sind verschiedenste Verfahren der Kalibrierung diskutiert worden, es sei hier auf [Fau93], [Tsa86], [Tsa89], [Du93], [Li94], [Ste95] verwiesen.
Als konventionelle Kalibrierverfahren bezeichnet man diejenigen Verfahren, die auf der Auswertung der Aufnahme eines besonderen *Kalibrierkörpers* basieren. Aktive Kalibrierverfahren arbeiten mit der Veränderung der *Kamerafreiheitsgrade* durch Rotation oder Translation der Kamera, oder durch Veränderung des Zoom. Aktive Verfahren versuchen, auf einen besonderen Kalibrierkörper zu verzichten.
Im folgenden wird die Aufgabe der Kalibrierung beschrieben, wie sie sich nach der konventionellen Methode unter Verwendung eines Kalibrierkörpers stellt.
Das Bild des Kalibrierkörpers, mit der Kamera aufgenommen, wird mit Bildver-

arbeitungsverfahren nach Szenenmerkmalen, wie Kanten, Ecken, Bohrungen oder Ähnlichem untersucht. Man erhält aus dem Bild die *Szenenkoordinaten*:

$$S(n) = \{s_i = (x_{s_i}, y_{s_i})^T \mid i \in \mathbb{N}_{<n}\}.$$

Durch Verwendung eines bekannten Kalibrierkörpers können diesen Szenenkoordinaten ihre Weltkoordinaten zugeordnet werden:

$$K(n) = \{k_i = (x_{k_i}, y_{k_i}, z_{k_i})^T \mid i \in \mathbb{N}_{<n}\}.$$

In Kapitel 2 wurde ein mathematisches Kameramodell vorgestellt. Die Aufgabe der Kalibrierung ist es, die Matrix P_c:

$$P_c = \begin{pmatrix} \alpha_{xc} & 0 & x_{0c} \\ 0 & \alpha_{yc} & y_{0c} \\ 0 & 0 & 1 \end{pmatrix}$$

so zu bestimmen, daß sich die modellhafte Projektion der Szenenmerkmale in die Bildebene mit der realen möglichst gut decken. Seien $m_i = (x_{m_i}, y_{m_i})^T$ die Bildkoordinaten, auf die das Szenenmerkmal k_i im Modell projiziert wird. Dann ist P_c so zu bestimmen, daß die Summe

$$\sum_{i=1}^{n} \|s_i - m_i\|^2$$

minimiert wird.

Bei den gemachten Versuchen unter Realweltbedingungen wurde mit einer Kalibrierung gearbeitet, deren mittlerer Fehler zwischen Szenenkoordinaten s_i und projizierten Koordinaten m_i geringer als 1 Pixel ist. Dabei wurden jeweils mehr als 150 Szenenmerkmale berachtet.

4.2 Anwendung auf dem realen System

Zur Stimuluspräsentation wird ein Stäubli RX90 Roboter verwendet. Dieses Modell ist ein Knickarm-Roboter mit sechs Freiheitsgraden. An dem Greifer des Roboters ist eine Taschenlampe montiert. Die Stimuluspunkte liegen in Weltkoordinaten des Roboters in einem Quader der Größe $[200, 400, 350]$. Die Angaben für $[x, y, z]$ sind in Millimetern, die Größe des Quaders begrenzt sich durch die Beweglichkeit des Roboters.
Die Bilder des Kamerakopfes werden mit Hilfe eines Bildverarbeitungsrechners

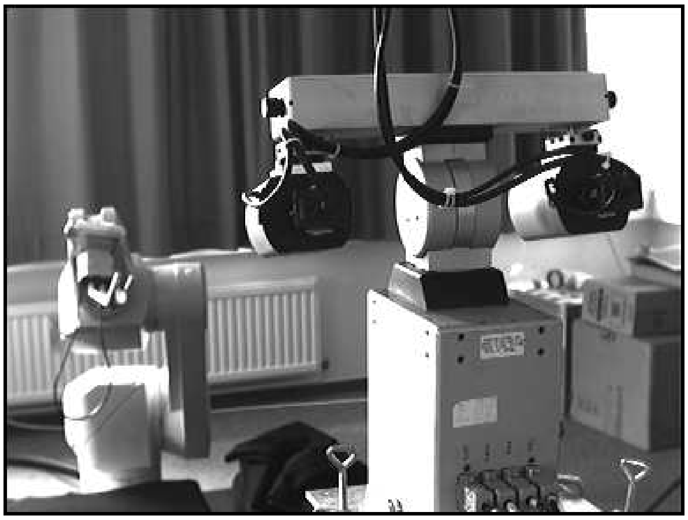


Abbildung 4.1: Versuchsaufbau mit Roboter und Kamerakopf.

der Firma Datacube analysiert. Die Berechnung der Sakkaden erfolgt auf einer SUN Workstation.

Der exakte Stimuluspunkt in der Bildebene ergibt sich als Schwerpunkt aller über einem Grenzwert liegenden Lichtpunkte. Ein weiterer Versuch gibt Aufschluß über die Stabilität der Lichtpunktbestimmung.

Genauigkeit der Lichtpunktbestimmung

Aufbauend auf den in Abschnitt 3.4 berechneten vernachlässigbaren Abweichungen durch die Kameratoleranz bestätigt ein weiterer Versuch die hohe Stellgenauigkeit des Kamerakopfes und die Stabilität der Lichtpunktdetektion. Ein fester Lichtpunkt im Raum wurde unter verschiedenen Kamerawinkeln betrachtet und jeweils der Schwerpunkt berechnet. Die Kamerawinkel wurden jeweils 100 Mal angefahren und die Mittelwerte der Schwerpunkte sowie die Standardabweichung berechnet. Die Standardabweichung liegt dabei mit 0.56 Pixel im Bereich der berechneten Kameratoleranzen (Abschnitt 3.4) und ist vernachlässigbar.

Ein Schritt in der Bestimmung des Fixationsfehlers besteht aus den Abschnitten:

- Anfahren der neuen Stimulusposition durch den Roboter
- Aufnehmen der Kamerabilder
- Lichtpunktbestimmung in den Bildern

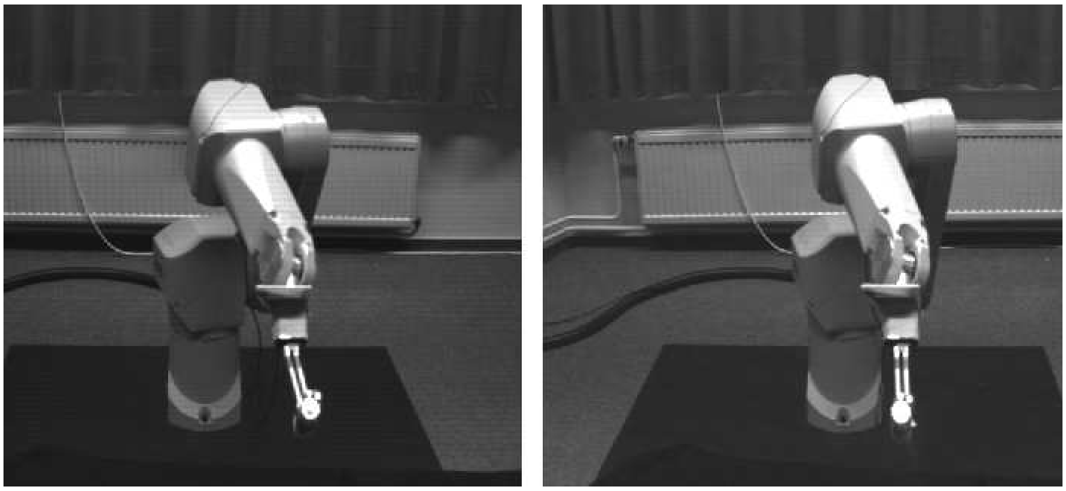
- Berechnen der Sakkade
- Ansteuern der Kamera auf die neuen Winkel
- Aufnehmen neuer Kamerabilder
- Lichtpunktbestimmung in den neuen Bildern zur Fehlerbestimmung

Der entscheidende Zeitschritt ist hier das Anfahren der Stimuluspunkte durch den Roboter, so daß ein Schritt zwischen zwei und drei Sekunden dauert.

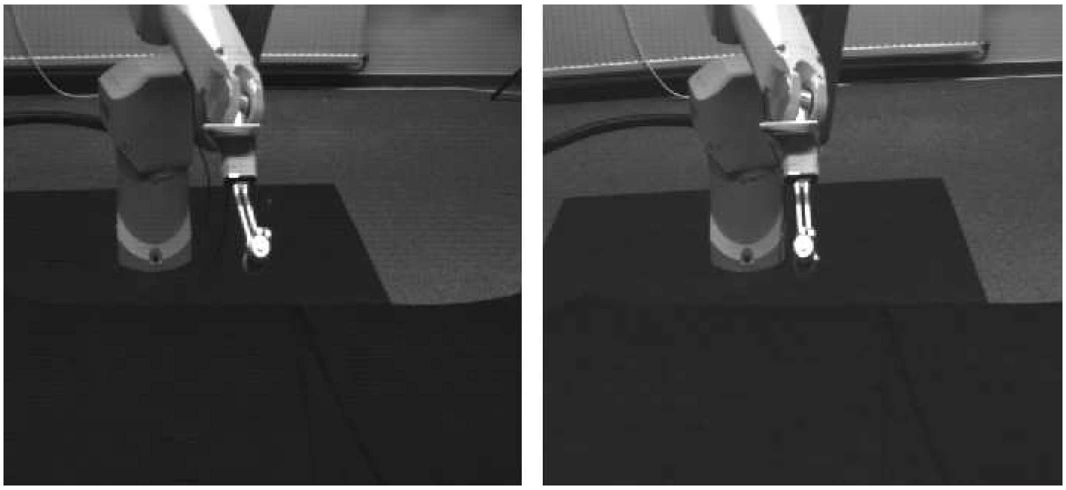
Im Versuchsaufbau ist der Kamerakopf zum Roboter so positioniert, daß der Abstand etwa 1.5 Meter beträgt (Abbildung 4.1). So genügt die Beweglichkeit des Roboters, um genügend große retinale Verschiebungen im Kamerabild zu erzeugen.
Ein Beispiel der Kamerabilder vor und nach der Sakkade ist in Abbildung 4.3 dargestellt.
Abbildung 4.2 zeigt den gemittelten Fehler über 1000 Sakkaden und die Winkelbereiche, in denen sich das Kamerasystem bewegte. Die Fehler der Sakkaden sind auf mehrere Gründe zurückzuführen. Das verwendete mathematische Modell ist nur eine unzureichende Modellierung der realen Kamera und die Annahmen sind nicht exakt erfüllt. Außerdem sind die intrinsischen Kameraparameter nicht perfekt zu bestimmen, so daß durch diese Differenzen zwischen Modell und realer Kamera die auf dem Modell perfekte Berechnung von Sakkaden zu Abweichungen unter Realweltbedingungen führt.

Sakkaden	Monokular	Binokular
Fehler in Pixel	2.39	3.20
Standardabweichung	1.44	1.86
Winkelbereiche in Grad:		
Schwenkwinkel	-/-	-7/7
Neigungswinkel	-16/-2	-15/-2
Vergenzwinkel	-3/10	2/4

Abbildung 4.2: Auf dem realen System werden Sakkaden ausgeführt, die anhand des beschriebenen Modells berechnet wurden. Die Kameras wurden sorgfältig kalibriert. Die Tabelle zeigt den gemittelten Fehler in Pixel und die getesteten Winkelbereiche.



(a)



(b)

Abbildung 4.3: Blick durch die linke und rechte Kamera vor und nach der Ausführung der Sakkade. Man erkennt die Taschenlampe im Robotergreifer.

Kapitel 5

Neuronale Architekturen

Dieses Kapitel gibt eine Einführung in neuronale Architekturen mit Vergleichen zwischen Kohonenkarten und Dynamischen Zell-Strukturen. Die beiden Aufgabenstellungen monokularer und binokularer Fall werden im Hinblick auf neuronale Netze näher besprochen. Die Frage nach der Organisation des überwachten Lernens (Supervised Learnig) führt zum *Error-Feedback-Lernen*. Dieses wird näher vorgestellt und an die Aufgabenstellung angepasst.

5.1 Einführung

Neuronale Architekturen gehören in die Klasse der *adaptiven Systeme*. Sie bieten die Möglichkeit, im Laufe einer Trainingsphase gewünschtes Verhalten zu erlernen bzw. sich ändernden Arbeitsbedingungen anzupassen. Trainingsphase und Arbeitsphase sind nicht notwendigerweise entkoppelt, neuronale Netze können selbstständig auf Veränderungen reagieren, explizite *Rekalibrierung* ist nicht notwendig.

Kohonens selbstorganisierende Karten (Self Organising Map: SOM), sind neben Backpropagation heutzutage die meist gebräuchlichen und erfolgreichsten künstlichen neuronalen Netze (Artificial Neural Nets: ANN). Sie werden verwendet zur adaptiven Vektor-Quantisierung, zum Clustern, zur Dimensions-Reduktion und durch Hinzufügen einer Ausgabe zu jedem Neuron als *sensomotorische Karten* [Rit91].

Ihre Haupteigenschaften sind:

- Der Aufbau von *topologieerhaltenden Karten*, d.h. ähnliche Reizimpulse werden auf benachbarte Neurone abgebildet.

- Die Karten spiegeln die *Wahrscheinlichkeitsdichte* der Eingabesignale wieder. Je größer die Wahrscheinlichkeitsdichte, desto größer ist auch die Neuronendichte.

Bekannte Nachteile der Kohonenkarten sind:

- Durch die feste Anzahl von Neuronen treten Probleme auf mit Anwendungen, bei denen die optimale Anzahl von Neuronen nicht vorher bekannt ist.
- Die vorher festzulegende Dimensionalität der Topologie führt zu Problemen, wenn sie nicht mit der Topologie der Anwendung übereinstimmt.
- Nicht immer ist es vorteilhaft, die Wahrscheinlichkeitsverteilung des Eingaberaumes zu modellieren. Ist das Ziel die Optimierung der Ausgabe, so können Regionen geringer Eingabedichte ebenso bedeutend, oder bedeutender, sein wie Regionen hoher Eingabedichte.

Diese Nachteile werden versucht, mit einem modifizierten Ansatz, im folgenden als DCS (Dynamic Cell Structures) bezeichnet, auszugleichen:

- Ein lokales Fehlermaß wird genutzt, um das Einfügen neuer Neuronen zu steuern.
- Eine einfache Regel führt zu einer Adaption der Topologie, basierend auf den Eingabereizen.
- Auf *Radialen Basis-Funktionen* (RBF) basierte Berechnung der Ausgabe.

Dieser Algorithmus basiert auf den Arbeiten von [Bru95] [Fri93a] [Fri93b] [Mar93].

5.2 Gemeinsamkeiten der Aufgabenstellung

Kohonenkarten und DCS besitzen Gemeinsamkeiten in der Architektur. Beide bestehen aus Ein- und Ausgabeschicht. In der Eingabeschicht ist jedem Neuron ein Vektor w in der Dimension der Eingabe zugeordnet, während jedem Neuron in der Ausgabeschicht eine Matrix A zugeordnet ist.
Wie in Abschnitt 3.5 gezeigt ist, hängt die zur Sakkade notwendige Winkelkorrektur in der Hauptsache von den Bildkoordinaten ab. Wir berechnen die Ausgabe eines Neurons deshalb als Matrixmultiplikation von A mit den Bildkoordinaten

des zu zentrierenden Szenenpunktes.
Weitere Gemeinsamkeiten in der Aufgabenstellung ergeben sich in den Ein- und Ausgabevektoren der neuronalen Netze, sowie natürlich in der Messung des Fixationsfehlers der Sakkaden.

Ein- und Ausgabevektoren

Ohne näher auf Kohonenkarten oder DCS eingehen zu müssen, können die Ein- und Ausgabevektoren angegeben werden. Die Eingabe besteht aus Kamerawinkeln und Bildkoordinaten. Die Winkel stammen aus einem vorher festzulegenden Intervall $I \subset [-\pi,\ \pi]$, die Bildkoordinaten aus dem Intervall $[0,\ 512]$. Um alle Eingabeparameter gleich zu gewichten, werden sie auf das Intervall $[-1,\ 1]$ *normalisiert.*
In Bezug zu Abschnitt 3.3 wird die Unabhängigkeit des Neigungswinkels im monokularen und die des Schwenkwinkels im binokularen Fall ausgenutzt, und als Ausgabe der neuronalen Netze wird ein Winkeloffset zu den momentanen Winkeln berechnet.

Für monokularen und binokularen Fall folgt damit:

- Monokularer Fall:
  Die Eingabe entsteht aus dem Vergenzwinkel $\theta_{l/r}$ und den Bildkoordinaten $x_{i_{l/r}}$, $y_{i_{l/r}}$ durch normalisieren:
  $$\left(\frac{\theta_{l/r}}{\theta_g}\ ,\ \frac{x_{i_{l/r}} - 256}{512}\ ,\ \frac{y_{i_{l/r}} - 256}{512} \right)$$
  Der Vergenzwinkel ist auf das Intervall $[-\theta_g,\ \theta_g]$ beschränkt.
  Die Ausgabe ist der Winkeloffset $\omega' = (\phi',\ \theta'_{l/r})$. Der Kamerakopf wird auf die neuen Winkel $(\phi + \phi',\ \theta_{l/r} + \theta'_{l/r})$ eingestellt.
  Als Variante dieses Ansatzes kann für beide Kameras unabhängig die Berechnung ausgeführt werden. Da es aufgrund der Kamerageometrie nicht notwendigerweise möglich ist, den Lichtpunkt entsprechend der Y-Achse in beiden Kameras zu zentrieren, sind auch unterschiedliche Neigungswinkel bei der unabhängigen Berechnung zu erwarten. Hier muß eine Kamera die Dominanz übernehmen.
- Binokularer Fall:
  Die vier Bildkoordinaten einer Lichtquelle im Raum sind nicht unabhängig. Sind beispielsweise beide X-Koordinaten und eine Y-Koordinate gegeben, so ist auch die zweite Y-Koordinate bestimmt. Berechnent man für beide Kameras aus den Bildkoordinaten jeweils den Sehstrahl, so müssen sich beide Sehstrahlen in einem Weltpunkt schneiden, dies kann aber nur der Fall sein, wenn beide Y-Koordinaten voneinander abhängen.
  Die Eingabe besteht deshalb nur aus drei Bildkoordinaten, sowie

Neigungs- und Vergenzwinkeln.
Die eindeutige Lösbarkeit der Aufgabe erfordert symmetrische Vergenz nach der Sakkade. Wird der Kamerakopf nur anhand von Sakkaden gesteuert, kann deshalb angenommen werden, daß auch vor der Sakkade symmetrische Vergenz vorliegt ($\theta_l = \theta_r$). Im folgenden wird diese Einschränkung angenommen, in der Eingabe des neuronalen Netzes steht nur ein (der) Vergenzwinkel $\theta = \theta_l = \theta_r$. Aus dieser Einschränkung kann $\theta > 0$ abgeleitet werden.
Es folgt die normalisierte Eingabe:

$$\left(\frac{\phi}{\phi_g}\, , \; \frac{\theta - 0.5\,\theta_g}{0.5\,\theta_g}\, , \; \frac{x_{i_l} - 256}{512}\, , \; \frac{y_{i_l} - 256}{512}\, , \; \frac{x_{i_r} - 256}{512} \right)$$

der Neigungswinkel ϕ stammt aus dem Intervall $[-\phi_g,\ \phi_g]$, der Vergenzwinkel θ aus dem Intervall $[0,\ \theta_g]$.
Die Ausgabe ist der Winkeloffset $\omega' = (\chi',\ \phi',\ \theta')$, aus dem die neuen Kamerawinkel folgen: $(\chi + \chi',\ \phi + \phi',\ \theta_l + \theta',\ \theta_r + \theta')$. Es gilt wieder symmetrische Vergenz.

Fixationsfehler der Sakkaden

Der Fixationsfehler der Sakkaden ergibt sich aus der mittleren Abweichung von der Bildmitte. Im monokularen Fall ist dies

$$F = \frac{|x_{i_{l/r}} - 256| + |y_{i_{l/r}} - 256|}{2}.$$

Im binokularen Fall ist zu bedenken, daß die beiden Y-Koordinaten der Projektion eines Szenenpunktes in die Kamerabilder nicht unabhängig sind. Auf einem realen System muß es deshalb nicht unbedingt möglich sein, einen Szenenpunkt entsprechend beider Y-Koordinaten zu zentrieren. Dem wird Rechnung getragen, indem in die Berechnung des Fixationsfehlers nur die Y-Koordinate einbezogen wird, bezüglich der auch zentriert wird:

$$F = \frac{|x_{i_l} - 256| + |y_{i_l} - 256| + |x_{i_r} - 256|}{3}.$$

Das Training der neuronalen Netze basiert auf überwachtem Lernen. Ein Eingabestimulus wird präsentiert, die Ausgabe berechnet und die gewünschte Ausgabe gelernt. Das Problem des Lernvorgangs auf einem realen System ist es, die gewünschte Ausgabe zu bestimmen. Von Kawato wurde die Methode des Error-Feedback-Lernens vorgeschlagen [Kaw90a]. Der Fehler der Sakkade wird dabei zur Generierung einer verbesserten Ausgabe genutzt. Diese verbesserte Ausgabe wird gelernt und so die Genauigkeit der Sakkaden erhöht.
Beim Error-Feedback-Lernen ist eine genaue Kenntnis der zu approximierenden

Funktion nicht notwendig, es eignet sich deshalb hervorragend für eine Anwendung unter Realweltbedingungen.
Als Error-Feedback-Regler verwenden wir einen einfachen Proportional-Regler, der schon von Mayhew erfolgreich zum Lernen von Sakkaden eingesetzt wurde [May92].

5.3 P-Regler

Aus den nach der Sakkade erhaltenen Bildkoordinaten können durch eine lineare Operation verbesserte Kamerawinkel erhalten werden. Dies ist von entscheidender Bedeutung für das Lernen der neuronalen Netze. So ergibt sich die Möglichkeit, nach jeder Sakkade eine gezielte Verbesserung in die richtige Richtung vorzunehmen.
Wir berechnen mit dem P-Regler aus den neuen Bildkoordinaten des Szenepunktes einen verbesserten Winkeloffset ω_P. Dieser kann sowohl zum Lernen des neuronalen Netzes verwendet werden als auch zu einer Korrektursakkade mit der Winkelstellung $\omega + \omega_P$.
Ein Eingabepaar $u' = (\omega', p')$ für den P-Regler besteht aus der Ausgabe des neuronalen Netzes ω' und den neuen nach der Sakkade abgelesenen Bildpunkten des Stimuluspunktes p'.

- Monokularer Fall:
  Sei ein Eingabepaar $u' = ((\phi, \theta_{l/r}), (x_{i_{l/r}}, y_{i_{l/r}}))$ gegeben, dann berechnen wir den verbesserten Winkeloffset ω_P wie folgt:

$$\begin{pmatrix} \phi_P \\ \theta_P \end{pmatrix} = \begin{pmatrix} \phi + k_1\,(y_{i_{l/r}} - 256) \\ \theta + k_2\,(x_{i_{l/r}} - 256) \end{pmatrix}$$

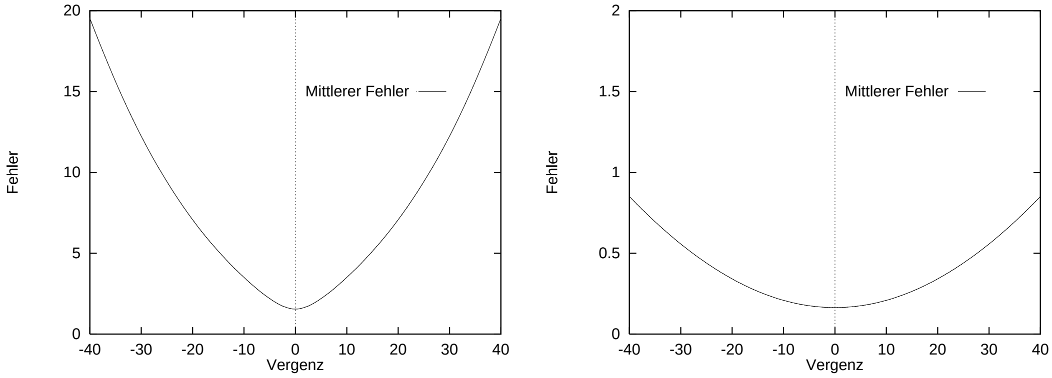
- Binokularer Fall:
  Sei ein Eingabepaar $\omega' = ((\chi, \phi, \theta), (x_{i_l}, y_{i_l}, x_{i_r}, y_{i_r}))$ gegeben, dann berechnen wir den verbesserten Winkeloffset ω_P wie folgt:

$$\begin{pmatrix} \chi_P \\ \phi_P \\ \theta_P \end{pmatrix} = \begin{pmatrix} \chi + l_1\,(x_{i_r} + x_{i_l} - 512) \\ \phi + l_2\,(y_{i_l} - 256) \\ \theta + l_3\,(x_{i_r} - x_{i_l}) \end{pmatrix}$$

Die Parameter $k_1, k_2, l_1, l_2, l_3 \in \mathbb{R}$ wurden anhand des Kameramodelles ermittelt.

Im monokularen Fall ist diese Idee anschaulich offensichtlich. Eine vertikale Abweichung kann durch Korrektur des Neigungswinkels, und eine horizontale durch Korrektur des Vergenzwinkels ausgeglichen werden. Auch die lineare Korrektur

im binokularen Fall ist so leicht einzusehen. Die Grundlage für diese Möglichkeit liegt in einer *starken linearen Komponente* der inversen Kinematik, ohne die eine so einfache Lösung nicht möglich wäre.
Um dies weiter zu veranschaulichen, bedienen wir uns wieder des Kameramodells, welches mit dem realen System entsprechenden intrinsischen Kameraparametern initialisiert wird.
Es wird gezeigt werden, daß der P-Regler eine ausgezeichnete Methode ist, um verbesserte Kamerawinkel zu erhalten. Schon nach kurzer Trainingszeit, wenn der Fehler der neuronalen Netze auf unter 10 Pixel abgesunken ist, erhält man durch eine Korrektursakkade Werte, die fast perfekt sind und den Sakkadenfehler nach der P-Korrektur auf weniger als 1 Pixel verringern.



(a) (b)

Abbildung 5.1: (a) Fehler nach Korrektur mit dem P-Regler unter Verwendung einer Kamera. Der Fehler ist aufgetragen über den Vergenzwinkel und gemittelt über die Bildkoordinaten.
(b) Wie (a), es sind aber nur Situationen betrachtet, bei denen der Stimuluspunkt in einem 20×20 Teilbild um die Bildmitte herum liegt.

- Monokularer Fall:
  Aus den Ergebnissen in Abschnitt 3.3 kann man schließen, daß auch hier der Absolutwert des Neigungswinkels nicht von Bedeutung ist, da die Berechnung vom Neigungswinkel unabhängig ist.
  Abbildung 5.1(a) stellt den Fehler nach der P-Korrektur für unterschiedliche Vergenzwinkel dar, gemittelt über die möglichen Bildkoordinaten vor der Korrektur ($(x_i, y_i) \in [0..512, 0..512]$).
  Der Vergenzbereich, in dem der P-Regler die besten Ergebnisse aufweist, ist abhängig von der Wahl der k_1, k_2. Dieser Bereich ist hier durch die Wahl $k_1 = -0.001$ und $k_2 = \pm 0.0012$ um $0°$ herum gewählt (Die Winkelangaben ω sind in Radian).

  Abbildung 5.1(b) liegt die gleiche Berechnung zugrunde wie 5.1(a), mit dem einzigen Unterschied, daß nicht mehr über die ganze Bildebene gemittelt

wurde, sondern nur über das 20×20 Teilbild um die Bildmitte herum.
Eine Netzkonvergenz auf einen Fehler von weniger als 10 Pixel wird wegen der starken linearen Komponente der Inversen Kinematik schnell erreicht sein, der Fehler nach der Korrektur der Sakkade durch den P-Regler beträgt dann durchschnittlich weniger als 1 Pixel. Dies spricht für die hervorragende Eignung dieses Ansatzes, um eine schnelle Konvergenz der neuronalen Netze zu erreichen.

- Binokularer Fall:
  Unter Verwendung beider Kameras ist die Verbesserung des P-Reglers abhängig von den Kamerawinkeln $(\chi, \phi, \theta_l, \theta_r)$ und den Bildkoordinaten $(x_{i_l}, y_{i_l}, x_{i_r}, y_{i_r})$. Die Idee des P-Reglers, so wir wir ihn vorgestellt haben, funktioniert nur unter symmetrischer Vergenz $(\theta_l = \theta_r)$). Diese ist nach der Ausführung der Sakkade immer gegeben, da nur die Forderung nach symmetrischer Vergenz eine eindeutige Lösung des Sakkadenproblems ergibt (Kapitel 3).
  Der P-Regler verwendet nur eine Y-Koordinate y_{i_l}. Auf einem realen Kamerakopf können Ungenauigkeiten in der Geometrie dazu führen, daß es nicht möglich, ist den Szenepunkt bezüglich beider Y-Koordinaten zu zentriern. Der vorgestellte P-Regler zentriert den Szenepunkt in der linken Kamera, indem y_{i_l} zur Berechnug der Winkelkorrektur verwendet wird.

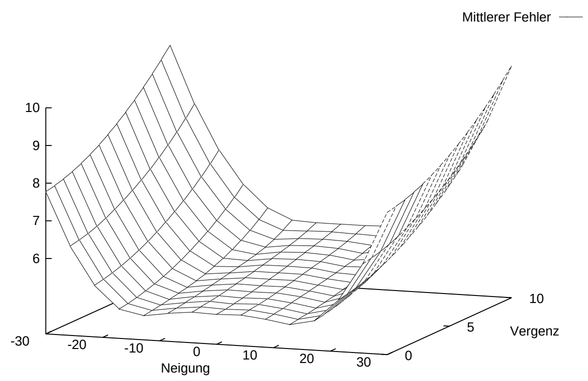


Abbildung 5.2: Fehler nach Korrektur mit dem P-Regler unter Verwendung beider Kameras. Der Fehler ist aufgetragen über den Neigungs- und Vergenzwinkel und gemittelt über die Bildkoordinaten.

Die Konstanten des P-Reglers sind $k_1 = 0.0007$, $k_2 = 0.001$ und $k_3 = 0.0006$. Abbildung 5.2 zeigt den Fehler nach der P-Korrektur aufgetragen über der Neigungswinkel ϕ und den Vergenzwinkel $\theta_l = \theta_r$. Der Schwenkwinkel ist

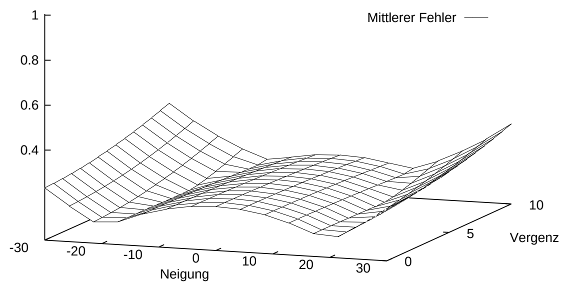


Abbildung 5.3: Fehler nach Korrektur mit dem P-Regler unter Verwendung beider Kameras. Der Fehler ist aufgetragen über den Neigungs- und Vergenzwinkel und gemittelt über die Bildkoordinaten. Dabei sind nur Situationen betrachtet, bei denen der Stimuluspunkt in beiden Kameras in einem 20×20 Teilbild um die Bildmitte herum liegt.

unabhängig und kann beliebig gewählt werden (Abschnitt 3.3). Der Fehler wurde dabei gemittelt über die möglichen Bildkoordinaten.

Der mittlere Fehler von bis zu 10 Pixel nach der P-Korrektur ist noch recht hoch, betrachtet man allerdings nur Bildkoordinaten, die für jede Kamera im 20×20 Teilbild um die Bildmitte herum liegen, so kann man erkennen, daß der mittlere Fehler weniger als 1 Pixel beträgt (Abbildung 5.3). Eine Konvergenz des Lernalgorithmus auf einen Fehler in der Größenordung von 10 Pixel wird schon mit wenigen Neuronen erreicht sein. Danach liefert der P-Regler fast den perfekten Wert, und die schnelle Konvergenz des Lernverfahrens ist gesichert.

5.4 Ablaufplan

Das Training der neuronalen Netze gestaltet sich ähnlich der in Kapitel 4 vorgestellten Sakkadensteuerung. Mit dem Bildverarbeitungssystem Datacube wird der Lichtpunkt im Kamerabild bestimmt. Das neuronale Netz berechnet den Winkeloffset der Sakkade auf einer SUN Workstation. Nachdem die neuen Winkel des Kamerakopfes eingestellt sind, wird im Kamerabild ein zweites Mal der Lichtpunkt bestimmt. Der P-Regler generiert aus dem Fixationsfehler einen verbesserten Winkeloffset, der gelernt wird. Dieser kann auch zu einer Korrektursakkade genutzt werden.

Im Schema sieht der Ablauf wie folgt aus:

- Ablesen von Kamerawinkeln ω und der Bildkoordinaten p des Lichtpunktes
- Berechnung der Ausgabe ω' des neuronalen Netzes
- Einstellen der neuen Kamerawinkel $\omega + \omega'$ und Ablesen der veränderten Bildkoordinaten p' des Lichtpunktes
- Berechnen der verbesserten Ausgabe ω_P aus den neuen Bildkoordinaten p' mit dem P-Regler; evtl. Korrektursakkade $\omega + \omega_P$
- Adaption des neuronalen Netzes mit ω_P

Dieses Schema wiederholt sich für jeden Trainingsschritt. Es ist in Abbildung 5.4 zusätzlich visuell veranschaulicht.

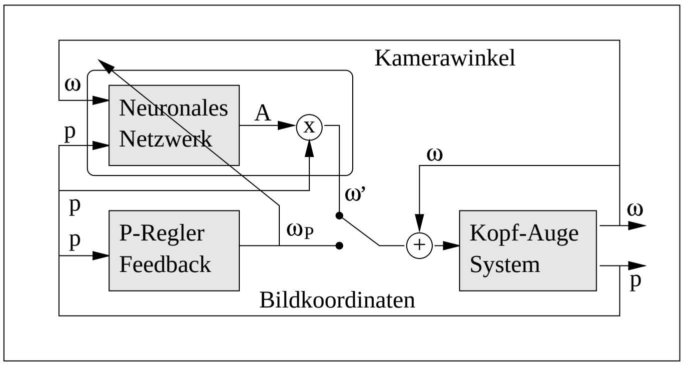


Abbildung 5.4: Die Abbildung zeigt den Datenfluß von Kamerawinkeln und Bildpunkten beim neuronalen Lernen zwischen den Komponenten Neuronales Netzwerk, P-Regler und dem Kopf-Auge System.

Kapitel 6

Kohonenkarten

Kohonenkarten stellen einen der beiden neuronalen Ansätze dar, mit denen in dieser Arbeit ein *iteratives Lernen* von Sakkaden realisiert ist. Der in diesem Kapitel vorgestellt Algorithmus basiert auf Arbeiten von T. Kohonen [Koh90] [Koh95].
Neben der Beschreibung des Lernalgorithmus wird insbesondere auf die bei Kohonenkarten festgelegte *Topologie* eingegangen. Welche Dimension das verwendete Neuronengitter im monokularen bzw. binokularen Fall besitzt, wird hier näher erläutert werden.

6.1 Einleitung

Kohonenkarten bestehen aus einer Schicht von Knoten (Neuronen), die durch eine *festgesetzte Topologie* miteinander verbunden sind. Der Vergleich mit *Neuronen* und *Synapsen* liegt hier nahe. Im erweiterten Modell von Ritter [Rit91] ist dieser *Eingabeschicht* eine *Ausgabeschicht* überlagert, die entsprechend der Eingangsreize und der damit verbundenen Aktivierungen der Neuronen eine Ausgabe berechnet.
Mathematisch beschrieben wird eine Funktion

$$f : \mathbb{R}^N \longrightarrow \mathbb{R}^M$$

approximiert.

6.2 Notationen

Um den verwendeten Algorithmus zu beschreiben, führen wir folgende Notationen ein:

- Sei $\Omega \subset \mathbb{R}^K$; $K \in \mathbb{N}$ der Eingaberaum der Kamerawinkel.
- Sei $\Omega' \subset \mathbb{R}^M$; $M \in \mathbb{N}$ der Ausgaberaum der Kamerawinkeloffsets.
- Sei $P \subset \mathbb{R}^L$; $L \in \mathbb{N}$ der Raum der Bildkoordinaten.
- Ein Eingabestimulus sei definiert als $u = (\omega, p)$ mit $\omega \in \Omega$; $p \in P$.
- Sei $W = 1, 2, .., I$ die Menge der Knoten.
- Seien $w_r \in \mathbb{R}^N$; $r \in W$; $N = K + L$ die Vektoren der Eingabeschicht.
- Seien $A_r \in \mathbb{R}^{L \times M}$; $r \in W$ die Matrizen der Ausgabeschicht.

6.3 Algorithmus

Der Algorithmus zur Erstellung einer Kohonenkarte teilt sich in mehrere Abschnitte, die sich während der Trainingsphase ständig wiederholen. Die Trainingslänge ist abhängig von der Größe des Netzes, d.h der Anzahl an Neuronen. Je mehr Neuronen das Netz enthält, desto mehr Lernschritte sind notwendig, um das Netz optimal zu trainieren. Von der Trainingslänge hängen die *Lernparameter* ab, sie werden in einer Lernperiode immer kleiner und führen damit zu einem 'Einfrieren' der Netzes. Hier sind mehrere Versuche notwendig, um das richtige Verhältnis von Trainingslänge und Lernparametern zu finden.
Die Topologie des Netzes hängt von der Aufgabenstellung ab. Ungeachtet der Dimensionalität der Eingabedaten, ist zu entscheiden, welche *Dimensionalität* der Raum hat, den diese Daten repräsentieren.
Wird beispielsweise ein Raum durch zwei statische Kameras beobachtet, so erhält man für jeden Szenenpunkt einen vierdimensionalen Vektor, nämlich X- und Y-Koordinaten des Punktes in der linken, bzw. rechten Kamera. Dieser vierdimensionalen Eingabevektor repräsentiert aber nur einen dreidimensionalen Raum.

Die Vektoren w_r der Eingabeschicht und die Matrizen A_r ; $r \in W$ werden mit zufälligen Werten initialisiert. Vorkenntnisse über den Wertebereich der Eingaben und über zu erwartende Matrizen können eingebracht werden, um eine Verkürzung der Lerndauer zu erzielen, sie sind aber nicht notwendig.
Da die Eingaben sich aus Winkeln und Bildpunkten zusammensetzen und unterschiedliche Wertebereiche besitzen, werden sie entsprechend ihrer Grenzwerte auf das Intervall $[-1, 1]$ *normalisiert* (Abschnitt 5.2). Diese Grenzwerte ergeben sich zum einen natürlich aus der Bildgröße, zum anderen aus dem willkürlich vorgegebenen Raum der Kamerawinkel. Dieser Raum wird so gewählt werden, daß ein sinnvolles Arbeiten möglich ist.
Vom *Eingabereiz* $u = (\omega, p)$ abhängig sind die *Aktivitäten* der Neuronen. Die Aktivität eines Neuron $r \in W$ berechnet sich aus dem Abstand $a_r = \|u - w_r\|^2$ von

Eingabereiz und dem Neuron zugeordneten Vektor. Als BMU (Best Matching Unit) bezeichnen wir das Neuron $s \in W$ mit dem geringsten Abstand:

$$\|u - w_s\| = \min_r \|u - w_r\|$$

für alle $r \in W$. Die Vektoren w_r aller Neuronen $r \in W$ werden dann entsprechend der Aktivitäten durch eine einfache Lernregel in Richtung von u verschoben. Die Lernschrittweiten sorgen dafür, daß diese Verschiebungen im Laufe des Trainings immer kleiner werden:

$$w_r^{neu} = w_r^{alt} + \epsilon\ h_{rs}(u - w_r^{alt})$$

Dabei gibt ϵ die Lernschrittweite an, und h_{rs} ist eine Funktion, die den - durch die Topologie gegebenen - *Nachbarschaftsabstand* von Neuron $r \in W$ zu Neuron $s \in W$ angibt. Die Nachbarschaftsfunktion h_{rs} hängt von einer Lernschrittweite σ ab, die mit der Zeit abnimmt, damit werden die Nachbarschaftsverbindungen mit der Zeit immer schwächer. Abhängig von der gegebenen Aktivität wird auch die Ausgabe berechnet. Wir haben jedem Neuron eine Matrix zugeordnet und berechnen die Ausgabe eines Neurons $r \in W$ durch $A_r p$ aus den Kamerakoordinaten. Dabei basiert die Ausgabe nur auf einer linearen Beziehung zu den Kamerakoordinaten, da diese in der Hauptsache für die Winkeländerungen maßgeblich sind. Die Ausgabe des gesamten Netzes errechnet sich dann als normalisierte Summe über die Ausgaben und Aktivitäten der einzelnen Neuronen:

$$\omega' = \sum_{r \in W} \frac{1}{a_T}\ a_r\ A_r\ p$$

Dabei ist a_T die Gesamtaktivität der Neuronen: $a_T = \sum_{r \in W}\ a_r\ ,\ a_r = \|u - w_r\|^2$

Nachdem wir die Ausgabe berechnet und das Netz in der Eingabeschicht adaptiert haben, kommen wir nun zur Adaption der Ausgabeschicht. Dafür benötigen wir den Mechanismus des in Abschnitt 5.3 erläuterten P-Reglers, um eine verbesserte Ausgabe zu generieren. Wir stellen also den Kamerakopf auf die durch das Netz gelieferten neuen Winkel ω' ein und registrieren die neuen Koordinaten p' des Szenenpunktes in den Kamerabildern. Mit Hilfe des P-Reglers wird dann der verbesserte Winkeloffset ω_P berechnet, der zum Lernen benutzt wird. Dabei kommt ein *iterativer Algorithmus* zur Anwendung, der aus der *linearen Regression* bekannt ist und erstmals von Widrow und Hoff [Wid60] vorgeschlagen wurde. Ein ähnlicher Algorithmus wurde auch von Ritter, Martinetz und Schulten [Rit91] verwendet. Wir berechnen einen *verbesserten Schätzwert* $A^* = \bar{A} + \Delta\bar{A}$ aus der Summe der Matrizen entsprechend der Aktivitäten durch:

$$\bar{A} = \sum_{r \in W} \frac{1}{a_T}\ a_r\ A_r$$

$$\Delta \bar{A} = \delta \, \left(\omega_P - \bar{A} \, p \right) \, p^T \tag{6.1}$$

Dabei ist $\delta = 1 \, / \, \|p\|^2$ gewählt.
Ein Trainingsschritt wird charakterisiert durch die Eingabe $u = (\omega, p)$ und die gelernte Ausgabe ω_P. Die Gesamtheit der Trainingsschritte ist eine Folge von Paaren: $(u^{(v)}, \omega_P^{(v)})$; $v = 1, 2, \ldots$.
Es approximieren genügend viele Einzelschritte die Wirkung eines einzigen Schrittes entlang der Richtung:

$$\sum_v \left(\omega_P^{(v)} - \bar{A}^{(v)} \, p^{(v)} \right) \, \left(p^{(v)} \right)^T \; = \; - \frac{dE(\bar{A})}{d\bar{A}}$$

Dabei ist $E(\bar{A})$ die zu minimierende Fehlerfunktion:

$$E(\bar{A}) = \frac{1}{2} \sum_v \left[\omega_P^{(v)} - \bar{A}^{(v)} \, p^{(v)} \right]^2$$

Man erkennt, daß das Verfahren zur Minimierung von $E(\bar{A})$ führt, indem es einen Gradientenabstieg 'im Mittel' bewirkt. Jeder Einzelschritt kann zu einer Erhöhung des Fehlers führen, langfristig über viele Adaptionsschritte hinweg ist aber eine Abnahme des Fehlers $E(\bar{A})$ garantiert.
Der verbesserte Schätzwert A^* wird genutzt, um genau wie bei der Adaption der Vektoren der Eingabeschicht, die Matrizen der Ausgabeschicht zu modifizieren:

$$A_r^{neu} = A_r^{alt} + \epsilon' h'_{rs} (A^* - A_r^{alt}),$$

für alle $r \in W$.
Die Lernschrittweite wird hier mit ϵ' bezeichnet und h'_{rs} ist wieder eine Nachbarschaftsfunktion die den Nachbarschaftsabstand der Neuronen $s, r \in W$ angibt. Sie hängt von einer Lernschrittweite σ' ab.

Verlauf der Trainingsparameter

Die Lernparameter ϵ und σ, bzw. ϵ' und σ' verändern sich mit der Anzahl der Lernschritte. Die Nachbarschaftsfunktionen h_{rs} , bzw. h'_{rs} hängen von den Aktivitätsradien σ und σ' ab.
Sei t_{max} die Anzahl der geplanten Lernschritte, dann ist zum Zeitpunkt des Lernschritts $1 \leq t \leq t_{max}$ die Lernschrittweite $\epsilon(t)$ gegeben durch:

$$\epsilon(t) = \epsilon_{max} \; (\epsilon_{min} \, / \, \epsilon_{max})^{t \, / \, t_{max}}$$

Innerhalb der Lernperiode von t_{max} Schritten, verringert sich die Lernschrittweite ϵ also von ϵ_{max} zu ϵ_{min}. Genauso folgt:

$$\sigma(t) = \sigma_{max} \; (\sigma_{min} \; / \; \sigma_{max})^{t \; / \; t_{max}} ,$$

$\epsilon'(t)$ und $\sigma'(t)$ folgen analog.
Als Nachbarschaftsfunktion h_{rs} wählen wir die Gaußfunktion:

$$h_{rs}(t) = \exp\left(-\frac{\|g_r - g_s\|^2}{2\,\sigma(t)^2}\right)$$

Dabei beschreibt $\|g_r - g_s\|$ den topologischen Abstand der beiden Neuronen $s, r \in W$. Durch den Verlauf von σ werden die Nachbarschaftsbeziehungen im Laufe einer Trainingsphase immer geringer. Die Nachbarschaftsfunktion h'_{rs} folgt analog.
Als Topologie verwenden wir nur rechteckige Gitterstrukturen, es sind aber auch andere, z.B. radialsymmetrische Gitter möglich. Dabei ist jedem Neuron ein Vektor $g_r = (d_1, d_2, .., d_{DIM}) \in \mathbb{N}^{DIM} \; ; r \in W$ zugeordnet, der seinen Platz in dem Gitter der Dimensionalität DIM beschreibt.

Im schematischen Überblick sieht der Algorithmus damit folgendermaßen aus:

1. Initialisierung der Vektoren w_r , $r \in W$ durch geeignete Anfangswerte. Initialisierung der Matrizen A_r , $r \in W$ durch geeignete Anfangswerte.

2. Stimuluswahl $u = (\omega, p)$, d.h Ablesen der momentanen Kamerawinkel ω , Wählen eines Szenenpunktes S und Ablesen seiner Bildkoordinaten p.

3. Bestimmung des Erregungszentrums $s \in W$ mit

   $$\|u - w_s\| = \min_r \|u - w_r\|$$

   für alle $r \in W$.

4. Berechnung des Kamerawinkeloffsets ω' nach:

   $$\omega' = \sum_{r \in W} \frac{1}{a_T} \; a_r \; A_r \; p$$

   Dabei $a_r = \|u - w_r\|$ die Aktivität der Neuronen und a_T die Gesamtaktivität der Neuronen: $a_T = \sum_{r \in W} \; a_r$

5. Einstellen der neuen Kamerawinkel $\omega + \omega'$ und Registrieren der Bildkoordinaten p' von S.

6. Adaption der Vektoren w_r , $r \in W$ gemäß

$$w_r^{neu} = w_r^{alt} + \epsilon\ h_{rs}(u - w_r^{alt})$$

7. Berechnen des verbesserten Ausgabewertes ω_P aus p'. Berechnen des verbesserten Schätzwertes

$$\bar{A} = \sum_{r \in W} \frac{1}{a_T}\ a_r\ A_r$$

$$A^* = \bar{A} + \delta\ \left(\omega_P - \bar{A}\ p\right)\ p^T$$

Dabei ist $\delta = 1 \,/\, \|p\|^2$ gewählt.
Führe Lernschritte durch gemäß

$$A_r^{neu} = A_r^{alt} + \epsilon' h'_{rs}(A^* - A_r^{alt}),$$

Gehe zu 2.

6.4 Die Dimension der topologischen Karte

Kohonenkarten besitzen anders als DCS fest *vorgegebene Topologie* und *Dimension*. Als Topologie ist üblicherweise ein rechteckiges Gitter gewählt. Die Dimension des Gitters ist entsprechend des durch die Eingabe zu repräsentierten Raumes zu wählen.
Ein Beispiel ist die Beobachtung eines Raumes durch zwei statische Kameras. Jeder Raumpunkt ist durch einen Punkt in der Bildebene jeder Kamera charakterisiert. Die Menge aller vierdimensionalen Vektoren, die sich so ergibt, repräsentiert jedoch nur einen dreidimensionalen Raum. Die topologische Karte ist dreidimensional zu wählen.

Betrachten wir zunächst nur monokulare Sakkaden. Wie in Abschnitt 3.1 gezeigt ist, teilt sich der mathematische Lösungsweg in zwei Abschnitte, die Berechnung des Sehstrahles und die Berechnung der neuen Winkel. Für jeden Sehstrahl ist offensichtlich genau eine Winkelstellung diejenige, unter der er in die Bildmitte zentriert ist, die Zuordnung von Sehstrahl zu Kamerawinkeln ist eindeutig. Man könnte also erwarten, daß eine zweidimensionale Netztopologie zu wählen ist, und jedes Neuron einen bestimmten Sehstrahl repräsentiert. Diese naheliegende Vermutung bestätigt sich bei genauerer Analyse allerdings nicht.
Betrachtet man die Menge der Vektoren $u = (\omega, p)$, die den gleichen Sehstrahl repräsentieren, so enthält sie unendlich viele Elemente, da ein Sehstrahl unter

verschiedenen Kamerawinkeln auf verschiedene Bildpunkte abgebildet wird. In einer zweidimensionalen Topologie würde von allen diesen Vektoren genau das gleiche Neuron angesprochen werden müssen, da ein Neuron einen Sehstrahl repräsentiert. Dies ist aber nicht möglich, da sich die Neuronenaktivitäten aus dem Abstand von Eingabevektor und Zentrumsvektor ergeben: $\|u - w_r\|^2$. Vektoren u, die genau den gleichen Sehstrahl beschreiben, werden zu unterschiedlichen Aktivierungen führen, das Gegenteil ist hier aber erwünscht.
Folglich muß die Dimension der Kohonenkarte entsprechend der Dimension des Eingabevektors gewählt werden. Im Fall von Sakkaden mit einer Kamera also dreidimensional (Bildpunktkoordinaten x_i, y_i und Vergenzwinkel θ).
Im Fall von Sakkaden mit beiden Kameras bedeutet dies einen fünfdimensionalen Raum unter ständiger symmetrischer Vergenz (Bildpunktkoordinaten x_l, y_l, x_r, Neigungswinkel ϕ und Vergenzwinkel θ).

Kapitel 7

Dynamische Zell-Strukturen

Dynamische Zell-Strukturen (DCS) bieten einen weitereren Ansatz, iteratives Lernen von Sakkaden mit neuronalen Netzen zu implementieren. Sie basieren auf Arbeiten von [Bru95], [Fri93a] und[Fri93b]. DCS verbinden den *inkrementellen* Aufbau eines Radialen-Basisfunktions-Netzwerks mit einer perfekten Topologie. Ein lokales Fehlermaß steuert das Einfügen von Neuronen derart, daß ein minimales Netzwerk ausgebildet wird, dessen Approximationsfehler unter einer vorgegebenen Toleranz liegt.

7.1 Einleitung

Dynamische Zell-Strukturen basieren nicht auf einer festen Topologie, sondern erlernen diese anhand der Trainingsbeispiele. Die Anzahl der Neuronen ist nicht vorher festgelegt, sondern wird entsprechend der gewünschten Fehlergenauigkeit in der Ausgabe selbständig erhöht, wobei das Training mit nur einem Neuron beginnt. Die Steuerung des Einfügens mit einem lokalen Fehlermaß in Verbindung mit der Ausbildung einer perfekten Topologie führt zu einer minimalen Anzahl von Neuronen, die die gewünschte Genauigkeit erzielt. DCS-Netze beschreiben deshalb keine Wahrscheinlichkeitsverteilung der Eingabedaten, wie Kohonenkarten. Diese Eigenschaft führt auch bei ungleichmäßig verteilten Eingabedaten, wie sie unter Realweltbedingungen auftreten, zu einer gleichmäßig guten Fehlergenauigkeit der Approximation.
Die Lernparameter der DCS sind konstant und nehmen nicht, wie bei Kohonenkarten, über die Zeit des Trainings ab. Bei Kohonenkarten war hier vor allem auch die richtige Wahl der Trainingsdauer nötig, die nur durch Versuche ermittelt werden kann. Oft wird man hier unnötig viele Trainingsschritte machen, ohne bessere Ergebnisse zu erreichen. Bei DCS wird diese Entscheidung vereinfacht, man kann z.B das Training abbrechen, wenn über eine bestimmte Anzahl von

Trainingschritten kein weiteres Neuron mehr eingefügt worden ist, oder der Mittelwert über die letzten Schritte geringer als die Toleranz ist. Die gewünschte Genauigkeit ist dann erreicht.

7.2 Notationen

Um den verwendeten Algorithmus zu beschreiben, führen wir folgende Notationen ein:

- Sei $\Omega \subset \mathbb{R}^K \;\; ; \;\; K \in \mathbb{N}$ der Eingaberaum der Kamerawinkel.
- Sei $\Omega' \subset \mathbb{R}^M \;\; ; \;\; M \in \mathbb{N}$ der Ausgaberaum der Kamerawinkeloffsets.
- Sei $P \subset \mathbb{R}^L \;\; ; \;\; L \in \mathbb{N}$ der Raum der Bildkoordinaten.
- Ein Eingabestimulus sei definiert als $u = (\omega, p)$ mit $\omega \in \Omega \; ; \; p \in P$.
- Sei $W = 1, 2, .., I$ die Menge der aktuellen Knoten. Dabei kann die Anzahl $I = |W|$ der Neuronen von Lernschritt zu Lernschritt wachsen.
- Seien $w_r \in \mathbb{R}^N \; ; \; r \in W \; ; \; N = K + L$ die Vektoren der Eingabeschicht.
- Seien $A_r \in \mathbb{R}^{L \times M} \; ; \; r \in W$ die Matrizen der Ausgabeschicht.
- Seien $r_r \in \mathbb{R} \; ; \; r \in W$ die Resourcen der Neuronen.
- Sei $C \in \mathbb{R}^{I \times I}$ die Adjazenzmatrix des Netzes.

7.3 RBF

Die beiden unter einer Eingabe $u = (\omega, p)$ am stärksten aktivierten Neuronen BMU (Best Matching Unit) und $SECOND$ sind gegeben durch:

$$\|u - w_{BMU}\| = \min_r \|u - w_r\| \;\; ; \; r \in W$$

$$\|u - w_{SECOND}\| = \min_r \|u - w_r\| \;\; ; \; r \in W \backslash \{BMU\}$$

Die Berechnung der Ausgabe des Netzes basiert auf einer Aktivierung nach *radialen Basisfunktionen.* Ist eine Eingabe u gegeben, so berechnen wir für alle

Neuronen aus der Nachbarschaft der BMU und für diese selbst eine Aktivität nach:

$$a_r = \frac{1}{1 + \sigma \, \|u - w_r\|^2} \quad ,$$

für alle $r \in Nh(BMU) \cup \{BMU\}$.
Dabei beschreibt $Nh(BMU)$ alle Neuronen, die eine Verbindung zur BMU besitzen, d.h in der Adjazanzmatriz C des Netzes gilt $C[BMU, r] > 0$ und $C[r, BMU] > 0$. Die Varianz der radialen Basisfunktionen ist mit σ bezeichnet. Die Ausgabe ergibt sich dann wie bei der Verwendung von Kohonenkarten als normalisierte Summe von Aktivitäten und der Matrixmultiplikation der jeweiligen dem Neuron zugeordneten Matrix A mit der Eingabe p. Dabei werden bei der Verwendung der DCS nur die Neuronen aus der Nachbarschaft der BMU und diese selbst betrachtet, weil in einer perfekt topologieerhaltenden Karte nur diese von ihren Abständen $\|u - w_r\|$ auch die am dichtesten benachbarten Neuronen sind.

$$\omega' = \sum_{r \in NH(BMU) \cup \{BMU\}} \frac{1}{a_T} \; a_r \; A_r \; p$$

Dabei ist a_T die Gesamtaktivität der Neuronen: $a_T = \sum_{r \in NH(BMU) \cup \{BMU\}} a_r$

Die Adaption der Matrizen erfolgt wieder über den Gradientenabstieg auf einer Fehlerfunktion. Wieder betrachten wir nur die Neuronen der Nachbarschaft der BMU und diese selbst. Wir berechnen einen verbesserten Schätzwert $A^* = \bar{A} + \Delta\bar{A}$ aus der Ergebnismatrix der $\bar{A}$ durch:

$$\bar{A} = \sum_{r \in W} \frac{1}{a_T} \; a_r \; A_r$$

$$\Delta\bar{A} = \delta \; \Big(\omega_P - \bar{A} \; p\Big) \; p^T$$

Dabei ist $\delta = 1 \; / \; \|p\|^2$ gewählt. Diesen verbesserten Schätzwert A^* benutzen wir dann, um genau wie bei der Adaption der Vektoren der Eingabeschicht die Matrizen der Ausgabeschicht zu modifizieren:

$$A_r^{neu} = A_r^{alt} + \eta \; \frac{a_r}{a_T} \; (A^* - \bar{A}), \tag{7.1}$$

für alle $r \in NH(BMU) \cup \{BMU\}$.
Der Adaptionsschritt (7.1) ähnelt der bei Kohonenkarten verwendeten Adaption (6.1), unterscheidet sich aber doch darin, daß die Korrekturmatrix hier nicht von A_r^{alt} abhängt.

7.4 Aufbau einer perfekten Topologie

Ein entscheidender Nachteil der Kohonenkarten ist, daß die Topologie vor Beginn des Trainings festgelegt werden muß. Dies erfordert a priori Wissen über die Struktur und Dimension des zu repräsentierenden Raumes. Der zu beschreibende Unterraum ist in seiner Struktur jedoch häufig nicht bekannt. Auch besteht die Möglichkeit, daß dieser Unterraum sich in verschiedene Abschnitte unterschiedlicher Dimension aufteilt. In diesem Fall erhält man mit Kohonenkarten nur noch eine *suboptimale* Topologie, und es es kann sein, daß im Netzwerk benachbarte Neuronen auf im Eingabesignalraum weit voneinander benachbarte Werte abgebildet werden [Mar92]. Eine ähnliche Adaption der Ausgabewerte von benachbarten Neuronen ist dann nicht mehr sinnvoll. An eine erfolgreiche Konvergenz des Kohonen-Algorithmus ist folglich eine *a priori* vorhandene genaue Kenntnis der zu erwartenden Untermannigfaltigkeit geknüpft.

Wir werden einen Algorithmus vorstellen, der ohne a priori Wissen über die Untermannigfaltigkeit eine *perfekt topologierepräsentierende Struktur* aufbaut. Ausgehend von nur einem Neuron wird das Netzwerk wachsen und dabei die Topologie ständig so adaptiert, daß sie perfekt erhalten bleibt.

Wir bedienen uns dabei des Begriffes der *Voronoi-Polygone* V_i, einer Menge von Punkten $w_i, i = 1, .., N$ im Raum $\mathbb{R}^D$ durch

$$V_i = \left\{ v \in \mathbb{R}^D \mid \|v - w_i\| \leq \|v - w_j\| \; \forall j \right\}$$

definiert. Das Dual des Voronoi-Diagramms ist die *Delaunay-Triangulation*. Sie entsteht, wenn Punkte w_i, deren zugehörige Voronoi-Polygone V_i aneinandergrenzen durch Linien miteinander verbunden werden. Jede Kante der Delaunay-Triangulation korrespondiert also zu einer ihr senkrechten Kante des Voronoi-Diagramms.
Im Zweidimensionalen ist der entstehende Graph der Delaunay-Triangulation eine Triangulation der N Punkte in der Ebene, d.h. er ist so darstellbar, daß sich keine Verbindungslinien schneiden und jede abgegrenzte Region ein Dreieck darstellt.
Die folgenden Beschreibungen folgen Arbeiten von T. Martinetz und K. Schulten [Mar94] und beschränken sich auf den zweidimensionalen Fall.
Die Existenz eines planaren Graphen macht Eulers Formel (Bollabás 1979) anwendbar, die hier besagt, daß für die Zahl M der Kanten der Delaunay-Triangulation im Vergleich zur Zahl N der Voronoi-Polygone gilt:

$$M \leq 3N - 6 \tag{7.2}$$

Aufgrund der Dualität besteht auch das Voronoi-Diagramms aus höchstens $3N-6$ Kanten.

Der Rechenaufwand zur Konstruktion der Delaunay-Triangulation steigt genau wie die Rechenzeit zur Konstruktion des Voronoi-Diagramms mit der Zahl M der zu bildenden Kanten an, entsprechend:

$$O(M\, log_2 M) \tag{7.3}$$

Im zweidimensionalen Fall wächst die Kantenanzahl M wie gezeigt höchstens linear mit der Zahl der Neuronen an (7.2). Die Konstruktion der Delaunay-Triangulation ist also in

$$O(N\, log_2 N) \tag{7.4}$$

Schritten möglich. In höherdimensionalen Fällen ist die Kantenzahl M schwerer abzuschätzen, wir nehmen deshalb den zweidimensionalen Fall als Motivation.

Das Ziel muß es sein, einen Algorithmus zu entwickeln, der zwischen den Neuronen des Netzwerkes eine Topologie entsprechend der Delaunay-Triangulation aufbaut. Dabei soll die Topologie während des Trainings aufgebaut werden und muß flexibel auf das Einfügen von weiteren Neuronen reagieren können.
Dann ist die Anforderung nach a priori Wissen über die Struktur der *Untermannigfaltigkeit* hinfällig und der Fall, daß benachbarte Neuronen auf weit entfernte Gebiete im Eingabesignalraum abgebildet werden, kann nicht mehr auftreten.
Die Beziehung 7.4 liefert eine wichtige Information zur Rekonstruktion des Netzwerkes nach dem Einfügen eines neuen Neurons. Da die Topologie noch nicht richtig angepaßt ist, kann es zu einem lokal zeitweise hohen Fehler kommen. Es ist also sinnvoll, zwischen dem Einfügen zweier Neuronen eine Anzahl an Trainingsschritten zu machen, ohne weitere Neuronen einzufügen. Ein Maß für diese Zeitspanne liefert die Beziehung 7.4, auch wenn sie nur für den zweidimensionalen Fall bewiesen ist. Wenn die Konstruktion der Delaunay-Triangulation eine Anzahl an Schritten in der Größenordnung $O(N\, log_2 N)$ benötigt, dann ist dies ein sinnvolles Maß für die Wartezeit zwischen dem Einfügen zweier Neuronen.

Die Lösung des Problems der Adaption der lateralen Neuronenverbindungen liefert ein Ansatz mit einer nichtlinearen Hebb'schen Lernregel. Die Adjazenzmatrix C wird dabei wie folgt adaptiert:

$$\Delta C_{ij} = \begin{cases} 1: & a_i \cdot a_j \geq a_k \cdot a_l \quad \forall\, (1 \leq k, l \leq I) \\ 0: & sonst \end{cases}$$

Die $a_r; r \in W$ sind die Aktivierungen der Neuronen entsprechend der Radialen Basisfunktionen: $a_r = 1/(1 + \sigma\, \|u - w_r\|^2)$. Diese Adaption führt zu einer perfekten topologieerhaltenden Karte [Mar94].

Aus der Vorgabe, die Topologie während des Trainings zu erlernen, kann man sofort folgern, daß es auch eine Möglichkeit geben muß, bestehende Kanten wieder zu löschen. Als Entscheidungskriterium bietet sich hier das Alter der Verbindungen an. Wird eine Verbindung nicht zeitweise aufgefrischt, so fällt sie unter eine Schranke und wird gelöscht.
Die *Adjazenzmatrix*, in der die Verbindungen der Delaunay-Triangulation gespeichert sind, besteht deshalb nicht nur aus binären Einträgen, sondern aus positiven reellen Zahlen, die widerspiegeln, wie '*stark*' zwei Neuronen verbunden sind. Bei einer Verbindungsstärke von Null existiert keine Verbindung. Die Verbindungsstärken zweier Neuronen in der Adjazenzmatrix sind nicht notwendigerweise gleich, d.h die Adjazenzmatrix ist nicht symmetrisch.
Bestehende Verbindungen können wieder gelöscht werden, wenn sie zu alt sind, d.h. wenn sie lange nicht verstärkt wurden. Durch eine Konstante τ werden Verbindungen abgeschwächt, so daß sie unter eine Schranke μ fallen und dann gelöscht werden, wenn sie nicht zeitweise verstärkt werden. Die adaptierte Hebb-Lernregel sieht damit folgendermaßen aus:

$$C_{ij}(t+1) = \begin{cases} 1 & : \; a_i \cdot a_j \geq a_k \cdot a_l \;\; \forall (1 \leq k, l \leq I) \\ 0 & : \; C_{ij}(t) < \mu \\ \tau C_{ij}(t) & : \; sonst \end{cases}$$

Diese Regel erfährt noch eine weitere Veränderung dahingehend, daß die Summe der von einem Neuron ausgehenden Verbindungsgewichte auf Eins normalisiert ist. Wird eine Verbindung verstärkt, müssen die anderen entsprechend verringert werden.
Nach der Präsentation einer Eingabe u wird zuerst getestet, ob ein neues Neuron eingefügt werden soll. Zu diesem Entscheidungskriterium kommen wir später genauer, bei der Besprechung der Resourcen.
Wird kein neues Neuron eingefügt und die Anzahl den aktuellen Neuronen ist größer als Eins, dann ist entscheidend, ob zwischen BMU und $SECOND$ bereits eine Verbindung besteht. Ist dies nicht der Fall, so wird diese Verbindung neu angelegt, mit einer Stärke abhängig von der Anzahl der Verbindungen von BMU bzw. $SECOND$. Die anderen von BMU bzw. $SECOND$ ausgehenden Verbindungen werden renormalisiert.
Besteht bereits eine Verbindung zwischen BMU und $SECOND$, so wird diese in beiden Richtungen um die Summe der Verbindungsstärken zu allen anderen benachbarten Neuronen multipliziert mit τ verstärkt. Die anderen Verbindungsstärken nehmen entsprechend ab, und die Normalisierung bleibt erhalten.
Dadurch kann der Fall eingetreten sein, daß von der BMU ausgehende Verbindungen ein zu niedriges Gewicht erreicht haben, und gelöscht werden sollen. Als Entscheidungskriterium für das Löschen einer Verbindung benutzen wir die bereits erwähnte Variable μ. Dabei wird eine Verbindung zwischen BMU und

$DEST$ gelöscht, wenn das Gewicht der von BMU nach $DEST$ gehenden Verbindung kleiner ist als μ geteilt durch die Anzahl der von der BMU ausgehenden Verbindungen $Outdeg(BMU)$ und wenn für die Verbindung von $DEST$ nach BMU das Gewicht kleiner ist als μ geteilt durch $Outdeg(DEST)$. Wird eine Verbindung gelöscht, so müssen natürlich auch wieder die Gewichte der anderen ausgehenden Verbindungen normalisiert werden.

Damit kommen wir zu dem bisher übergangenen Teil des Einfügens eines neuen Neurons. Im ersten Lernschritt ist noch kein Neuron vorhanden, hier kreieren wir schlicht ein Neuron. Ab dem zweiten Lernschritt benötigen wir ein Entscheidungskriterium für das Einfügen eines Neurons, wir benutzen dazu die noch zu erläuernden Resourcen. Ist die Entscheidung für das Einfügen eines neuen Neurons positiv ausgefallen, gibt es zwei Wege, um fortzufahren. Der erste Fall tritt ein, wenn bisher das Netz nur aus einem Neuron besteht oder der Abstand von Eingabe und BMU größer ist als der Abstand von BMU und $SECOND$. Hier wird ein neues Neuron mit dem Gewicht der Eingabe initialisiert und eine Verbindung zwischen diesem und der BMU erstellt. Die ausgehenden Verbindungen der BMU werden normalisiert.
Im zweiten Fall wird ein neues Neuron mit einem Gewicht initialisiert, das dem Mittel der Gewichte von BMU und $SECOND$ entspricht. Bei der Ausgabematrix wird entsprechend verfahren. Es werden Verbindungen von neuem Neuron zu BMU und $SECOND$ erstellt und es wird wieder normalisiert. Existiert eine Verbindung zwischen BMU und $SECOND$, so wird diese gelöscht.

Die Adaption der Gewichte der Neuronen der Eingabeschicht erfolgt über eine Regel ähnlich der bei Kohonenkarten:

$$w_r^{neu} = w_r^{alt} + \epsilon_{Nh}\; s_r^{\nabla}\; \frac{a_r}{a_T}\; (u - w_r^{alt}),$$

für alle $r\ inNh(BMU)$ und

$$w_{BMU}^{neu} = w_{BMU}^{alt} + \epsilon_{BMU}\; s_{BMU}^{\nabla}\; \frac{a_{BMU}}{a_T}(u - w_{BMU}^{alt})$$

Die Parameter ϵ_{Nh} und ϵ_{BMU} sind Konstanten, die sich während des Trainings nicht verändern.
Wir werden zwei Versionen von Adaptionen vergleichen: Bei der ersten ist s_r^{∇} schlicht Eins, also redundant. Bei der zweiten ist s_r^{∇} entsprechend eines *Gradientenabstiegs* auf den Zentrumsvektoren gewählt. Wir wählen s_r^{∇} bei einer Adaption in Richtung des positiven Gradienten und Null sonst. Eine genauere Beschreibung der Gradientenabstieg-Modulation gibt der folgende Abschnitt.

7.5 Gradientenabstieg

Das Einfügen von Neuronen ist an den lokalen Fehler gekoppelt. Die spätere Adaption der Zentrumsvektoren nach der einfachen Kohonenregel berücksichtigt diesen Fehler jedoch nicht. Eine Koppelung der Adaption an den Ausgabefehler erhält man über eine erweiterte Kohonenregel unter Beachtung der Ableitung des Ausgabefehlers nach den Zentrumsvektoren.

Die Ableitung der Fehlerfunktion $F = (A^* - \bar{A})^2$ nach dem Zentrumsvektor w_r ; $r \in W$ ergibt sich aus:

$$\frac{\partial F}{\partial w_r} = \frac{\partial F}{\partial \bar{A}} \frac{\partial \bar{A}}{\partial a_r} \frac{\partial a_r}{\partial w_r} \quad ; \quad \frac{\partial F}{\partial \bar{A}} = \sum_{k,l} \left[\left(\omega^P - \bar{A} p \right) \frac{p^T}{\|p\|^2} \right]$$

$$\frac{\partial \bar{A}}{\partial a_r} = \frac{A_r - \bar{A}}{\sum_{r \in W} a_r} \quad ; \quad \frac{\partial a_r}{\partial w_r} = -2\sigma \left(w_r - u \right) a_r^2$$

Zusammengefaßt folgt:

$$\frac{\partial F}{\partial w_r} = 2\sigma \sum_{k,l} \left[\left(\omega^P - \bar{A} p \right) \frac{p^T}{\|p\|^2} \right] \left[A_r - \bar{A} \right] \frac{a_r^2}{\sum_{r \in W} a_r} \left(u - w_r \right)$$

Am Erfolgreichsten hat es sich erwiesen, aus einem Teil der Ableitung eine Signum-Funktion zu definieren. Ist der Ausdruck

$$2\sigma \sum_{k,l} \left[\left(\omega^P - \bar{A} p \right) \frac{p^T}{\|p\|^2} \right] \left[A_r - \bar{A} \right] a_r$$

größer als Null, so ist $s_r^\nabla = 1$, sonst Null.

Die Adaption der Zentrumsvektoren erfolgt dann durch:

$$w_r^{neu} = w_r^{alt} + \epsilon_{Nh} \; s_r^\nabla \; \frac{a_r}{a_T} \; (u - w_r^{alt}),$$

für alle $r \in Nh(BMU)$ und

$$w_{BMU}^{neu} = w_{BMU}^{alt} + \epsilon_{BMU} \; s_{BMU}^\nabla \; \frac{a_{BMU}}{a_T} (u - w_{BMU}^{alt})$$

7.6 Resourcen

Die Entscheidung über das Einfügen eines neuen Neurons ist an die Güte der Ausgabe des neuronalen Netzes gekoppelt. Eine *lokale Fehlerfunktion* liefert ein Maß für die Güte der Arbeit des Netzes. Jedem Neuron $r \in W$ ist ein *Resource-Wert* r_r zugeordnet, der zu Beginn Null ist, und aktualisiert wird, wenn dieses Neuron BMU ist. Bei jeder Aktualisierung wird ein *gleitender Mittelwert* zwischen bisheriger Resource und neuem Fehler $failure$ gebildet:

$$\Delta r_{BMU} = \beta \, (failure - r_{BMU})$$

$$r_{BMU} = r_{BMU} + \Delta r_{BMU}$$

Der gleitende Mittelwert ermöglicht es, wähernd des Trainings immer einen Eindruck vom lokalen Fehler im Vergleich zur Toleranz zu haben. Die Wahl der Konstante $0 < \beta < 1$ verhindert, daß ein einzelner hoher Fehler im Verhältnis zu vielen geringen Fehlern überbewertet wird.

7.7 Abbruchkriterium

Das Ziel der DCS ist es, ein neuronales Netz aufzubauen, dessen Funktionsapproximation über den ganzen Eingabebereich hin einen Fehler unterhalb der Toleranz aufweist. Diese Idee setzt ein Kriterium voraus, das entscheidet, wann der Fehler unterhalb der Toleranz liegt, und den Abbruch des Trainings auslöst.
Als Abbruchkriterien sind mehrere Lösungen denkbar. Eine Möglichkeit wäre es, eine Testmenge an Eingabedaten zu nehmen, deren Fehler unterhalb der Toleranz liegen muß, um den Abbruch auszulösen. Ein Nachteil dieser Methode ist der hohe Rechenaufwand, den ein ständiges Kontrollieren der Testmenge mit sich bringt. Eine andere Möglichkeit wäre es, den Abbruch auszulösen, wenn über eine bestimmte Anzahl an Trainingsschritten kein neues Neuron eingefügt worden ist. Diese Idee führt jedoch zu Problemen mit der Forderung nach perfekt topologieerhaltenden Karten. Die Wartezeit zwischen dem Einfügen zweier Neuronen steigt mit deren Anzahl entsprechend $n\, log_2\, n$ an. Die Zahl der Trainingsschritte, während deren kein Neuron eingefügt worden sein darf, kann nicht unabhängig von der Neuronenanzahl n gewählt werden.
Das von uns verwendete Abbruchkriterium verwendet einen Mittelwert über die letzten k Schritte. Eine Schrittzahl von $k = 1000$ beim Off-Line Lernen mit Kohonenkarten und $k = 5000$ beim Lernen von DCS hat sich als sinnvoll erwiesen und ist groß genug, um nicht auf kurzzeitige Schwankungen zu reagieren. Ein Vorteil dieser Methode ist die schnelle Berechenbarkeit des Mittelwertes.

7.8 Algorithmus

Alle Trainingsparameter sind Konstanten und bleiben während des Trainings unverändert. Diese Bedingung ergibt sich aus der unbestimmten Trainingslänge der DCS-Netze. Sie macht die Dynamischen Zell-Strukturen auch hervorragend anwendbar auf verschiedenste Aufgabenstellungen. Während bei Kohonenkarten beispielsweise bei Off-Line und On-Line Training unterschiedliche Trainingsparameter gewählt werden, um ein optimales Verhalten zu erreichen, können DCS mit einem Parametersatz allen Anwendungen gerecht werden.

Im schematischen Überblick sieht der Algorithmus folgendermaßen aus:

1. Stimuluswahl $u = (\omega, p)$, d.h Ablesen der momentanen Kamerawinkel ω , Wählen eines Szenenpunktes S und Ablesen seiner Bildkoordinaten p.

2. Bestimmung des Erregungszentrums $BMU \in W$ mit

   $$\|u - w_{BMU}\| = \min_r \|u - w_r\|$$

   für alle $r \in W$ und des nach der BMU am besten matchenden Neurons $SECOND \in W \backslash \{BMU\}$:

   $$\|u - w_{SECOND}\| = \min_r \|u - w_r\|$$

   für alle $r \in W \backslash \{BMU\}$

3. Berechnung des Kamerawinkeloffsets ω' nach:

   $$\omega' = \sum_{r \in Nh(BMU) \cup \{BMU\}} \frac{1}{a_T} \; a_r \; A_r \; p$$

   Dabei ist

   $$a_r = \frac{1}{1 + \sigma \; \|u - w_r\|^2}$$

   die Aktivität der Neuronen und a_T die Gesamtaktivität der Neuronen: $a_T = \sum_{r \in Nh(BMU) \cup \{BMU\}} \; a_r$

4. Einstellen der neuen Kamerawinkel $\omega + \omega'$ und Registrieren der Bildkoordinaten p' von S.

5. Anhand der neuen Bildkoordinaten p' wird der Fehler $failure$ berechnet.

6. Adaption der Resource der BMU entsprechend:

$$r_{BMU} = r_{BMU} + \beta * (failure - r_{BMU})$$

7. Adaption des Netzwerkes:
Die Adaption teilt sich in zwei Wege. Der erste Weg ist das Einfügen eines neuen Neurons. Er wird eingeschlagen, wenn die Resource der BMU und der Fehler $failure$ größer als die Toleranz ist ($r_{BMU} > tolerance$ und $failure > tolerance$), oder das erste Neuron überhaupt eingefügt werden soll. Wir rufen hier die Funktion $insertion$ auf.
Der zweite Weg ist die normale Adaption des Netzes, bestehend aus der Adaption der Gewichte $w_r\ ,\ r \in W$, der Adaption der Matrizen $A_r\ ,\ r \in W$ und der Adaption der Verbindungen, d.h der Adjazenzmatrix C. Beschrieben wird dies durch die Funktionen $adaptWeights$, $adaptMatrices$ und $adaptLinks$.

8. Bilden des mittleren Fehler über die letzten 5000 Schritte: $failure(5000)$. Ist $failure(5000) < tolerance$, dann ist das Training erfolgreich beendet. Sonst: Gehe zu 1.

Im folgenden sollen die oben erwähnten Funktionen näher spezifiziert werden:

- $insertion$
  Ist die Anzahl der Neuronen noch gleich Null, so wird ein Neuron mit dem Gewicht der Eingabe u eingefügt. Die Ausgabematrix A kann mit Zufallswerten initialisiert werden, oder auch entsprechend eines Vorwissens.

  Ist die Resource der BMU größer als die Toleranz: $r_{BMU} > tolerance$, dann gibt es zwei Möglichkeiten: Das Einfügen eines Neurons mit Verbindung zur BMU oder das Einfügen eines Neurons zwischen BMU und $SECOND$.

  Ein Neuron mit Verbindung zur BMU fügen wir ein, wenn das Netz genau ein Neuron enthält, d.h kein Neuron $SECOND$ existieren kann, oder wenn der Abstand von BMU zur Eingabe u größer ist, als der Abstand von BMU und $SECOND$, d.h das neue Neuron würde von der Tolopogie her nicht zwischen BMU und $SECOND$ liegen. Wir fügen ein Neuron mit dem Gewicht der Eingabe u ein und ordnem diesem die gleiche Matrix A_{BMU} zu. Die Verbindungen zwischen BMU und neuem Neuron, einschließlich der Normalisierung der Verbindungen, entstehen durch eine Funktion $connect$: $connect(BMU, NewNode)$ und $connect(NewNode, BMU)$.

  Im anderen Fall fügen wir ein Neuron zwischen BMU b und $SECOND$ ein, mit dem gemittelten Gewicht: $w_{new} = (w_{BMU} + w_{SECOND})/2$ und der gemittelten Matrix $A_{new} = (A_{BMU} + A_{SECOND})/2$ von BMU und $SECOND$. Wir verbinden die Neuronen durch: $connect(BMU, NewNode)$,

$connect(NewNode, BMU)$, $connect(SECOND, NewNode)$ und $connect(NewNode, SECOND)$. Die Verbindungen zwischen BMU und $SECOND$ werden gelöscht. Als Folge dessen müssen die Verbindungsgewichte renormalisiert werden: $renormalizeNodeLinks(BMU)$ und $renormalizeNodeLinks(SECOND)$.

- $adaptWeights$
  Wir adaptieren nur die Gewichte der Neuronen aus der Nachbarschaft $Nh(BMU)$ und der BMU selbst:

  $$w_r^{neu} = w_r^{alt} + \epsilon_{Nh}\ s_r^{\nabla}\ (u - w_r^{alt}),$$

  für alle $r \in Nh(BMU)$ und

  $$w_{BMU}^{neu} = w_{BMU}^{alt} + \epsilon_{BMU}\ s_{BMU}^{\nabla}\ (u - w_{BMU}^{alt})$$

- $adaptMatrices$
  Auch die Adaption der Matrizen betrifft nur die Nachbarschaft $Nh(BMU) \cup \{BMU\}$. Wir berechnen erst einen verbesserten Schätzwert A^*, wie auch bei den Kohonenkarten beschrieben, und adaptieren die Matrizen damit wie folgt:

  $$\bar{A} = \sum_{r \in Nh(BMU) \cup \{BMU\}} \frac{1}{a_T}\ a_r\ A_r$$

  $$A^* = \bar{A} + \delta\ \Big(\omega_P - \bar{A}\ p\Big)\ p^T,$$

  mit $\delta = 1\ /\ \|p\|^2$.

  $$A_r^{neu} = A_r^{alt} + \eta\ \ \frac{a_r}{a_T}(A^* - \bar{A}),$$

  für alle $r \in Nh(BMU) \cup \{BMU\}$.

- $adaptLinks$
  Die Adaption der Verbindungen hängt davon ab, ob zwischen BMU und $SECOND$ bereits eine Verbindung besteht.

  Ist dies nicht der Fall, werden sie einfach verbunden durch: $connect(BMU, SECOND)$ und $connect(SECOND, BMU)$.

  Ist dies bereits der Fall, so werden die Verbindungen zwischen BMU und $SECOND$ verstärkt und die restlichen Verbindungen von BMU und $SECOND$ entsprechend der Normalisierung vermindert. Dies geschieht durch eine Funktion $adaptNodeLinks(BMU, SECOND)$ und

$adaptNodeLinks(SECOND, BMU)$.
Durch das Abschwächen der Verbindungen können bei einigen die Schwellen unterschritten worden sein, sie werden gelöscht. Die Bedingung dafür lautet:
Ist für ein Neuron $DEST \in Nh(BMU)$:

$$C[BMU, DEST] < \mu/Outdeg(BMU)$$

und

$$C[DEST, BMU] < \mu/Outdeg(DEST),$$

dann wird die Verbindung zwischen $DEST$ und BMU gelöscht, und die Verbindungen von $DEST$ werden renormalisiert :
$renormalizeNodeLinks(DEST)$.
Die Koppelung von μ an $Outdeg()$ ist von Bedeutung bei der Adaptivität an unterschiedliche Dimensionalitäten. Da die Gewichte immer normalisiert sind, ist bei höheren Dimensionalitäten das Gewicht einer einzelnen Verbindung natürlich geringer als bei kleinen Dimensionalitäten. Deshalb muß auch die Schranke für das Löschen einer Verbindung angepaßt sein.
Wurde mindestens eine Verbindung gelöscht, so werden abschließend die Verbindungen der BMU renormalisiert :
$renormalizeNodeLinks(BMU)$.

- $connect(n1, n2)$
  Sei $newWeight = 1 \;/\; (Outdeg(n1)+1)$. Es folgt für alle Neuronen $DEST \in Nh(n1)$:

  $$C[n1, DEST] = C[n1, DEST] * (1 - newWeight),$$

  und für die neue Verbindung:

  $$C[n1, n2] = newWeight$$

  Damit gilt auch weiterhin :

  $$\sum_{DEST \in Nh(n1)} C[n1, DEST] \;=\; 1 \;,$$

  wenn dies auch vorher der Fall war.

- $adaptNodeLinks(n1, n2)$
  Wir berechnen über alle Neuronen $DEST \in Nh(n1)\backslash\{n2\}$:

  $$C[n1, DEST] = (1 - \tau) * C[n1, DEST]$$

  Die Verbindung von $n1$ zu $n2$ wird erhöht um:

  $$C[n1, n2] = C[n1, n2] + \Delta C$$

  $$\Delta C = \sum_{DEST \in Nh(n1)\backslash\{n2\}} \tau * C[n1, DEST]$$

  Wieder bleibt die Normalisierung erhalten.

- $renormalizeNodeLinks(n1)$
  Eine Renormalisierung über alle Nachbarn $DEST$ eines Neurons $n1$ erfolgt entsprechend:

  $$DIV = \sum_{DEST \in Nh(n1)} C[n1, DEST]$$

  $$C[n1, DEST] = C[n1, DEST] \ / \ DIV,$$

  für alle $DEST \in Nh(n1)$.

Kapitel 8

Off-Line-Lernen der Sakkadensteuerung

In den letzten drei Kapiteln wurden die neuronalen Ansätze Kohonenkarte und Dynamische Zell-Strukturen beschrieben sowie die Aufgabenstellungen spezifiziert. Die für die unterschiedlichen Anwendungen nötigen Ein- und Ausgaben der neuronalen Netzwerke wurden analysiert. Bei Kohonenkarten wurden die Topologien festgelegt.
In diesem Kapitel werden Ergebnisse eines Off-Line Trainings mit Kohonenkarten und DCS beschrieben werden. Die inverse Kinematik des in den Kapiteln 2 und 3 vorgestellten Kameramodells wird erlernt werden.

Das Off-Line Training ist die Grundlage für das On-Line Training unter Realweltbedingungen. Es basiert auf der Annahme bestimmter Werte für die *intrinsischen Kameraparameter*. Je besser diese mit der Realität übereinstimmen, desto schneller wird im On-Line Training das vortrainierte Netzwerk auf die realen Bedingungen angelernt und die größtmögliche Genauigkeit erreicht. Aus Kapitel 4 ist bekannt, daß reale Trainingszeit weitaus teurer ist im Vergleich zum Modelltraining, deshalb ist eine möglichst große Annäherung des Modells an die Realität wünschenswert.

Die Lernparameter sind für alle Versuche jeweils gleich gewählt. Es sind dies 9 Parameter bei Kohonenkarten gegenüber 7 Parametern bei DCS.
Da die Initialisierung der Netze mit Zufallszahlen und die Präsentation der Trainingsstimuli von großer Bedeutung für das Lernen sind, wurden sie in jeweils zehn Versuchen variiert.

8.1 Monokularer Fall

Der einfachste Fall sind monokulare Sakkaden. Die Sakkade wird dabei durch Veränderung von Neigungs- und Vergenzwinkel ausgeführt, der Schwenkwinkel bleibt konstant.
Entscheidend für die zu erreichende Genauigkeit der Sakkaden ist besonders die Wahl des Arbeitsbereiches der Kamera im Verhältnis zur Neuronenanzahl. Bei Verwendung einer Kamera ist dieser wegen der Unabhängigkeit des Neigungswinkels ϕ gegeben durch den Arbeitsbereich des Vergenzwinkels θ. Eine sinnvolle Wahl des Arbeitsbereiches ist $\theta \in [-40°, +40°]$, der für die gemachten Versuche verwendet wurde.
Der angegebene Fehler ist jeweils der Mittelwert über die letzten 1000 Trainingsschritte.

8.1.1 Lernen mit Kohonenkarten

Der Ansatz der Kohonenkarten bietet viele Variationsmöglichkeiten in den Trainingsparametern. Dies sind einmal die Größe des Netzes, die Wahl der Topologie und die Anzahl der Trainingsschritte, aber auch die Lernschrittweiten ϵ^{Center}, ϵ^{Out} und die Nachbarschaftsradien σ^{Center} und σ^{Out}, jeweils mit Anfangs- und Endwert.

Trainingsparameter:	
σ^{Center}	$1.5 - 0.3$
ϵ^{Center}	$1.0 - 0.01$
σ^{Out}	$1.5 - 0.3$
ϵ^{Out}	$0.9 - 0.4$

Abbildung 8.1: Die Trainingsparameter der Kohonenkarten sind für alle Versuche gleich gewählt. Die Tabelle zeigt jeweils Anfangs- und Endwert während der festgelegten Trainingsdauer.

Abbildung 8.1 zeigt die für alle Versuche mit Kohonenkarten gewählten Lernparameter. Sie sind so gewählt, daß angenähert optimale Ergebnisse erzielt wurden. Die gezeigte Versuchsreihe (Abbildung 8.2) variiert die Netzgröße von 64 über 125 bis zu 216 Neuronen. Die Topologien enthalten in jeder Dimension die gleiche Anzahl an Neuronen, da keine a priori Information über eine bessere Verteilung zur Verfügung steht (siehe Abschnitt 6.4).

Beim Erlernen des Kameramodells besteht die Möglichkeit, die korrekten Winkeloffets der Sakkaden überwacht zu lernen. Um gleich den Bezug zum On-Line Training herzustellen, wurde jedoch das Error-Feedback-Lernen mit dem beschriebenen P-Regler angewendet (Abschnitt 5.3).

Trainingsparameter und Ergebnisse:			
Neuronen	$4 \times 4 \times 4 = 64$	$5 \times 5 \times 5 = 125$	$6 \times 6 \times 6 = 216$
Trainingsschritte	5000	5000	5000
Feedback	Proportional	Proportional	Proportional
1 Fehler	3.48	2.97	2.41
2	3.49	2.87	2.55
3	3.61	2.93	2.36
4	3.53	2.91	2.53
5	3.52	2.92	2.34
6	3.49	2.86	2.55
7	3.49	2.84	2.41
8	3.41	2.92	2.55
9	3.49	2.85	2.57
10	3.46	2.82	2.38
Mittel	3.50	2.89	2.47
Standardabweichung	0.034	0.041	0.085

Abbildung 8.2: Fehler in Pixel bei Variation der Netzgröße unter Verwendung von Kohonenkarten.

Wie erwartet verringert sich der Fehler mit Zunahme der Neuronenanzahl. Die Verbesserung der Genauigkeit von 64 Neuronen zu 216 Neuronen beträgt etwa 1 Pixel. Dieser relativ gute Wert mit nur 64 Neuronen ist auf die starke lineare Komponente der inversen Kinematik zurückzuführen.
Abbildung 8.3 zeigt den Verlauf des Fehlers beim Training einer Kohonenkarte mit 216 Neuronen. Durch die Initialisierung mit Zufallswerten ist der Fehler zuerst sehr groß, nimmt aber schnell ab. Bereits nach 3500 Schritten ist eine Genauigkeit von 3 Pixel erreicht, die schließlich auf 2.5 Pixel verbessert wird.

8.1.2 Lernen mit DCS

Die Traininsparameter der Dynamischen Zell-Strukturen sind gegeben durch die Lernschrittweiten ϵ_{BMU}, ϵ_{Nh} und η, sowie dem Radius der radialen Basisfunktionen σ und der Gleitwertkonstante β bei der Berechnung der Resourcen. Für die Adaption der Topologie sind die Parameter τ und μ zu wählen.
Aus der Vorgabe einer Toleranz folgen dann als Trainingsergebnis die Neuronenanzahl und die notwendige Trainingsschrittzahl. Die Topologie wird, wie in Kapitel 7 beschrieben während des Trainings selbst aufgebaut.
Die Adaption der Zentrumsvektoren wird zwischen den beschriebenen Ansätzen der einfachen Kohonenregel und der Gradientenabstieg-Modulation variiert (Abschnitt 7.4).

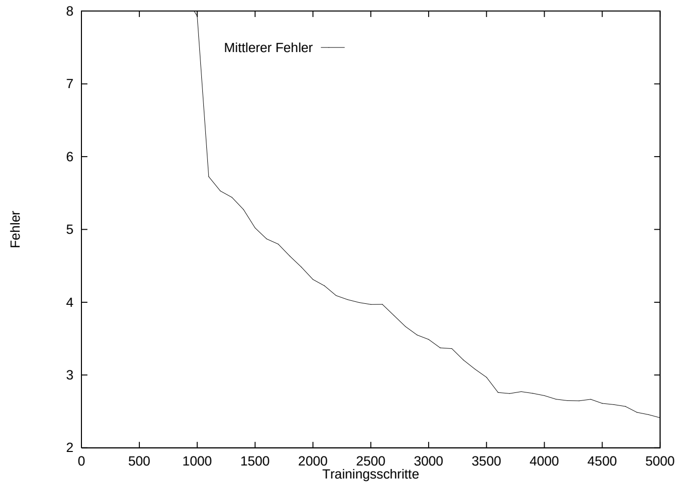


Abbildung 8.3: Verlauf des Fehlers beim Training einer Kohonenkarte mit 216 Neuronen.

In den Versuchen haben wir uns am Fehler der Versuche mit Kohonenkarten (Abbildung 8.2) orientiert und dargestellt, mit welcher Neuronenanzahl mit DCS der gleiche Fehler möglich ist. Die Trainingsergebnisse der beiden Versionen (Einfache Kohonenregel und Gradientenabstieg-Modulation) auf verschiedene Toleranzen sind in den Abbildungen 8.5, 8.6 und 8.7 dargestellt.
Die Gradienten-Modulation führt zu besseren Ergebnissen als die einfache Kohonenregel, indem der Ausgabefehler auch bei der Adaption der Zentrumsvektoren berücksichtigt wird.
Es ist zu erkennen, daß die Neuronenanzahl unter Verwendung von DCS jedesmal geringer ist, als mit Kohonenkarten. Diese Verbesserungen sind auf die fehlerabhängige Neuronenverteilung in Verbindung mit der perfekten Topologie zurückzuführen.
Bei der Versuchsreihe wurden alle Trainingsparameter gleich gewählt (siehe Abbildung 8.4).
Abbildung 8.8 zeigt den Fehlerverlauf und die Neuronenanzahl beim Training einer Dynamischen Zell-Struktur auf eine Toleranz von 2.5 Pixel (0.5%). Zu erkennen ist der zuerst schnelle Anstieg der Neuronenanzahl, der mit dem schnellen Abnehmen des Fehlers korreliert. Je mehr Neuronen vorhanden sind, desto größer wird der Abstand zwischen dem Einfügen zweier Neuronen. Dies geht einerseits auf den geringen Fehler nahe der Toleranz zurück, der die Resourcen der Neuronen nur langsam ansteigen läßt, aber besonders auf die Bedingung $n \log_2 n$ Schritte zwischen dem Einfügen zweier Neuronen zu warten, um eine perfekte Topologie zu erhalten (Abschnitt 7.4).

Trainingsparameter:	
η	0.7
β	0.03
σ	1.0
$\epsilon_{BMU}/\epsilon_{Nh}$	0.05/0.006
τ	0.1
μ	0.33

Abbildung 8.4: Die Trainingsparameter der Dynamischen Zell-Strukturen sind für alle Versuche gleich gewählt. Sie besitzen keine Abhängigkeit von der Trainingslänge, da diese im voraus nicht festgelegt ist.

Trainingsparameter und Ergebnisse:		
Version	Einfache Kohonenregel	Gradienten Modulation
Fehler	3.5	3.5
Feedback	Proportional	Proportional
1 Neuronen/Schritte	62/9462	62/9435
2	63/9867	58/8236
3	65/10619	52/6864
4	67/11110	60/8706
5	66/11020	53/7064
6	64/10305	53/6857
7	64/10234	53/6867
8	66/11089	61/9290
9	66/10886	63/9700
10	65/10829	58/8823
Mittel	65/10542	57/8184
Standardabweichung	1.3/460	3.7/1017

Abbildung 8.5: DCS-Training auf einen Fehler von 3.5 Pixel

Trainingsparameter und Ergebnisse:		
Version	Einfache Kohonenregel	Gradienten Modulation
Fehler	2.9	2.9
Feedback	Proportional	Proportional
1 Neuronen/Schritte	107/32347	73/13782
2	108/32938	68/11613
3	104/30303	79/16748
4	102/29191	82/17701
5	100/27916	89/21708
6	102/29119	77/15596
7	104/30527	85/19338
8	113/36187	88/21218
9	105/31319	84/19092
10	102/28947	92/23088
Mittel	105/30879	82/17988
Standardabweichung	2.9/1855	5.9/2890

Abbildung 8.6: DCS-Training auf einen Fehler von 2.9 Pixel

Trainingsparameter und Ergebnisse:		
Version	Einfache Kohonenregel	Gradienten Modulation
Fehler	2.5	2.5
Feedback	Proportional	Proportional
1 Neuronen/Schritte	147/65241	114/36788
2	147/65652	123/43677
3	152/70653	109/33549
4	146/64830	120/41267
5	131/49611	121/41890
6	156/74817	118/39715
7	147/65027	119/39920
8	150/69134	123/42962
9	135/53608	114/36453
10	161/80309	113/35940
Mittel	147/65788	117/39216
Standardabweichung	6.0/6252	4.0/2827

Abbildung 8.7: DCS-Training auf einen Fehler von 2.5 Pixel

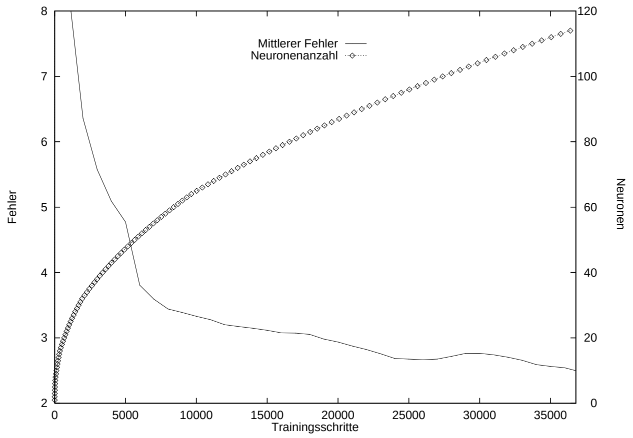


Abbildung 8.8: Verlauf des Fehlers und der Neuronenanzahl beim Training einer Dynamischen Zell-Struktur. Die Zentrumsvektoren wurden entsprechend eines Gradientenabstiegs adaptiert.

8.1.3 Neuronenverteilung in Winkelbereichen

Eine Besonderheit von Dynamischen Zell-Strukturen ist das Einfügen von Neuronen entsprechend des Fehlers in der Ausgabe. Über die Resourcen der Neuronen wird der Fehler lokal akkumuliert, und gegebenenfalls wird ein neues Neuron eingefügt, wenn die Toleranz überschritten wird. Eine interssante Frage stellt sich darin, wo diese Bereiche hoher Neuronendichte liegen, d.h. in welchen Bereichen mehr Neuronen nötig sind, um die Funktion ausgabenorientiert gleich gut zu approximieren.
Dieser Abschnitt lehnt sich an Kapitel 3.5 an, in dem einige Aspekte der zu approximierenden Abbildungen untersucht wurden.

Im monokularen Fall wurde gezeigt, daß die Abbildung stark vom momentanen Vergenzwinkel abhängt. Im folgenden wird deshalb die Neuronenverteilung unter Verwendung von DCS in Abhängigkeit des Vergenzwinkels näher untersucht. Abbildung 8.9 zeigt die Neuronenverteilung nach dem Training und ihr gegenübergestellt die Verteilung der ursprünglichen Einfügeorte neuer Neuronen für beide Versionen von DCS. Die Graphiken zeigen jeweils das Mittel der zehn Versuche aus Abbildung 8.7.
Zu bemerken ist, daß die Eingangsreize im Training bezüglich des Vergenzwinkels gleichverteilt sind.
Zu erkennen ist jeweils die größere Neuronenanzahl bei höheren Vergenzwinkeln sowie ein erhöhtes Einfügen von Neuronen bei hohen Vergenzwinkeln. Obwohl die neuronalen Netze mit Winkeln von $-40°$ bis $40°$ trainiert wurden, sind am

äußeren Rand keine Neuronen zu finden, es sind dort jedoch Neuronen eingefügt worden. Dieser Effekt geht auf die Adaption der Zentrumsvektoren nach der Kohonenregel zurück und ist unter Verwendung der Gradientenabstieg-Modifikation abgeschwächt.
In Abschnitt 3.5 wurde gezeigt, daß sich die zu approximierende Inverse-Kinematik Funktion mit wachsendem Vergenzwinkel immer stärker verändert, so daß gerade in den hohen Winkelbereichen eine größere Anzahl an Neuronen notwendig ist, um die Funktion ausgabenorientiert gleich gut zu approximieren. Diese Erkenntnisse finden sich hier bestätigt. Die DCS-Netze besitzen bei höheren Vergenzwinkeln mehr Neuronen. Der Vorteil der Gradientenabstieg-Modifikation zeigt sich darin, daß auch bei der Adaption der Zentrumsvektoren der Ausgabefehler berücksichtigt wird, somit der Nachteil der einfachen Kohonenregel ausgeglichen wird und sich auch nahe der Grenzen des Winkelbereichs noch Neuronen befinden.

Die höhere Anzahl von Neuronen bei hohen Vergenzwinkeln legt die Vermutung nahe, daß hier auch besonders viele Neuronen eingefügt wurden. Abbildung 8.9 zeigt aber, daß die meisten Neuronen nicht direkt an den Grenzen des Winkelbereichs, sondern leicht davon entfernt eingefügt wurden. Der Grund hierfür liegt darin, daß ein Neuron meistens zwischen den beiden am besten matchenden Neuronen BMU und $SECOND$ eingefügt wird. Nur, wenn der Abstand $\|Stimulus - BMU\|$ größer ist als $\|BMU - SECOND\|$, wird es am Stimulusort eingefügt.
Es erscheint vielleicht hilfreich, alle Neuronen am Stimulusort einzufügen, um die Neuronen schon beim Einfügen weiter an den Rand des Winkelbereichs zu plazieren. Dadurch können aber Neuronen sehr dicht nebeneinander liegen, die sich während des weiteren Trainings nur sehr langsam wieder voneinander entfernen, auch wenn dies sinnvoll wäre. Nachteile entstehen dann dadurch, daß in einigen Fällen in der Umgebung ein weiteres Neuron neu eingefügt wird, bevor die beiden dicht zusammenliegenden Neuronen sich entfernen können.

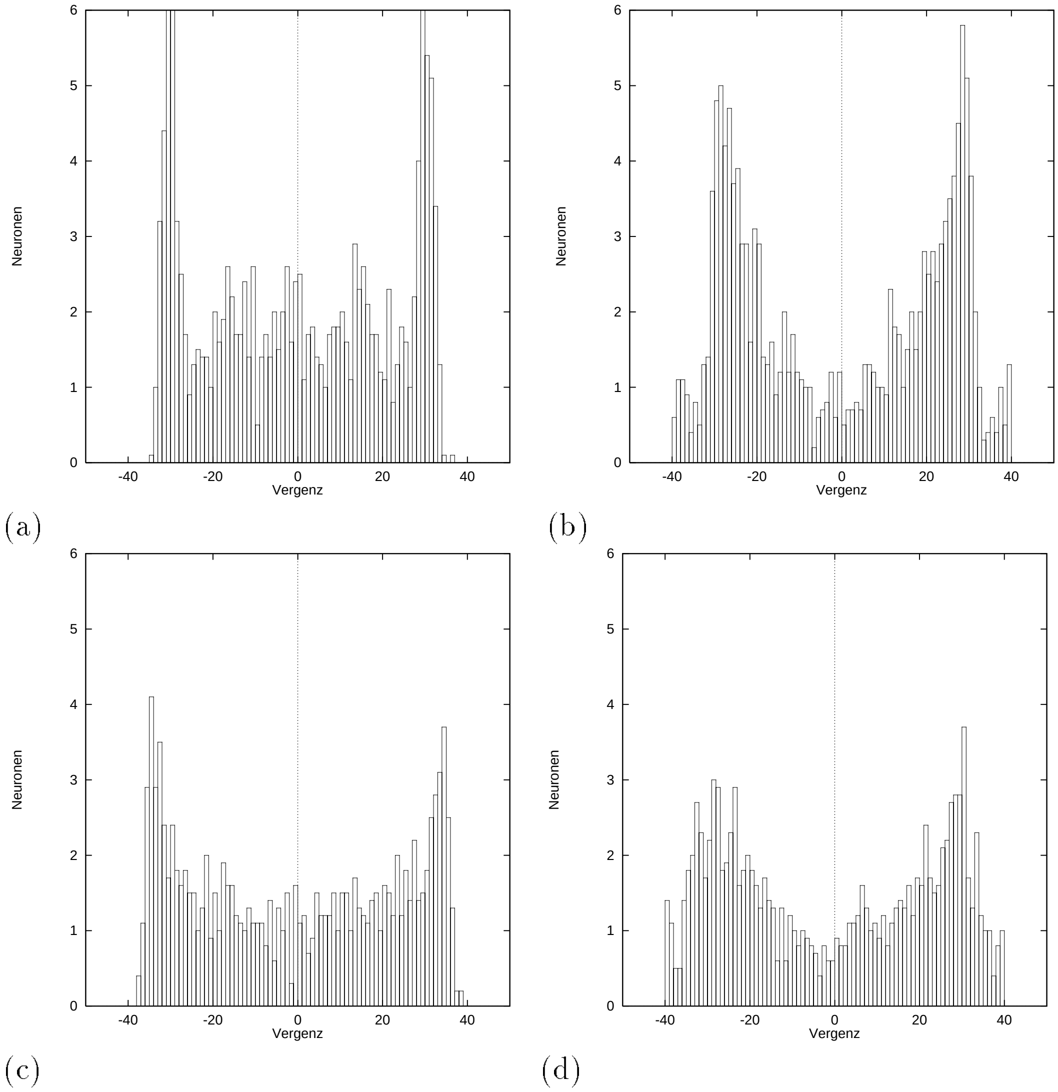


Abbildung 8.9: Neuronenverteilung in Abhängigkeit vom Vergenzwinkel bei DCS unter Verwendung von Version 1: Einfache Kohonenregel (a) und Version 2: Gradienten-Modulation (c). Dem gegenübergestellt sind die Einfügeorte neuer Neuronen bei Version 1 (b) und Version 2 (d). Die Darstellung ist gemittelt über 10 Versuche (siehe Abbildung 8.7).

8.1.4 Perfekte Lernvorgabe vs. P-Regler

In diesem Abschnitt werden Versuche gegenübergestellt, die sich in der Vorgabe von Trainingswerten entsprechend dem P-Regler gegenüber dem Lernen des perfekten Winkeloffsets unterscheiden. Wie die Abbildung 8.10 zeigt, ergeben sich keine wesentlichen Unterschiede im Ergebnis, wenn die Lernvorgabe zwischen P-Regler und Perfekter Lernvorgabe variiert wird. Interessanterweise ist bei Kohonenkarten der Fehler etwas niedriger und bei DCS die vorgegebene Toleranz von 2.5 Pixel, das entspricht bei einer Bildgröße von 512×512 etwa 0.5%, sogar in weniger Lernschritten erreicht, wenn als Lernvorgabe nur die ungenauen Werte des P-Reglers genommen werden. Daß bei richtiger Einstellung des P-Reglers eine schnellere Konvergenz erzielt wird, als bei einer perfekten Trainingsvorgabe, ist ein Effekt, der auch von Mayhew [May91] schon beobachtet worden ist.

Ergebnisse:	
Neuronen	216
Schritte	5000
Feedback	Perfekt
1	2.44
2	2.58
3	2.77
4	2.40
5	2.57
6	2.34
7	2.57
8	2.38
9	2.61
10	2.59
Mittel	2.52
Standardabweichung	0.234
Vergleich	2.47

(a)

Ergebnisse:	
Version	3
Fehler	2.5
Feedback	Perfekt
1	123/43267
2	123/43169
3	123/41407
4	110/34271
5	122/42448
6	123/43109
7	106/31757
8	126/45585
9	124/43859
10	113/36169
Mittel	119/40503
Standardabweichung	5.9/3863
Vergleich	117/39216

(b)

Abbildung 8.10: Vergleich zwischen P-Regler und perfekter Lernvorgabe als Lernvorgabe bei Kohonenkarten (a) und DCS (b). Die dargestellten Versuche sind mit perfekter Lernvorgabe gemacht, der Vergleichswert mit P-Regler entstammt der Tabelle 8.2 bzw. 8.7.

8.2 Binokularer Fall

Die nächste Stufe besteht in der Nutzung beider Kameras des Kamerakopfes und damit auch des Schwenkwinkels. In den Versuchen wird davon ausgegangen, daß vor der Sakkade *symmetrische Vergenz* herrscht, das bedeutet, die Vergenzwinkel sind gleich. Da die symmetrische Vergenz nach der Sakkade immer gegeben ist, ist es auch nicht nötig, verschiedene Vergenzwinkel vor der Ausführung zuzulassen, um den Kamerakopf mit Sakkaden zu steuern. Erst wenn mehrere Arten von Augenbewegungen zusammen die Steuerung ergeben, wird dies vielleicht notwendig sein.
Der gewählte Bereich ist für den Neigungswinkel $[-30°, +30°]$ und für die Vergenzwinkel $[0°, 10°]$.

8.2.1 Lernen mit Kohonenkarten

Trainingsparameter und Ergebnisse:	
Neuronen	$3 \times 3 \times 3 \times 3 \times 3 = 243$
Trainingsschritte	10000
Feedback	Proportional
1 Fehler	2.65
2	2.59
3	2.53
4	2.58
5	2.44
6	2.50
7	2.47
8	2.55
9	2.56
10	2.52
Mittel	2.54
Standardabweichung	0.047

Abbildung 8.11: Fehler im Pixel unter Verwendung von Kohonenkarten.

Bei der Verwendung beider Kameras unter ständiger symmetrischer Vergenz besteht die Eingabe des Netzes aus einem fünfdimensionalen Vektor aus Kamerawinkeln und Bildpunkten. Die Kohonenkarte ist, wie in Abschnitt 5.2 erklärt, auch fünfdimensional zu wählen. Selbst mit nur drei Neuronen pro Dimension ist das gesamte Netz dann 243 Neuronen groß, mit vier Neuronen pro Dimension wären es schon 1024 Neuronen. Die gemachten Versuche beziehen sich deshalb auf Kohonenkarten mit 243 Neuronen. Die Anzahl der Trainingsschritte wurde gegenüber den Versuchen mit einer Kamera auf 10000 erhöht, um der erhöhten Neuronenanzahl Rechnung zu tragen.

8.2.2 Lernen mit DCS

Trainingsparameter und Ergebnisse:		
Version	Einfache Kohonenregel	Gradienten Modulation
Fehler	2.5	2.5
Feedback	Proportional	Proportional
1 Neuronen/Schritte	116/37949	79/16450
2	110/34251	83/18385
3	110/34283	90/21937
4	115/37530	82/18055
5	110/34180	84/18863
6	105/31268	87/20780
7	107/32499	84/18898
8	115/37519	88/20711
9	116/37899	81/17581
10	110/34244	86/19892
Mittel	111/35162	84/19155
Standardabweichung	3.1/2049	2.6/1340

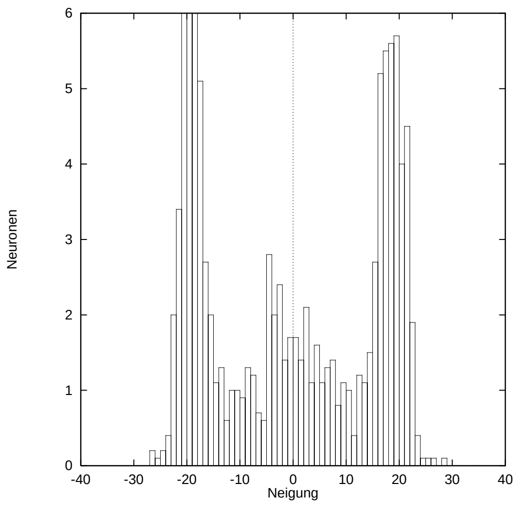
Abbildung 8.12: DCS-Training auf einen Fehler von 2.5 Pixel

Der Vergleich zwischen Kohonenkarten (Abb. 8.11) und DCS (Abb. 8.12) zeigt deutlich den Unterschied. Während der Fehler von 2.5 Pixel (0.5%) mit 243 Neuronen in der Kohonenkarte erreicht wurde, sind bei Dynamischen Zell-Strukturen nur 111 bzw. 84 Neuronen notwendig. Die Gradienten-Modulation führt wieder zu besseren Ergebnissen als die einfache Kohonenregel.

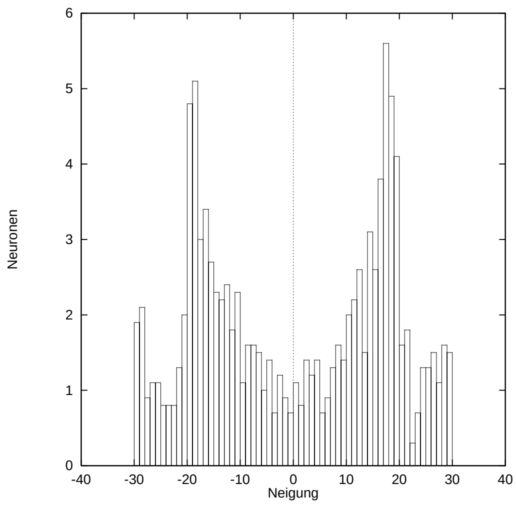
8.2.3 Neuronenverteilung in Winkelbereichen

Die im folgenden beschriebenen Erkenntnisse folgen analog dem Abschnitt 8.1.3. Einziger Unterschied ist, daß der interessante Winkelbereich, über den die Neuronenverteilung aufgetragen ist, im binokularen Fall der Neigungswinkel ist.
Es sind wieder die Mittelwerte aus zehn Versuchen (Abbildung 8.12) dargestellt. Zu erkennen ist in Abbildung 8.13 die zum Rand des trainierten Winkelbereichs

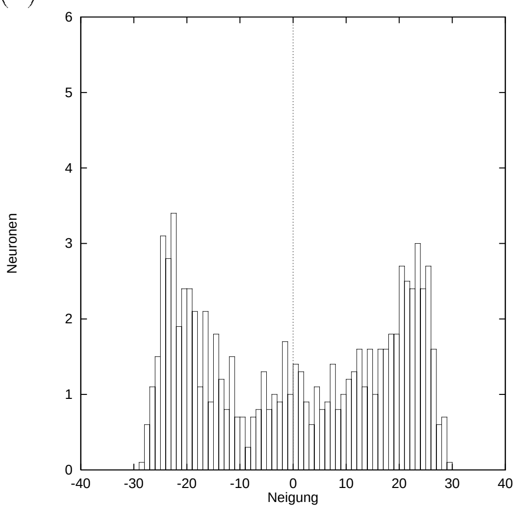
von $-30°$ bis $30°$ hin wachsende Neuronenanzahl. Dies bestätigt wieder die in Kapitel 3.5 anhand des Kameramodells gemachte Vermutung. Der Vorteil der Gradientenabstieg-Version zeigt sich wieder darin, daß sich auch in den Bereichen nahe der Grenzwerte noch Neuronen befinden. Gerade dies sind die Bereiche, in denen viele Neuronen nötig sind, um die Funktion gut zu approximieren.



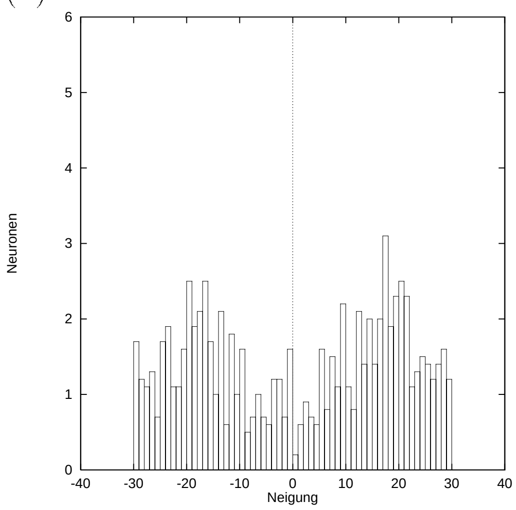
(a)



(b)



(c)



(d)

Abbildung 8.13: Neuronenverteilung in Abhängigkeit vom Neigungswinkel bei DCS unter Verwendung von Version 1: Einfache Kohonenregel (a) und Version 2: Gradienten-Modulation (c). Dem gegenübergestellt sind die Einfügeorte neuer Neuronen bei Version 1 (b) und Version 2 (d). Die Darstellung ist gemittelt über 10 Versuche.

Kapitel 9

On-Line-Lernen der Sakkadensteuerung

In diesem Kapitel wird die Anwendung der adaptiven Sakkadenregelung unter *Realweltbedingungen* untersucht. Nach den im letzten Kapitel gewonnenen Erkenntnissen über das Training unter Modellbedingungen sind auch hier gute Ergebnisse zu erwarten. Die Versuche unter Realweltbedingungen verlaufen vom Aufbau her analog zu den Versuchen mit konventioneller Kamerakalibrierung (Kapitel 4).

Es werden Kohonenkarten und DCS-Netze verwendet werden, die im Modelltraining eine Genauigkeit von 2.5 Pixel, also etwa 0.5%, erreicht haben.
Im Modelltraining ist zu erkennen, daß sehr viele Trainingsschritte notwendig sind, um den kleinsten Fehler zu erreichen. Die $5000 - 10000$ Trainingsschritte bei Kohonenkarten, sowie mehrere 10000 Schritte bei DCS stellen eine starke Beanspruchung der Kamera sowie des Roboters dar. Außerdem wäre ein solches Training bei einer Zeit von 2-3 Sekunden pro Trainingsschritt sehr zeitaufwendig. Durch die Verwendung vortrainierter Netze lassen sich jedoch viele Trainingsschritte einsparen. Sie bieten den Vorteil, schon eine ausgebildete Topologie aufzuweisen, deren Aufbau besonders viel Zeit in Anspruch nimmt. Diesen Aufbau verlagern wir in das Modelltraining und adaptieren im Training unter Realweltbedingungen nur noch die Ausgabematrizen der Netzwerke. Wie gezeigt werden wird, ergeben sich so schon mit 1000 Trainingsschritten sehr gute Ergebnisse. Ein Training dieser Länge dauert bei einer Zeit von zwei Sekunden pro Schritt ca. 35 Minuten.
Es ist zu erwähnen, daß im Training unter Realweltbedingungen der Raum der Eingabestimuli kleiner ist als im Modelltraining. Dies bedingt sich einfach aus der Reichweite des Roboters. Es ist aber selbstverständlich der gleiche Bereich gewählt, wie bei den Versuchen mit konventioneller Kamerakalibrierung.

Um die hervorragende Eignung der neuronalen Netze unter dieser Aufgabenstellung zu verdeutlichen, teilt sich das On-Line Training in zwei Abschnitte: Im

ersten wird das vortrainierte Netz auf die Realweltbedingungen angelernt, und im zweiten erfolgt nach der Verstellung des *optischen Freiheitsgrades* Zoom eine Anpassung auf die neuen Bedingungen.
Die Möglichkeit der neuronalen Ansätze, auch unter verschiedenen Zoomstellungen zu arbeiten, ist im Vergleich zur konventionellen Kamerakalibrierung besonders interessant, da das vorgestellte Kameramodell (Kapitel 2) davon ausgeht, daß sich Neigungs- und Vergenzachse im optischen Zentrum der Kamera schneiden. Eine Verstellung des Zoom bringt aber eine Verschiebung des *optischen Zentrums* mit sich, die vom Modell nicht unterstützt wird.
Die Verschiebung des optischen Zentrums könnte natürlich durch einen weiteren Parameter in das Modell eingebunden werden, würde dann aber auch eine komplexere Kalibrierung erfordern.

9.1 Monokularer Fall

Abbildung 9.1 zeigt die Trainingsparameter und die erreichten Winkelbereiche beim Training einer Kohonenkarte bzw. Dynamischen Zell-Struktur unter Realweltbedingungen.
Die Traingsparameter der DCS entsprechen denen des Modelltrainings, während die Parameter der Kohonenkarten in den Anfangswerten leicht verringert sind, um die Reichweite der bei Kohonenkarten globalen Adaption zu verringern. Bei DCS, die eine lokale Adaption besitzen, ist eine Anpassung der Trainingsparameter deshalb nicht notwendig.

Trainingsparameter:	
Neuronen	$6 \times 6 \times 6 = 216$
Trainingsschritte	1000
σ^{Out}	$1.0 - 0.3$
ϵ^{Out}	$0.6 - 0.4$
Fehler	1.25
Standardabweichung	0.79
Winkelbereiche:	
Neigungswinkel	$-17°/-4°$
Vergenzwinkel	$-1°/13°$

(a)

Trainingsparameter:	
Neuronen	126
Trainingsschritte	1000
η	0.7
σ	1.0
Fehler	1.75
Standardabweichung	1.20
Winkelbereiche:	
Neigungswinkel	$-15°/-2°$
Vergenzwinkel	$-2°/12°$

(b)

Abbildung 9.1: Trainingsparameter und erreichte Winkelbereiche des Kamerakopfes beim Training einer Kohonenkarte (a) und DCS (b).

Die Abbildungen 9.2 und 9.3 zeigen den Fehlerverlauf während des Trainings. Der Fehler ist jeweils der Mittelwert über die letzten 500 Sakkaden, wobei die

Abschnitte 'Anlernen auf die realen Bedingungen' und 'Neues Anlernen auf den verstellten Zoom' getrennt zu betrachten sind.
Die erreichte Genauigkeit von 1.25 bzw. 1.75 Pixel liegt weit unter der vortrainierten Genauigkeit von 2.5 Pixel. Der Grund hierfür liegt in den kleineren Winkelbereichen des realen Trainings gegenüber dem Modelltraining. Gerade in den hohen Winkelbereichen sind größere Fehler zu erwarten, wie die Analyse der Inversen Kinematik zeigt (Abschnitt 3.5). Da die nach dem Training erreichte Fehlergenauigkeit ein Mittelwert ist, ist bei Kohonenkarten zu erwarten, daß der Fehler in kleinen Winkelbereichen geringer als der Mittelwert und in höheren größer als dieser ist.

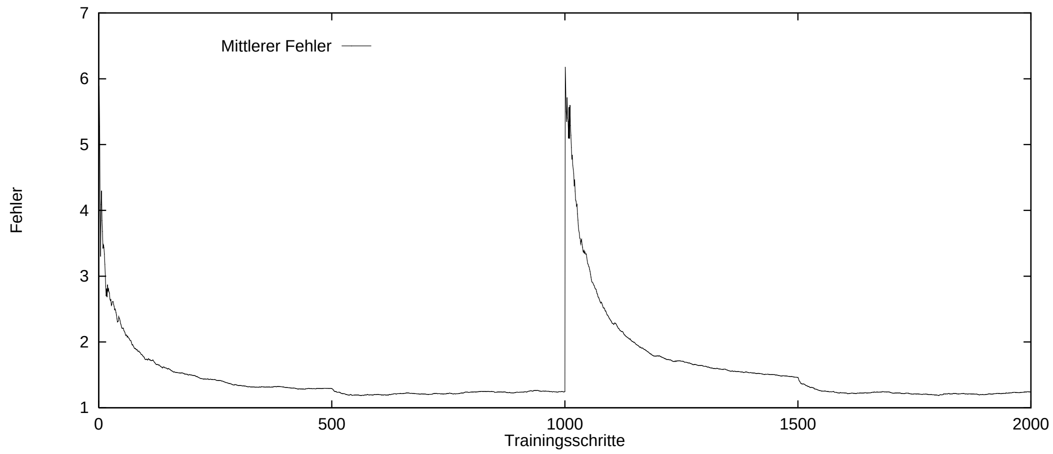


Abbildung 9.2: Fehlerverlauf beim Training eines Kohonennetzes unter Realweltbedingungen.

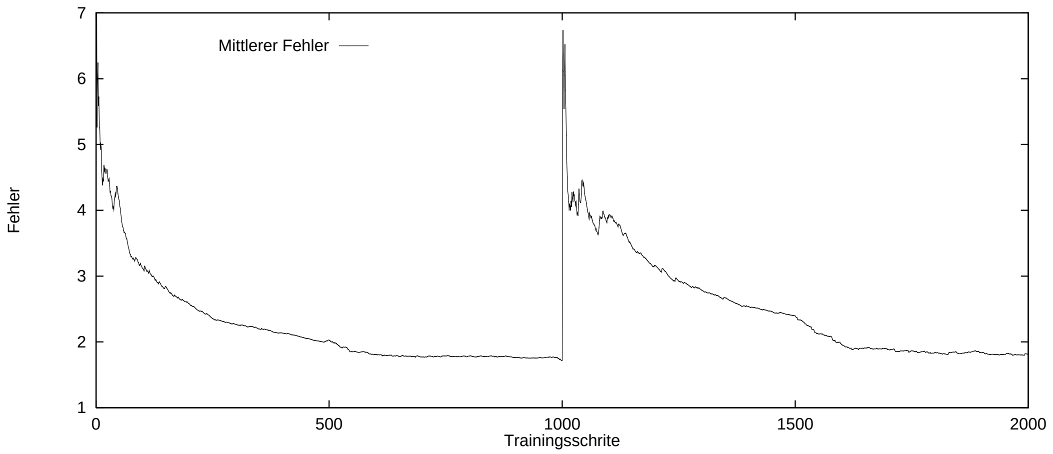


Abbildung 9.3: Fehlerverlauf beim Training einer Dynamischen Zell-Struktur unter Realweltbedingungen.

Bei DCS werden die Neuronen entsprechend des Fehlers verteilt (Abschnitte 8.1.3 und 8.2.3), um eine gleichmäßige Fehlergenauigkeit über den gesamten trainierten Bereich zu erhalten. Aber auch hier reichen die wenigen Neuronen aus, den Fehler in kleinen Winkelbereichen unter der Toleranz zu halten.
Ein On-Line Training mit den Winkelbereichen des Off-Line Trainings ist aufgrund der geringen Roboterbeweglichkeit nicht möglich.

Die gute Qualität der neuronalen Ansätze zeigt sich besonders im Rückblick auf die Ergebnisse mit konventioneller Kamerakalibrierung. Während dort eine Genauigkeit von 2.4 Pixel erreicht wurde, sind es hier weniger als 1.8 Pixel.

9.2 Binokularer Fall

Betrachten wir im folgenden die Versuchsergebnisse unter Verwendung beider Kameras mit symmetrischer Vergenz. Abbildung 9.4 zeigt die Trainingsparameter und Winkelbereiche. Die Abbildungen 9.5 und 9.6 zeigen den Fehlerverlauf wähernd des Trainings. Dieses teilt sich wieder in zwei Abschnitte mit Anlernen auf die realen Bedingungen und neues Anlernen auf die Zoomverstellung. Der Zoom wurde dabei nur an einer Kamera verstellt. Es ist die Kamera gewählt, von der auch die Y-Koordinate für die Berechnungen genommen wird.

Trainingsparameter:	
Neuronen	$3^5 = 243$
Trainingsschritte	1000
σ^{Out}	$1.0 - 0.3$
ϵ^{Out}	$0.5 - 0.4$
Fehler	2.2
Standardabweichung	1.15
Winkelbereiche:	
Schwenkwinkel	$-2°/11°$
Neigungswinkel	$-16°/-2°$
Vergenzwinkel	$3°/5°$

(a)

Trainingsparameter:	
Neuronen	124
Trainingsschritte	1000
η	0.7
σ	1.0
Fehler	2.0
Standardabweichung	1.29
Winkelbereiche:	
Schwenkwinkel	$-4°/9°$
Neigungswinkel	$-14°/-2°$
Vergenzwinkel	$2°/4°$

(b)

Abbildung 9.4: Trainingsparameter und erreichte Winkelbereiche des Kamerakopfes beim Training einer Kohonenkarte (a) und DCS (b) unter Verwendung beider Kameras mit ständiger symmetrischer Vergenz.

Der Fehler ist wieder der Mittelwert über die letzten 500 Sakkaden. Durch die Abhängigkeit der beiden Y-Koordinaten des Lichtpunktes und den Ungenauigkeiten des realen Systems ist es nur möglich, den Fehler bezüglich einer Y-Koordinate zu minimieren. Der angegebene Fehler ist deshalb der Mittelwert

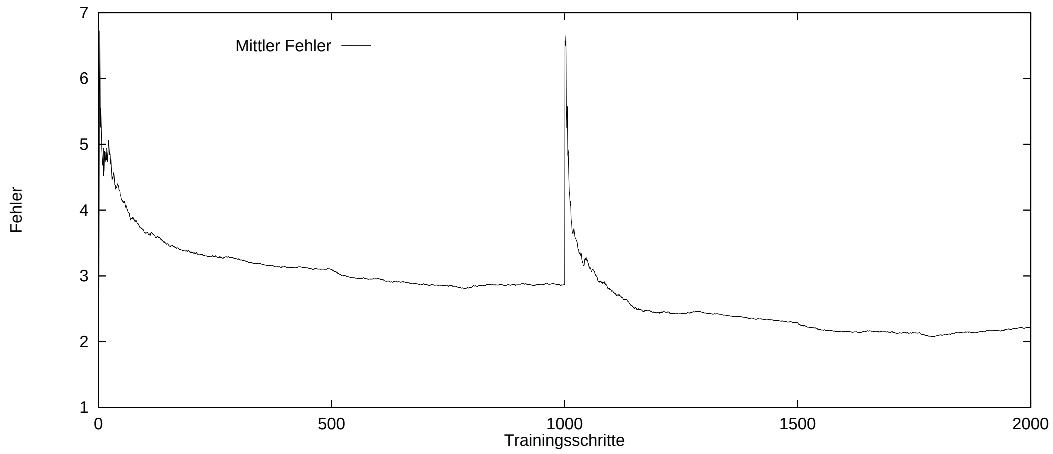


Abbildung 9.5: Fehlerverlauf beim Training eines Kohonennetzes unter Realweltbedingungen.

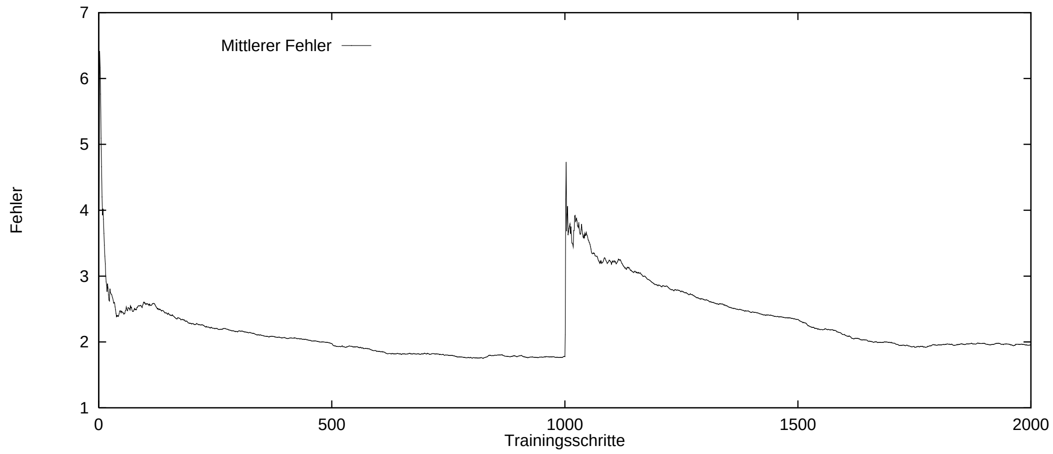


Abbildung 9.6: Fehlerverlauf beim Training einer Dynamischen Zell-Struktur unter Realweltbedingungen.

zwischen den Abweichungen der beiden X-Koordinaten und einer Y-Koordinate (Abschnitt 5.2).

Die erreichten Genauigkeiten von 2.0 bzw. 2.2 Pixel liegen wieder unter der vortrainierten Genauigkeit von 2.5 Pixel. Der mit klassischen Kamerakalibrierung erreichte Wert von 3.2 Pixel Genauigkeit wurde auch hier unterboten.

9.3 Rechenzeit

Das Ziel eines technischen Systems muß es sein, seine Aufgabe neben der Exaktheit und Zuverlässigkeit auch möglichst schnell auszuführen. Im Fall der be-

schriebenen Sakkadensteuerung ist dies von besonderer Bedeutung, da sich in der Praxis an die Zentrierung eines Szenenpunktes eine Auswertung der neuen Kamerabilder, im Hinblick auf eine gewünschte Aufgabe hin, anschließen wird. Damit für diese Auswertung möglichst viel Zeit bleibt, muß die Zeit der Ausführung der Sakkade minimiert werden. Diese begrenzt sich einerseits durch die Stellgeschwindigkeit des Kamerakopfes und andererseits durch die Rechenzeit des Algorithmus zur Berechnung der Sakkade.
Die Zeiten der Berechnung der Sakkade und der Adaption sind in Abbildung 9.7 für den binokularen Fall dargestellt. Die Angaben beziehen sich auf die gesamt benötigte Rechenzeit und nicht nur auf die CPU-Zeit.
Die Rechenzeit mit DCS ist kürzer als mit Kohonenkarten. Dies geht besonders auf die Berechnung und Adaption nur mit den benachbarten Neuronen des am besten matchenden Neurons zurück, aber auch auf die geringere Neuronenanzahl bei DCS (bei gleicher Genauigkeit der Sakkaden), die die Zeit zur Berechnung des am besten matchenden Neurons bestimmt. Bei Kohonenkarten wird jedesmal das gesamte Netzwerk zur Berechnung der Ausgabe genutzt und auch adaptiert.

	Berechnen [sec]	Adaptieren [sec]
Kohonenkarten (234 Neuronen)	0.009	0.011
DCS (124 Neuronen)	0.003	0.002

Abbildung 9.7: Rechenzeiten im binokularen Fall mit neuronalen Netzen. Es sind die Dauer der Sakkadenberechnung und der Adaption dargestellt. Die Berechnungen beziehen sich auf eine SUN SparcStation 10.

Kapitel 10

Diskussion und Ausblick

Den Abschluß dieser Arbeit bildet eine Zusammenfassung der vorgestellten Methoden und Ergebnisse. Diese werden bewertet, und es wird ein Ausblick auf Erweiterungen in zukünftigen Arbeiten gegeben.

10.1 Diskussion

In dieser Arbeit wurden monokulare und binokulare Sakkaden auf einem binokularen Kamerakopf implementiert. Es wurde die konventionelle Methode mit neuronalen Ansätzen verglichen.
Die konventionelle Methode basiert auf einem mathematischen Modell des Kamerakopfes. Anhand dieses Modells wurde die Inverse Kinematik berechnet. Der Nachteil dieses Verfahrens liegt darin, daß sich mathematisches Modell und Realität nicht exakt decken. Außerdem ist eine sorgfältige Kalibrierung der Kameras, d.h. die Bestimmung der intrinsischen Kameraparameter, notwendig, um genaue Ergebnisse zu erhalten.

Die neuronalen Ansätze zur Implementation von Sakkaden basieren auf einem iterativen Lernalgorithmus. Sie benötigen keine Annahmen über die realen Kameras und kommen ohne Kalibrierung aus.
Duch die Verwendung eines iterativen Lernverfahrens kann das System jederzeit auf Veränderungen reagieren, da Trainings- und Arbeitsphase nicht entkoppelt sein müssen.
Es wurden Sakkaden mittels Kohonenkarten und Dynamischen Zell-Strukturen als neuronale Basis implementiert. Kohonenkarten besitzen bekannte Nachteile, zum einen ist dies die fest vorzugebende Topologie. Zum anderen wäre eine Verteilung der Neuronen entsprechend des Approximationsfehlers wünschenswert. Kohonenkarten gehorchen jedoch immer der Wahrscheinlichkeitsverteilung der

Eingaben.
Eine Neuronenverteilung entsprechend des Approximationsfehlers wird von P. Dean und J. Mayhew in ihrer Arbeit über die Implementation von neuronalen Sakkaden als Weiterentwicklung gefordert [May91]. Genau diese Eigenschaft besitzen die Dynamischen Zell-Strukturen (DCS). Duch ein inkrementelles Wachstum entsprechend des Fehlers und den Aufbau einer perfekten Topologie entwickelt sich das Netzwerk. Ist der Fehler unter eine Toleranzgrenze abgesunken, wird das Training beendet, man erhält ein minimal kleines neuronales Netz. Der selbständige Aufbau der lateralen Neuronenverbindungen basiert auf einer nichtlinearen Hebb'schen Lernregel und basiert auf Arbeiten von T. Martinetz [Mar94].
Der Nachteil der DCS gegenüber Kohonenkarten ist die weit höhere Trainingszeit, die durch das inkrementelle Wachstum bedingt ist. Dafür zeigen die Versuche in Kapitel 8 deutlich, daß die gleiche Genauigkeit der Sakkaden unter Verwendung von DCS mit deutlich weniger Neuronen erreicht wird.
Die Neuronenverteilung zeigt dabei eine ganz eindeutige Struktur. Im monokularen Fall sind mehr Neuronen bei hohen als bei kleinen Vergenzwinkeln zu finden. Im binokularen Fall erhält man die gleiche Verteilung auf den Neigungswinkel bezogen.
Diese Ergebnisse werden gestützt durch die Analyse der Inversen Kinematik in Abschnitt 3.5. Anhand des mathematischen Modells wurde hier aufgezeigt, daß die zu approximierende Inverse-Kinematik-Funktion für kleine Vergenz- bzw. Neigungswinkel näherungsweise linear ist, während sie für große Winkel stark nichtlinear wird.
Die Betrachtung der Neuronenverteilungen zeigt aber auch bei DCS das Fehlen von Neuronen direkt an den Grenzen der Winkelbereiche, obwohl diese hier besonders wichtig wären. Unter Verwendung der einfachen Kohonenregel ist dies deutlich zu erkennen, die Gradient-Modulation zeigt hier eine eindeutige Verbesserung, die sich auch in der Trainingszeiten und den benötigten Neuronen bestätigt.

Das Training basiert auf der Methode des Error-Feedback-Lernens, welche von M. Kawato vorgeschlagen wurde [Kaw90a]. Es wurde ein einfacher Proportional-Regler implementiert, mit dem gute Ergebnisse erzielt werden. Die in Abschnitt 5.3 gezeigte Verbesserung der Genauigkeit durch den P-Regler wird in Abschnitt 8.1.4 bestätigt. Im Off-Line Training sind die mit P-Regler im Vergleich zur Vorgabe des perfekten Winkeloffsets gemachten Ergebnisse gleichwertig.

Die langen Trainingszeiten machen ein On-Line Training ohne Vorkenntnisse sehr langwierig. Die neuronalen Netze werden deshalb Off-Line auf das Kameramodell vortrainiert und können On-Line in kurzer Zeit angepaßt werden. Die guten Ergebnisse des Off-Line Trainings finden sich auch unter Realweltbedingungen bestätigt (Kapitel 9).

10.2 Ausblick

Die Aufgabe, Sakkaden mittels neuronaler Netze zu erlernen, ist ausführlich bearbeitet worden. Die von J. Mayhew und P. Dean in ihrem Ausblick angesprochenen Erweiterungen zur Implementation neuronaler Sakkaden ([May91]) wurden mit Dynamischen Zell-Strukturen realisiert. Zukünftige Erweiterungen ergeben sich damit nur in der Organisation des Trainings. Ein Lernen auf natürlichen Szenen und schließlich die Integration der Sakkade in ein System aus verschiedenen Augenbewegungen wäre wünschenswert.

Augenscheinlich ist die Methode, einen Lichtpunkt als Stimulus für die Sakkade zu verwenden, sehr simpel.
Kombiniert man einen Attentionsmechanismus mit der Möglichkeit, das ausgewählte Szenemerkmal nach der Sakkade exakt lokalisieren zu können, kann das Training unter Realweltbedingungen ohne Roboter auskommen und - in einem Raum mit genügend geeigneten Merkmalen - der Kamerakopf ein autonomes System darstellen.
Schließlich ist das oberste Ziel die Integration der Sakkade in ein System, das mehrere Augenbewegungen verwirklicht, wie es vom Menschen bekannt ist (siehe Abschnitt 1.1). Entsprechend der Umweltbedingungen und Reize kann sich die Kamerabewegung aus Sakkaden, Folge- und Vergenzbewegungen zusammensetzen. Bedingung für ein solches Zusammenspiel sind allerdings auch geeignete Attentionsmechanismen und eine Entscheidungsebene, in der der Bewegungsmodus der jeweiligen Situation entsprechend ausgewählt wird.

Anhang A

Inverse Kinematik

Im Fall binokularer Sakkaden ist der erste Schritt zur Berechnung der Sakkadenwinkel mittels Inverser Kinematik die Berechnung des Weltpunktes, der in den Kamerabildern zentriert werden soll. Das betrachtete Kameramodell ist in Kapitel 2 vorgestellt worden. Die Projektion des Weltpunktes $\vec{p}_w$ in die Bildebenen der Kameras ergibt zwei Bildpunkte. Um aus diesen und den Kamerawinkeln den Weltpunkt zu rekonstruieren, werden zuerst die beiden Sehstrahlen berechnet. Der Weltpunkt ergibt sich dann als Schnittpunkt der Sehstrahlen. Dieses Ergebnis wird in Abschnitt 3.2.1 genutzt um die neuen Kamerawinkel zu berechnen.

Ist ein Parametersatz $(\chi, \phi, \theta_l, \theta_r, x_{i_l}, y_{i_l}, x_{i_r}, y_{i_r})$ der Abbildung eines Szenenpunktes $\vec{p}_w$ in die Bildebenen gegeben, so schneiden sich die Sehstrahlen für linke und rechte Kamera in $\vec{p}_w$:

$$\vec{p}_{w_l} = \lambda_l T_\chi T_\phi T_{\theta_l} \vec{p}_{c_l} \quad ; \quad \lambda_l \in R$$

$$\vec{p}_{w_r} = \lambda_r T_\chi T_\phi T_{\theta_r} \vec{p}_{c_r} \quad ; \quad \lambda_r \in R$$

Durch Definition von Konstanten $c_1, c_2, .., c_{10}$ läßt sich die Gleichung

$$T_\chi T_\phi T_{\theta_r} \begin{pmatrix} \lambda_r x_{c_r} \\ \lambda_r y_{c_r} \\ \lambda_r \\ 1 \end{pmatrix} = T_\chi T_\phi T_{\theta_l} \begin{pmatrix} \lambda_l x_{c_l} \\ \lambda_l y_{c_l} \\ \lambda_l \\ 1 \end{pmatrix} = \vec{p}_w$$

umformen in:

$$\begin{pmatrix} \lambda_r c_1 + c_2 \\ \lambda_r c_3 \\ \lambda_r c_4 + c_5 \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda_l c_6 + c_7 \\ \lambda_l c_8 \\ \lambda_l c_9 + c_{10} \\ 1 \end{pmatrix} \tag{A.1}$$

Dabei sind:

$$\begin{aligned}
c_1 &:= x_{c_r}\cos\chi\,\cos\theta_r\, - x_{c_r}\sin\chi\,\cos\phi\,\sin\theta_r \\
&\quad -y_{c_r}\sin\chi\,\sin\phi\, -\,\cos\chi\,\sin\theta_r \\
&\quad -\sin\chi\,\cos\phi\,\cos\theta_r \\
c_2 &:= 0.5\,b\cos\chi \\
c_3 &:= -x_{c_r}\sin\phi\,\sin\theta_r\, + y_{c_r}\cos\phi - \sin\phi\,\cos\theta_r \\
c_4 &:= x_{c_r}\sin\chi\,\cos\theta_r\, + x_{c_r}\cos\chi\,\cos\phi\,\sin\theta_r \\
&\quad +y_{c_r}\cos\chi\,\sin\phi\, -\,\sin\chi\,\sin\theta_r \\
&\quad +\cos\chi\,\cos\phi\,\cos\theta_r \\
c_5 &:= 0.5\,b\sin\chi \\
c_6 &:= x_{c_l}\cos\chi\,\cos\theta_l\, + x_{c_l}\sin\chi\,\cos\phi\,\sin\theta_l \\
&\quad -y_{c_l}\sin\chi\,\sin\phi\, +\,\cos\chi\,\sin\theta_l \\
&\quad -\sin\chi\,\cos\phi\,\cos\theta_l \\
c_7 &:= -0.5\,b\cos\chi \\
c_8 &:= x_{c_l}\sin\phi\,\sin\theta_l\, + y_{c_l}\cos\phi - \sin\phi\,\cos\theta_l \\
c_9 &:= x_{c_l}\sin\chi\,\cos\theta_l\, - x_{c_l}\cos\chi\,\cos\phi\,\sin\theta_l \\
&\quad +y_{c_l}\cos\chi\,\sin\phi\, +\,\sin\chi\,\sin\theta_l \\
&\quad +\cos\chi\,\cos\phi\,\cos\theta_l \\
c_{10} &:= -0.5\,b\sin\chi
\end{aligned}$$

Wir bestimmen λ_r aus der ersten und dritten Zeile der Gleichung. Die zweite Zeile ist genau dann erfüllt, wenn die Bildkoordinaten der Eingabe vom gleichen Weltpunkt stammen. Anderenfalls können sich die beiden Sehstrahlen nicht in einem Weltpunkt schneiden.

$$\lambda_r := \frac{-c_2\,c_9 + c_6\,c_5 - c_6\,c_{10} + c_7\,c_9}{c_1\,c_9 - c_4\,c_6}$$

Aus der Gleichung

$$\begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda_r\,c_1 + c_2 \\ \lambda_r\,c_3 \\ \lambda_r\,c_4 + c_5 \\ 1 \end{pmatrix}$$

erhält man den Weltpunkt.

Anhang B

Unabhängigkeiten

Betrachtet man die Projektion eines Szenepunktes in die Bildebene einer Kamera, so stellt sich die Frage, wohin dieser Bildpunkt bei einer Veränderung der Kamerawinkel im Bild verschoben wird, und von welchen Parametern diese Verschiebung abhängt. Im folgenden werden zuerst monokulare Sakkaden betrachtet. Im binokularen Fall ergeben sich die Beweise jeweils auf analogem Weg. Dabei wird gezeigt, daß die Berechnung des korrekten Winkeloffsets im monokularen Fall vom momentanen Neigungswinkel unabhängig ist. Im binokularen Fall kann man die Unabhängigkeit vom Schwenkwinkel zeigen.
Daran anschließend wird gezeigt, daß allgemein für die Berechnung der neuen Bildkoordinaten eines Szenepunktes nach einer Veränderung der Kamerawinkel um einen Offset $(\Delta\phi, \Delta\theta_{l/r})$ die Kenntnis des momentanen Neigungswinkels nicht notwendig ist.

B.1 Monokulare Sakkaden

Berechnet man die Sakkaden anhand des in Kapitel 3 beschriebenen Kamerasystems als Offset auf die momentanen Kamerawinkel, so stellt sich die Frage, welche Eingabeparameter nötig sind, um diese Aufgabe korrekt zu lösen. Es wird im folgenden gezeigt, daß im monokularen Fall der momentane Neigungswinkel nicht notwendig ist, um den Winkeloffset zu berechnen.

Die Abhängigkeiten werden nur für die linke Kamera berechnet, für die rechte folgen sie analog.

Dazu untersuchen wir der Einfachheit halber nicht die Differenzen $\phi_l^* - \phi_l$ bzw. $\theta_l^* - \theta_l$ direkt, sondern deren Tangens:

$$\tan(\phi_l^* - \phi_l) = \tan\Big(\arctan(-y_{w_l}) - \phi_l\Big) = \frac{-y_{w_l} - \tan\phi}{1 - y_{w_l}\tan\phi}$$

$$= \frac{y_{c_l}}{x_{c_l} \sin \theta_l \; - \cos \theta_l}$$

Sei $\Delta\phi := \phi^* - \phi$. Wie man an der obigen Formel erkennt, ist $\phi_l^* - \phi_l$ abhängig von θ_l und den Kamerakoordinaten x_{c_l} und y_{c_l}. Sie ist unabhängig von ϕ. Folglich ist $\Delta\phi$ unabhängig von ϕ.

$$\tan(\theta_l^* - \theta_l)$$

$$= \tan\left(\arctan\left(\frac{x_{w_l}}{-y_{w_l} \sin(\phi + \Delta\phi) \; + \cos(\phi + \Delta\phi)}\right) - \theta_l\right)$$

$$= \frac{\dfrac{x_w}{\cos\phi \, \cos\Delta\phi - \sin\phi \, \sin\Delta\phi - (\sin\phi \, \cos\Delta\phi + \cos\phi \, \sin\Delta\phi) \; y_w} - \tan\theta_l}{1 + \dfrac{x_w \tan\theta_l}{\cos\phi \, \cos\Delta\phi - \sin\phi \, \sin\Delta\phi - (\sin\phi \, \cos\Delta\phi + \cos\phi \, \sin\Delta\phi) \; y_w}}$$

$$= \frac{\left[\begin{array}{l} x_{c_l} \cos\Delta\phi \, \cos^2\theta_l \; - \cos\theta_l \, \sin\theta_l \; - y_{c_l} \sin\theta_l \, \sin\Delta\phi \; - x_{c_l} \cos\Delta\phi - \\ x_{c_l} \cos^2\theta_l \; + \sin\theta_l \, \cos\Delta\phi \, \cos\theta_l \end{array}\right]}{\left[\begin{array}{l} y_{c_l} \cos\theta_l \, \sin\Delta\phi \; + x_{c_l} \cos\theta_l \, \cos\Delta\phi \, \sin\theta_l \; - \cos\Delta\phi \, \cos^2\theta_l \; - 1 - \\ x_{c_l} \sin\theta_l \, \cos\theta_l \; + \cos^2\theta_l \end{array}\right]}$$

Es ergeben sich damit folgende Abhängigkeiten: $\Delta\theta_l = \theta_l^* - \theta_l$ ist abhängig von $\theta_l, \Delta\phi$ und den Kamerakoordinaten x_{c_l} und y_{c_l} und unabhängig von ϕ.

B.2 Allgemeiner monokularer Fall

Betrachtet man die Projektion eines Weltpunktes $\vec{p}_w$ in die Bildebene auf Koordinaten p, so können die Bildkoordinaten p' nach einer Winkeländerung um $(\Delta\phi, \, \Delta\theta_{l/r})$ unabhängig vom Neigungswinkel ϕ berechnet werden. Der Fall der monokularen Sakkaden, bei dem der Winkeloffset $(\Delta\phi, \, \Delta\theta_{l/r})$ zur Zentrierung des Weltpunktes $\vec{p}_w$ im Kamerabild führt, ist ein Spezialfall dieser Eigenschaft.

Beweis:
Seien Kamerawinkel $(\phi, \, \theta)$, ein Sehstrahl $\vec{p}_w$ und ein Winkeloffset $(\Delta\phi, \Delta\theta)$ gegeben. Der Weltpunkt $\vec{p}_w$ werde auf die Bildkoordinaten (x_i, y_i) abgebildet.
Für einen beliebigen Winkel ϕ' sei der Weltpunkt $\vec{p'}_w$ so gewählt, daß die Abbildung in Bildkoordinaten wieder (x_i, y_i) ergibt.

Dann sind die neuen Bildkoordinaten entsprechend $\vec{p}_w$ und $\vec{p'}_w$, die sich für die Kamerawinkel $\phi + \Delta\phi$, $\phi' + \Delta\phi$ und $\theta + \Delta\theta$ ergeben, gleich:

$$\vec{p}_{c_1} = T^{-1}_{(\theta+\Delta\theta)} T^{-1}_{(\phi+\Delta\phi)} \vec{p}_w$$

$$\vec{p}_{i_1} = P\left(\frac{\vec{p}_{c_1}}{z_{c_1}}\right)$$

$$\vec{p}_{c_2} = T^{-1}_{(\theta+\Delta\theta)} T^{-1}_{(\phi'+\Delta\phi)} \vec{p'}_w$$

$$\vec{p}_{i_2} = P\left(\frac{\vec{p'}_{c_2}}{z_{c_2}}\right)$$

Durch Nachrechnen erhält man:

$$\vec{p}_{i_1} = \vec{p}_{i_2}.$$

Für die beiden Situationen aus Kamerawinkeln und Bildkoordinaten (ϕ, θ, x_i, y_i) und $(\phi', \theta, x_i, y_i)$ wird der Weltpunkt $\vec{p}_w$ bzw. $\vec{p'}_w$ nach Veränderung der Kamerawinkel um $(\Delta\phi, \Delta\theta)$ wieder auf die gleichen Bildkoordinaten (x_i', y_i') projiziert $(\vec{p}_{i_1} = \vec{p}_{i_2})$.
Die Berechnung der neuen Bildkoordinaten ist damit unabhängig vom Neigungswinkel ϕ.

Abbildungsverzeichnis

Symbolverzeichnis

χ	Schwenkwinkel	12
ϕ	Neigungswinkel	13
$\theta_{l/r}$	Linker bzw. Rechter Vergenzwinkel	13
b	Stereobasis	13
$\vec{p}_w$	Punkt im Weltkoordinatensystem	13
$\vec{p}_c$	Punkt im Kamerakoordinatensystem	14
$\vec{p}_i$	Punkt im Bildkoordinatensystem	14
$T_\chi, T_\phi, T_{\theta_l}, T_{\theta_r}$	Rotationsmatrizen	14
$\alpha_{xc}, \alpha_{yc}, x_{0c}, y_{0c}$	Intrinsische Kameraparameter	15
P_c	Matrix der intrinsischen Kameraparameter	15
Ω	Eingaberaum der Kamerawinkel	47
Ω'	Ausgaberaum der Kamerawinkeloffsets	47
ω	Kamerawinkel vor der Sakkade	17
ω^*	Kamerawinkel der perfekten Sakkade	17
ω'	Mit neuronalem Netz berechneter Kamerawinkeloffset	41
ω_P	Mit P-Regler verbesserter Kamerawinkeloffset	41
P	Raum der Bildkoordinaten	47
p	Bildkoordinaten vor der Sakkade	17
p'	Bildkoordinaten nach der Sakkade	41
k_1, k_2, l_1, l_2, l_3	Konstanten des P-Reglers	41
S	Menge der Szenenkoordinaten (Kalibrierwürfel)	33
K	Menge der Weltkoordinaten (Kalibrierwürfel)	33
$u = (\omega, p)$	Eingabe des neuronalen Netzes	47
W	Menge der Neuronen des neuronalen Netzes	47
W	Anzahl der Neuronen des neuronalen Netzes	47
w_r	Zentrumsvektor des Neurons $r \in W$	47
A_r	Matrix des Neurons $r \in W$	47
a_r	Aktivierung des Neurons $r \in W$	47
r_r	Resource des Neurons $r \in W$	54
C	Adjazenzmatrix	54
V_i	Voronoi-Polygon des Neurons $r \in W$	56
BMU	Die Eingabe u am besten matchendes Neuron	48

$SECOND$	Die Eingabe u am zweitbesten matchendes Neuron	48
$\bar{A}$	Ausgabematrix des neuronalen Netzes	48
A^*	Verbesserter Schätzwert der Matrix $\bar{A}$	48
ϵ, ϵ'	Lernschrittweiten	48
h_{rs}, h'_{rs}	Nachbarschaftsfunktionen	59
g_r	Gitterort des Neurons $r \in W$	50
β	Gleitwertkonstante der Resource-Berechnung	55
η	Lernschrittweite	55
s_r^{∇}	Modulationskoeffizient der Adaption	59
$Nh(r)$	Nachbarschaft des Neurons $r \in W$	48
$Outdeg(r)$	Anzahl der Nachbarn des Neurons $r \in W$	59
σ	Varianz der radialen Basisfunktionen	55
τ	Alterungskonstante der Neuronenverbindungen	58
μ	Untere Grenze für eine Neuronenverbindung	58

Literaturverzeichnis

[Baj88] R. Bajcsy. *Active Perception.* Proceedings of the IEEE, Vol. 76, No. 8, pp. 996-1005, 1988.

[Bru95] J. Bruske, G. Sommer. *Dynamic Cell Structure Learns Perfectly Topology Preserving Map.* Neural Computation 7, pp. 845-865, 1995.

[Car88] R. Carpenter. *Movement of the eyes.* 2nd ed. Pion, London, 1988.

[DA95] K. Daniilidis, M. Hansen, C. Krauss, G. Sommer. *Auf dem Weg zum künstlichen aktiven Sehen: Modellfreie Bewegungsverfolgung durch Kameranachführung.* Mustererkennung 1995, Tagungsband 17. DAGM–Symposium, Bielefeld, Springer Verlag, pp. 277–284, 1995.

[Du93] F. Du and M. Brady. *Self-calibration of the intrinsic parameters of cameras for active vision systems.* CVPR, pp. 477-482, 1993.

[Fau93] O. Faugeras. *Three-Dimensional Computer Vision.* MIT Press, 1993.

[Fri93a] B. Fritzke. *Vector quantisation with a growing and splitting elastic net.* Proc. of ICANN 93, pp. 580-585, 1993.

[Fri93b] B. Fritzke. *Growing Cell Structures - a self organizing network for unsupervised and supervised training.* ICSI Berkeley, Technical Report, tr-93-026. Also in: Neural Networks, Vol. 7, No. 9, same title, pp. 1441-1460, 1995.

[Gro85] R. Groner, G. McConkie, C. Menz. *Eye Movements and Human Information Processing.* North-Holland, 1985.

[Hub89] D. H. Hubel. *Auge und Gehirn: Neurobiologie des Sehens.* Spektrum-der-Wissenschaft Verlagsgesellschaft, Heidelberg, 1989.

[Hus93] D. Hush, B. Horne. *Progress in Supervised Neural Networks.* IEEE Signal Processing Magazine 10, pp. 8-39, January 1993.

[Kaw90a] M. Kawato. *Feedback-error learning neural network for supervised motor learning.* Eckmiller R. (ed) Advanced Neural Computers, Elsevier, Amsterdam, pp. 365-372, 1990.

[Kaw90b] M. Kawato. *Computational Schemes and Neural Network Models for Formation and Control of Multijoint Arm Trajectory.* Neural Networks for Control, Eds. W. T. Miller, R. S. Sutton und P. J. Werbos, MIT Press, pp. 197-228, 1990.

[Koh90] T. Kohonen. *The self-organizing map.* Proceedings of the IEEE, 78(9), pp. 1464-1480, 1990.

[Koh95] T. Kohonen. *Self-Organizing Maps.* Springer Verlag, Berlin Heidelberg, 1995.

[Li94] M. Li. *Camera calibration of a head-eye system for active vision.* Lecture Science in Computer Vision, ECCV3, pp. 543-554, ECCV94, 1994.

[Mar92] T. Martinetz. *Selbstorganisierende neuronale Netzwerke zur Bewegungssteuerung.* Dissertation, Infix-Verlag, DISKI, 1992.

[Mar93] T. Martinetz. *Competetive Hebbian Learning Rule Forms Perfectly Topology Preserving Maps.* Proc. of ICANN 93, pp. 426-438, 1993.

[Mar94] T. Martinetz, K. Schulten. *Topology Representing Networks.* Neural Networks, No.7, Vol. 3, pp. 505-522, 1994.

[May91] J. E. W. Mayhew, P. Dean. *Saccade control in a simulated robot camera-head system: Neural net architectures for efficient learning of inverse kinematics.* Biological Cybernetics, Springer Verlag, pp. 27-36, 1991.

[May92] J. E. W. Mayhew. *The adaptive control of a four degree-of-freedom stereo camera head.* Phil. Trans. Royal Society London B , 337, pp. 315-326, 1992.

[Mur92] D. W. Murray. *Design of Stereo Heads.* In A. Blake und A. Yuille, editors, active vision, pp. 155-172, MIT Press, 1992.

[Pla91a] J. Platt. *A Resource-Allocating Network for Function Interpolation.* Neural Computation, Vol.2, pp. 213-225, 1991.

[Pla91b] J. Platt. *Learning by Combining Memoralization and Gradient Descent.* NIPS'91, pp. 714-721, 1991.

[Rit91] H. Ritter, T. Martinetz, K. Schulten. *Neuronale Netze.* Addison-Wesley, 1991.

[Spa87] D. L. Sparks, J. S. Nelson. *Sensory and motor maps in the mammalian superior colliculus.* TINS 10, pp. 312-317, 1987.

[Spe86] *Wahrnehmung und visuelles System.* Spektrum der Wissenschaft, Heidelberg, 1986.

[Ste95] G. Stein. *Accurate internal camera calibration using rotation, with analysis o sources of errors.* In International Conference on Computer Vision, pp. 230–236, 1995.

[Töl92] S. Tölg. *Strukturuntersuchungen zur Informationsverarbeitung in neuronaler Architektur am Beispiel der Modellierung von Augenbewegungen für aktives Sehen.* VDI-Verlag GmbH, 1992.

[Tsa86] R.Y. Tsai. *An efficient and accurate camera calibration technique for 3d machine vision.* CVPR86, pp. 364-374, 1986.

[Tsa89] R. Tsai. *Synopsis of recent progress on camera calibration for 3d machine vision.* In O. Khatib et al., editor, The Robotics Review, pp. 147–159, The MIT Press, Cambridge, MA, 1989.

[Wid60] B. Widrow, M. E. Hoff. *Adaptive switching circuits.* WESCON Conv Record, Part IV, pp. 96-104, 1960.

[Yar67] A. L. Yarbus. *Eye Movements and Vision.* Plenum Press, New York, 1967.

Erklärung:

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

Kiel, den