Evolution von Bildverarbeitungs-schichten

Master Thesis
zur Erlangung des Hochschulgrades
Master of Science in Cognitive Science

Institut für Kognitionswissenschaft
Universität Osnabrück

Erstgutachter: Prof. Dr. Martin Riedmiller
Zweitgutachterin: Dr. Barbara Hammer

April 2004

Sascha Lange
Sascha.Lange@uos.de

Ich danke ...

... meinen Betreuern Herrn Prof. Dr. Martin Riedmiller und Frau Dr. Barbara Hammer für die Betreuung dieser Arbeit und dafür, dass sie auch sonst immer ein offenes Ohr für mich hatten.

... Roland Hafner für seine Hilfe bei der Durchführung der Roboterversuche und für die Zeit, die er sich hierzu genommen hat.

... der Neuroinformatik Gruppe für die bereitgestellte Hard- und Software.

... Christiane und meinen Eltern, die mich während der gesammten Zeit ertragen und meine Arbeit am Ende auch noch korrekturlesen mußten...

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In der Robotik stellt die Interaktion des Roboters mit seiner Umwelt ein zentrales Problem dar. Mit den Sensoren erfasste Reize müssen schnell verarbeitet und die korrekten Handlungen verzögerungsfrei ausgewählt werden, um rechtzeitig auf Gefahren und andere zeitkritische Vorgänge reagieren zu können. Dabei spielen kamerabasiertes Handeln und die Reaktion auf visuelle Reize eine immer größere Rolle. Soll das Verhalten eines solchen Roboters allerdings mit Techniken des Maschinellen Lernens gelernt werden, treten bei der Verwendung von rohen Bilddaten einige Probleme auf. Die hohe Dimensionalität und die damit verbundene hohe Redundanz von Bilddaten (oftmals mehrere zehntausend Pixel) verhindern in den meisten Fällen, dass sie erfolgreich als direkte Eingabe für einen allgemeinen Lernalgorithmus dienen können. Außerdem ist die zu bewältigende Datenmenge in der Regel zu groß, als dass sie in Echtzeit von einem gewöhnlichen Lernalgorithmus bewältigt werden könnte (man stelle sich zum Beispiel ein neuronales Netz mit 640×320 Eingabeneuronen, einer angemessenen versteckten Schicht von zum Beispiel 100 Neuronen und vorhandenen Shortcutverbindungen vor).

Nichtsdestotrotz gibt es Ansätze, z.B. neuronale Netze direkt mit vollständigen Bildmatrizen zu füttern. Ein Beispiel hierfür ist das an der Carnegie Mellon University entwickelte ALVINN (siehe auch Abschnitt 2.2). Hier werden die in der Auflösung auf 30×32 Pixel stark reduzierten Bilder von einer an einem fahrenden Auto angebrachten Kamera direkt verwendet, um ein neuronales Netz auf die Generierung eines angemessenen Winkels für den Lenkradeinschlag zu trainieren. Das trickreiche Vorgehen [31] beim Trainieren des Netzes führte zum teilweisen Erfolg. Auf gut ausgebauten und markierten Fahrbahnen können die Fahrstreifen erkannt und die Spur bei beeindruckenden Geschwindigkeiten gehalten werden. In anderen Situationen wurde das neuronale Netz in der Trainingsphase aber auch auf falsche Merkmale überspezialisiert und folgte fälschlich dem Verlauf einer Pfütze oder eines Straßengrabens.

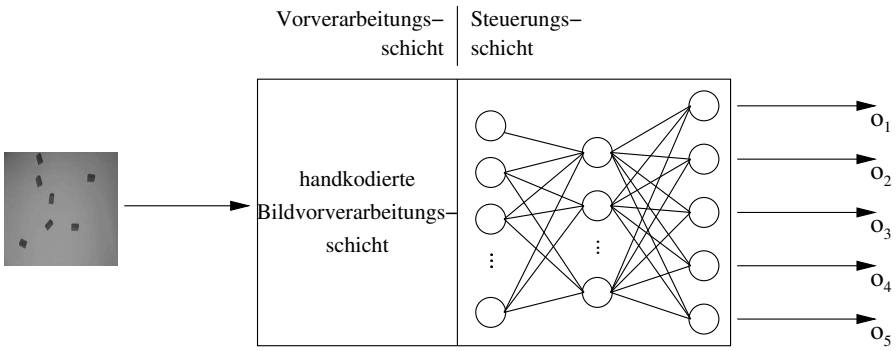

Abbildung 1.1: Zwei-Schichten-Architektur mit getrennten Bildverarbeitungs- und Kontrollschichten. In der Regel wird nur die Kontrollschicht gelernt.

Allgemein ist die Wahrscheinlichkeit aufgrund der hohen Dimension der Bilder hoch, dass sich in einer gegebenen Menge von Bildern einzelne, wenige Pixel finden lassen, deren Farbwerte eine korrekte Trennung der Bilder ermöglichen. Durch dieses einfache "Memorieren" der Bilder würden die der Entscheidung tatsächlich zugrundeliegenden Regularitäten und gemeinsamen Strukturen in den meisten Fällen unbeachtet bleiben. Versucht man anschließend, das durch das Memorieren erlernte Verhalten auf neue Situationen (Bilder) zu übertragen, bleibt der Erfolg[1] sehr wahrscheinlich aus.

Oft findet man daher Lösungen, die die Bilder vor der Verwendung als Eingabe für einen Lernalgorithmus vorverarbeiten. In der Regel werden hierbei Techniken des maschinellen Sehens zur Extraktion eines sogenannten Featurevektors, der prominente und wichtige Bildmerkmale mit einer deutlich reduzierten Dimensionalität kodiert, verwendet. Redundante und unwichtige in den Bildern enthaltene Informationen werden einfach entfernt, bzw. bleiben unberücksichtigt. Welche Informationen von Bedeutung sind und daher zur Weiterverarbeitung extrahiert werden, ist natürlich stark subjektiv und von der Aufgabe abhängig.

In Abbildung 1.1 ist die grobe Architektur eines diesem Paradigma folgenden Agenten dargestellt. Während die eigentliche Kontrollaufgabe durch die Vorverarbeitungsschicht ausreichend vereinfacht wurde und nun von einem Lernalgorithmus gelernt werden kann, wird das Bildverarbeitungssubsystem zumeist per Hand für die spezielle Aufgabe entwickelt. Die ursprüngliche Aufgabe wird also nicht mehr als ganzes betrachtet, sondern in zwei getrennte Teilaufgaben unterteilt, von denen nur die eine (einfachere) gelernt wird. Für den Entwickler bleibt weiterhin die Arbeit, für jede neue Aufgabe zur Berechnung der Lösung geeignete Bildmerkmale auszuwählen. Anschließend muss er feststellen, wie sich

[1] Mit einer erfolgreichen Übertragung des Trainingswissens (Generalisierung) ist hier gemeint, dass die durchschnittliche Abweichung der Ausgabe vom optimalen Steuersignal geringer als beim Raten ist.

die ausgewählten Merkmale von anderen Merkmalen unterscheiden und im Bild zweifelsfrei (robust) detektieren lassen. Nach der Definition eines geeigneten Extraktionsalgorithmus muss dieser noch implementiert werden.

Betrachtet man diesen Selektionsvorgang genauer, so wird deutlich, dass die Auswahl der aufgabenspezifischen Features zwar maßgeblich durch den Inhalt der Bilder (sensorische Reize), aber auch durch die Art der Aufgabe – also auch durch die optimalen Steuersignale – bestimmt wird. Zur Verdeutlichung sei hier ein konkretes Beispiel angeführt (siehe Abschnitt 4.1.2):

Besteht die Eingabemenge der Bilder aus Aufnahmen der Oberseite eines Würfels vor einem weißen Hintergrund und Aufnahmen des leeren Hintergrundes und ist die Aufgabe, zu bestimmen, ob sich der Würfel im Bild befindet, würde es ausreichen, nach einer quadratischen Fläche auf weißem Hintergrund zu suchen. Mit der Ausgabe eines einzelnen Wahrheitswertes könnte so ein "Würfel-da" Detektor realisiert werden. Würde die Aufgabe nun aber darin bestehen, den Würfel – falls im Sichtfeld der Kamera platziert – mittels eines Roboterarms zu greifen, könnte man weiterhin nach dem gleichen Feature "Quadratische-Fläche-auf-weißem-Hintergrund" suchen. Man müsste aber zusätzlich deren Mittelpunkt oder eine "Bounding Box" berechnen, um den Roboterarm an die richtige Stelle steuern zu können. Wenn es nun aber darum geht, die gezeigte Augenzahl von der Oberseite abzulesen, um zum Beispiel in einem Spielkasino automatisch ein Protokoll zu führen, dann würde die Suche nach dem Rechteck alleine nicht mehr ausreichen. Man müsste nun z.B. einen Algorithmus verwenden, der die würfelspezifischen Markierungen finden und verstehen kann. Die Suche nach der quadratischen Fläche, bzw. den Umrissen des Würfels könnte in einer solchen Aufgabenstellung völlig überflüssig werden.

Es ist ein leichtes, dieses Spiel weiter zu treiben und sich alleine durch die Modifikation der erforderlichen Ausgabe des Programms beliebig viele Aufgaben auszudenken, die alle die Extraktion unterschiedlicher Mengen von Merkmalen aus der gleichen Serie von Bildern erfordern.

Andererseits zeigt die Analyse aber auch, dass in einer solchen Aufgabenbeschreibung in Form von Beispielen für sensorische Eingaben und zugehörigen optimalen Steuersignalen eigentlich schon ausreichend viele Informationen vorhanden sind. Das Einbringen weiteren a priori Wissens durch den Entwickler sollte eigentlich unnötig sein, zumal auch ein Mensch alleine an Hand dieser Informationen die nötigen Verarbeitungsschritte und Merkmale herleiten könnte.

Die offene Fragestellung bleibt: Wie kann man aus einer gegebenen Menge von Bildern und optimalen Steuersignalen die hilfreichen Features ablesen und Hinweise auf die Konstruktion der Featureextraktoren gewinnen?

Genau mit dieser Frage beschäftigt sich die vorliegende Arbeit. Es soll ein praxistauglicher Algorithmus entwickelt werden, von dem anschließend bewiesen werden soll, dass er in der Lage ist, die oben spezifizierten "visuellen" Aufgaben als Ganzes zu lernen und robust zu lösen. Dabei sollen von dem Algorithmus hoch spezialisierte, aufgabenabhängige Vorverarbeitungsschichten erzeugt (Features ausgewählt) werden.

1.2 Überblick

In der vorliegenden Arbeit wird ein Algorithmus vorgestellt, der zwar auch die zweigeteilte Architektur aus getrennter Vorverarbeitungs- und Kontrollschicht verwendet, aber die Aufgabe als Ganzes betrachtet. Im Gegensatz zu den handkodierten Lösungen wird nicht nur das Kontrollsubsystem eingelernt, sondern auch die Vorverarbeitungsschicht automatisch während der Lernphase erzeugt und für jede Aufgabe speziell angepaßt.

Zu diesem Zweck wird ein hybrider Algorithmus entwickelt, der sowohl evolutionäre Techniken als auch überwachtes Lernen mittels neuronaler Netze anwendet. Die Menge der durch den Algorithmus lösbarer Lernprobleme ist derzeit auf Probleme überwachten Lernens beschränkt. Ein evolutionärer Algorithmus wird verwendet, um in einer äußeren Schleife Populationen von möglichen Lösungskandidaten zu erstellen und zu optimieren. Diese Kandidaten werden dann in einer inneren Schleife mittels lokalen Trainings auf den Trainingspattern des überwachten Lernproblems trainiert und an Hand des Fehlers hinsichtlich ihrer "Fitness" bewertet. Hierbei wird die Kontrollschicht des Kandidaten durch das lokal trainierte neuronale Netz und die Bildvorverarbeitungsschicht durch eine evolutionär bestimmte Auswahl von parametrisierten "Primitiven" realisiert.

Die Arbeit untergliedert sich in sechs Kapitel. Kapitel 1 ist diese kurze Einleitung. In Kapitel 2 wird ein ausführlicher Überblick über verwandte Ansätze und eine Diskussion der bisherigen Erfolge vorgenommen. Anschließend wird in Kapitel 3 der neue Algorithmus vorgestellt. In Kapitel 4 wird die Durchführung einiger Experimente beschrieben, die Ergebnisse werden ausgewertet und diskutiert. Abschließend wird in Kaptitel 5 noch einmal kurz zusammengefasst und noch offene Fragen werden aufgelistet.

Kapitel 2

Stand der Forschung

In diesem Abschnitt werden einige Arbeiten vorgestellt, die ähnliche Ansätze zum in dieser Arbeit vorgestellten Algorithmus verfolgen, als Grundlage für diese Arbeit dienen oder mögliche Einsatzgebiete aufzeigen. Es wird versucht, jede dieser Arbeiten in einen ganz konkreten Bezug zu der vorliegenden Arbeit zu bringen und Gemeinsamkeiten und Unterschiede der Grundideen herauszuarbeiten. Ziel ist es, die Arbeit klar von anderen Arbeiten abzugrenzen und dem Leser eine bessere Einordnung des vorgestellten Algorithmus in die aktuelle Forschung zu ermöglich.

2.1 Featureselektion und Reduzierung der Dimension mittels statistischer Methoden

Bei Bildern besteht das Problem, dass die Daten in der Regel zu hochdimensional und zu unstrukturiert für eine erfolgreiche Weiterverarbeitung sind. Aus der Statistik wurden daher etliche Methoden entliehen, die bei der Analyse und Strukturierung solcher Daten helfen können. Zu nennen sind hier vor allem die auf der Faktorennanalyse basierenden Verfahren wie Principle Component Analysis (PCA[1]) [20] und Independent Component Analysis (ICA) [18]. Bei PCA werden unkorrelierte Faktoren in den Daten gesucht, die unter der Annahme von normalverteilten Zufallsvariablen unabhängig voneinander sind. Bei ICA werden dagegen unabhängige Faktoren in nicht normalverteilten Daten gesucht. Normalerweise wird bei beiden Verfahren nicht die Dimension des Datenraumes reduziert, sondern die Vektoren werden nur in einen günstigeren Raum gleicher Dimensionalität transformiert (unkorrelierte, bzw. unabhängige Basisvektoren). Allerdings werden die "wichtigsten" Basisvektoren zuerst gefunden, so dass man die transformierten Vektoren im Idealfall ab einer bestimmten Stelle abschneiden kann, ohne zu viele Informationen zu verlieren. Es gibt bei beiden Methoden

[1] Es wird hier die englische, allgemein gebräuchlichere Abkürzung anstelle des deutschen Namens "Hauptkomponentenanalyse" verwendet.

auch Abarten, die die Auswahl einzelner, besonders wichtiger Dimensionen erlauben, ohne die ursprünglichen Basisvektoren transformieren zu müssen. Allerdings ist das Ergebnis schlechter und in vielen Fällen sogar unbrauchbar (z.B., wenn die aktuelle Basis weit entfernt von der Basis der Faktoren ist und alle Dimensionen hoch korreliert, bzw. voneinander abhängig sind) [10, 21].

PCA und ICA berücksichtigen normalerweise keine vorhandene Information über die Klassifizierung der vorhandenen Vektoren. Da die Klasseneinordnung aber hinsichtlich der Relevanz einzelner Vektoreinträge eine wichtige und hilfreiche Information darstellt, gibt es erweiterte Verfahren, die diese Information in einer Regressionsanalyse berücksichtigen können (siehe zum Beispiel [20, S. 167ff]).

Für das konkrete Problem, in Bildern gute Features zu finden, gibt es im Prinzip zwei unterschiedliche Ansatzpunkte für statistische Methoden. In einem naiven Ansatz würde man die Bilddaten selbst hinsichtlich relevanter Komponenten analysieren lassen. Das scheitert daran, dass die Basis des "Bildraumes" extrem groß ist (entspricht der Auflösung; bei typischen 640×480 Pixeln und 3 Farbwerten fast eine Million Basisvektoren), einzelne Pixel selten eine Bedeutung haben und fraglich ist, ob durch eine einfache Linearkombination die tatsächlichen Strukturen (z.B. im Fall von nicht linearen Abhängigkeiten) repräsentiert werden können (Bilder mit dem gleichen Objekt an verschiedenen Stellen hätten immer noch keine ähnlichen Repräsentierungen).

Erfolg versprechender sind da Abwandlungen, die eine deutlich kleinere Basis von größeren Bildbausteinen verwenden und den damit verbundenen Informationsverlust in Kauf nehmen. Erfolgreich sind statistische Ansätze, insbesondere wenn es um eine fest abgegrenzte Klasse von fest positionierten Objekten geht (z.B. im Bild zentrierte Gesichter, siehe "Eigenface" Ansatz [43]).

Ein gänzlich anderer Ansatz wäre, eine große Menge von Features zu berechnen, von denen man meint, dass zumindest einige allein oder in Kombination mit anderen hilfreich sein könnten. Dann kann man die statistischen Methoden einsetzen, um diejenigen Features herauszufiltern, die tatsächlich nützlich sind. Im Hinblick auf eine aufgabenspezifische Bildverarbeitungsschicht müssen also erstmal alle in Frage kommenden Werte und Features berechnet werden, um sie dann z.B. in den PCA Algorithmus füttern zu können. Und das Ganze nur, um im Nachhinein feststellen zu können, welche Werte nicht weiterhelfen und daher nicht berechnet werden müssen. Das Problem der Nichterfassbarkeit nicht linearer Abhängigkeiten zwischen den berechneten Features bliebe natürlich beim Einsatz linearer Methoden wie PCA und ICA auch weiterhin bestehen.

Auch im hier vorgestellten Ansatz muss prinzipiell jedes mögliche Feature einmal berechnet werden, um die besten herauszufinden. Wenn es allerdings darum geht, die beste Kombination zu finden, stossen PCA und ICA alleine schon aus praktischen Gesichtspunkten wegen der unglaublich vielen Kombinations- und Einstellungsmöglichkeiten einzelner Bildanalysemethoden schnell an ihre Grenzen. Es ist schlicht nicht möglich, eine Menge von Vektoren, in denen für je ein Bild alle Ausgaben aller möglichen Operatoren eingetragen sind, in Hinblick auf die relevanten Komponenten zu analysieren. Die in dieser Arbeit vorgeschlagene evolutionäre Suche umgeht dieses Problem, indem sie immer nur

eine Kombination eines kleinen Teils der möglichen Berechnungen testet und so konstruktiv an das Problem herangeht.

2.2 ALVINN

Beim ALVINN wird ein Weg beschritten, der gänzlich ohne Vorverarbeitung und explizite Featureauswahl auskommen soll. Das von Dean A. Pomerleau bereits Ende der 80er und Anfang der 90er an der Carnegie Mellon University (CMU) entwickelte, aufsehenerregende Fahrzeug-Steuerprogramm namens ALVINN (Autonomous Land Vehicle In a Neural Network) ist mittels eines neuronalen Netzes ohne Bildvorverarbeitung in der Lage, mit einer auf die Fahrbahn gerichteten Kamera auf einer gut ausgebauten Strasse die Spur bei hohen Geschwindigkeiten (55 mph) ohne Lenkeingriffe des Fahrers zu halten [29–31]. Entwickelt und getestet wurde das Steuerprogramm auf dem so genannten Navlab der CMU. Das Navlab ist ein kleiner Lastwagen mit einer vorn angebrachten Kamera. Der Lenkradeinschlag kann sowohl manuell vom Fahrer als auch automatisch von einem Computer bestimmt werden.

ALVINN ist ein Steuerprogramm für das Navlab, das im wesentlichen aus einem einfachen Feedforward Netz besteht. Die von der Kamera kommenden Bilder werden auf eine Auflösung von 30×32 Pixel reduziert und ansonsten unverarbeitet an die 960 Neuronen der Eingabeschicht angelegt. Die Ausgabeschicht besteht aus 30-50 Neuronen, von denen jedes Neuron eine bestimmte Lenkradposition repräsentiert. Das Netz wird mittels überwachten Lernens trainiert. Dazu werden Trainingsbeispiele aus den Handlungen des Fahrers generiert (Bild und aktueller, vom Fahrer bestimmter Lenkradeinschlag).

Damit das Training erfolgreich verläuft, werden einige Tricks angewendet. Um die Zahl der Trainingsbeispiele auf einer kurzen Strecke zu erhöhen und auch nicht gesehene Situationen zu trainieren, werden die aufgenommenen Bilder unter Ausnutzung der Kenntnis über die Lage der Fahrbahnebene um verschiedene Größen translatiert und rotiert. Fehlende Pixel werden dabei extrapoliert. Die Steuersignale werden unter Kenntnis der Systemeigenschaften entsprechend angepasst. So werden in einer relativ kurzen Trainingsphase auch Beispiele für Situationen erzeugt, die in der kurzen Trainingszeit gar nicht aufgetreten sind. Da jüngere Trainingsbeispiele bereits gelerntes Wissen aus dem Netz verdrängen können, werden alte Beispiele gepuffert und bei jedem Zyklus der Trainingsphase wiederholt nachtrainiert.

Mit diesem System ist es gelungen, ein Fahrzeug auf gut markierten Fahrbahnen zu steuern. Auch um Kurven soll es sicher fahren können und Hindernisse sollen erkannt werden. Allerdings muss das System auf jede neue Fahrbahn trainiert werden und zeigt Probleme mit der Auswahl von falschen Features (Pfütze anstelle der Fahrbahnmarkierungen). Pomerlau ist auch lange Strecken gefahren, aber oft mußte das Netz aufgrund der veränderten Umgebungsbedingungen nach einiger Zeit neu trainiert werden.

Generell ist fraglich, ob eine direkte Übertragung dieses Ansatzes mit der starken Reduzierung der Auflösung (die Markierungen sind groß genug und

werden vermutlich nach der Reduktion sogar noch deutlicher und einfacher zu finden sein) auf andere Aufgaben mit komplexeren Featuren möglich ist. Bei der aktuellen Version des Navlabs werden heute zumindest mehrere spezialisierte, handkodierte Featuredetektoren mit anschließender sensorische Integration an Stelle des ALVINN verwendet [42].

2.3 Evolution von rekurrenten neuronalen Kontrollern

Frank Paseman beschreibt in [27, 28] den Einsatz evolutionärer Algorithmen, um Topologien neuronaler Netze inklusive der Werte der Gewichte für an Braitenbergs Gedankenexperimente [7] angelehnte Navigationsaufgaben zu optimieren. In einer der Aufgabenstellungen geht es darum, ein neuronales Netz zu finden, das die zwei Motoren des Differentialantriebes eines kleinen Khepera Roboters [13, 22] so ansteuert, dass er sich möglichst schnell in einem Labyrinth fortbewegt, ohne an die Wände zu stoßen. Als einzige Eingabe dienen hierbei die Werte der auf der linken und rechten Seite der Front angebrachten Abstandssensoren (in manchen Experimenten wurden auch die Werte der zwei nach hinten gerichteten Abstandssensoren des Kheperas berücksichtigt). Die so evolvierten Netze werden in einer Simulation im Labyrinth getestet. Die Performance (ohne Kollisionen zurückgelegte Strecke nach einer festen Zeitvorgabe) dient dann als Fitnessmaß für die Auswahl der Vorfahren für die nächste Generation.

Zwar werden bei dieser Arbeit keine visuellen Daten verarbeitet, sondern nur sehr kleine, zwei- [28] bis achtdimensionale Featurevektoren [27], dennoch ist die Idee, Kontroller evolutionär zu verbessern und direkt in der Umgebung zu testen, sehr interessant. Außerdem kommt dieser Ansatz im Gegensatz zum ALVINN völlig ohne von außen bereitgestellte Trainingsdaten aus. Es handelt sich somit um ein selbstorganisierendes Lernverfahren.

Allerdings wird bei diesem Ansatz nicht aus Erfahrungen gelernt. In der Simulation wird lediglich getestet, wie sich ein Individuum verhält. Dem Individuum ist es nicht erlaubt, an Hand der gewonnenen Erfahrungen in einem Evaluationszyklus das Verhalten, z.B. durch eine Gewichtsänderung und / oder Ansätze aus dem Reinforcement Learning, zu adaptieren. Pasemann beschränkt sich völlig auf die Verbesserung des Genotypen, während der Phenotyp (Instanz des auf einen Khepera aufgespielten Kontrollers) überhaupt nicht lernt.

Der Nachteil dieses einfachen und eleganten Ansatzes ist daher auch die Anzahl der benötigten Iterationen. Um diese, an der Dimension der Eingabe- und Ausgabedaten gemessene, im Vergleich zu dem in dieser Arbeit verfolgten Vorhaben, recht einfache Aufgabe zu lösen, müssen bereits 100 Generationen mit einer durchschnittlichen Populationsgröße von 30 Individuen untersucht werden. Man kann erwarten, dass diese Zahl bei der Verwendung eines wesentlich höherdimensionalen Suchraumes in nicht zu bewältigende Größenordnungen ansteigen würde. Daher wäre eine Kombination von genotypischem und phenotypischem Lernen erstrebenswert.

2.4 Evolution von featureextrahierenden Programmen

Für den Bereich visueller Features gibt es bereits verschiedene Versuche, die Featureextraktoren mittels evolutionärer Algorithmen aus einfachen Primitiven zu konstruieren. Martin C. Martin [26] untersucht den Einsatz genetischen Programmierens zur Programmierung eines Algorithmus, der in einer beliebigen Bildspalte die $y-$Koordinate des ersten nicht zum Boden gehörenden Pixels berechnet. Die Kandidatenprogramme werden auf einer Serie von Bildern, für die die richtigen Daten von einem Menschen bereitgestellt werden müssen, getestet. Die Fitness wird als durchschnittliche Abweichung von der tatsächlichen Position der Kante definiert.

Die Bilder werden von der Kamera eines mobilen Roboters aufgenommen, der über die Böden von Büroräume fährt. Mittels genetischen Programmierens wird dann aus einfachen Kontrollstrukturen und Bilverarbeitungsoperatoren ein Featureextraktor programmiert. Der so programmierte, auf den Trainingsbildern erfolgreichste Featureextraktor wurde anschließend von Martin auf dem Roboter dazu verwendet, die Kamerabilder in Echtzeit in sechs fest vorgegebenen Spalten zu untersuchen und die $y-$Koordinate (Entfernung) der Bodenkante an ein handkodiertes Steuerungsprogramm zu übermitteln. Mit den von Martin gefundenen Extraktoren war es möglich, einige Zeit kollisionsfrei zu navigieren.

Allerdings ist zu beachten, dass Martin keinesfalls die Steuerungsaufgabe als Ganzes lernt. Stattdessen werden die sinnvollen Features und eine geeignete, einfache Repräsentation (y-Koordinate) klar vorgegeben. Genetisch programmiert wird dann ein Algorithmus, der das vorgegebene Feature ausreichend genau extrahieren kann.

In einer nachfolgenden Arbeit schlagen Andrew J. Marek, William D. Smart und Martin C. Martin [25] vor, Problemen bei der Generalisierungsfähigkeit mit einer größeren Diversivität und Schwierigkeit der Trainingsbilder zu begegnen: Bei Tests mit vier verschiedenen Raumtypen stellen Marek et al. fest, dass die Extraktoren, die auf einer gemischten Trainingsmenge evolviert wurden, bei der Übertragung auf eine andere Umgebung im Allgemeinen deutlich bessere Ergebnisse als die auf den "homogenen" Sets evolvierten Extraktoren erzielen. Allerdings erreichen die besten Individuen einer der vier homogenen Umgebungen vergleichbare Werte: Diese Umgebung ist die einzige, bei der es sich nicht um einen Korridor, sondern um ein Büro handelt. Hier sind im Vergleich zu den Korridoren vielfältigere Formationen und unterschiedlichere Hindernisse zu berücksichtigen. Überraschend ist, dass diese Individuen, die von einem Büroraum auf die drei Korridore übertragen werden, bessere Ergebnisse liefern, als die von einem zum anderen Korridor übertragenen Individuen.

Tony Belpaeme verwendet ähnliche Primitive und genetisches Programmieren für die Erstellung allgemeinerer Bildvorverarbeitungsschichten. In seinen Versuchen wird von ihm keine Repräsentation oder Information über richtige Features vorgegeben. Er bemerkt in seiner Arbeit vielmehr, dass als Kriterium für die Güte der erstellten Extraktoren die Leistung in der Steuerungsaufgabe

selbst dienen kann. Weil die Berechnung der Fitness eines jeden Kandidaten – zumindest nach Belpaemes Aussage – so aber zu lange dauern würde, verwendet er stattdessen die Entropie der Ausgabe der Featureextraktoren als aufgabenunabhängiges Maß. Wenn das definierte Maß steigt, liegen weniger Ausgabevektoren im Featureraum dicht beieinander und können somit einfacher unterschieden werden. So ist es dann bei einer hohen Entropie möglich, die Bilder an Hand der Featurevektoren zu diskriminieren und ein Steuersignal zuzuordnen.

Leider zeigt Belpaeme nur, dass dieses Entropiemaß auf der Trainingsmenge zu guten Resultaten führt. Angaben über die Generalisierungsleistung fehlen. Es ist dem Autor auch keine andere Arbeit bekannt, in denen die generierten Featureextraktoren auf anderen als den Trainingsbildern getestet werden. Somit bleiben vor allem zwei grundsätzliche Fragen offen:

1. Liegen die Featurevektoren neuer, ungesehener Bilder ebenfalls möglichst weit von den anderen Vektoren entfernt?

2. Gehören benachbarte Vektoren zu Bildern aus derselben Klasse, bzw. bilden die Vektoren im Merkmalsraum Cluster mit anderen Vektoren von Bildern der gleichen Klasse?

Da das Fitnessmaß nur auf eine möglichst breite Streuung der Vektoren abzielt, ist eine Ordnung entsprechend der zweiten Frage zumindest unwahrscheinlich, bzw. eher zufällig. Selbst wenn die erste Frage mit ja beantwortet werden könnte, wäre in einem solchen Fall eine gute Generalisierungsleistung eines nachgeschalteten Kontrollsubsystems ausgeschlossen, während die Trainingsleistung immer noch recht hoch sein könnte. Im Falle eines neuronalen Netzes entstünde allerdings das Problem, dass das Netz zum Auswendiglernen der ungeordneten (Frage 2 ist mit nein zu beantworten) Beispiele eine sehr hohe VC-Dimension besitzen müsste. Eine hohe VC-Dimension resultiert aber auch in höheren Schranken für die Abschätzung des Generalisierungsfehlers [36] – was erfolgreiches Übetragen unwahrscheinlicher macht.

Bei Belpaemes Vorschlag handelt es sich also um einen sehr interessanten Algorithmus zur Diskriminierung einer gegebenen Bildmenge, der aber unter Verwendung der Entropie höchstwahrscheinlich nicht für die in dieser Arbeit gestellte Aufgabe geeignet ist. Die Idee, das Subsystem in der Aufgabe selbst zu testen, erscheint erfolgsversprechender, sofern sich das von Belpaeme befürchtete Laufzeitproblem lösen ließe.

Das in [11] von Bruce A. Draper beschriebene Schema Learning System (SLS) verfolgt einen solchen aufgabenorientierten Ansatz. Das SLS ist in der Lage, aufgabenspezifische visuelle Erkennungsstrategien überwacht zu lernen. Hierzu verwendet es eine Bibliothek von miteinander kombinierbaren und ausführbaren Prozeduren. Für die erweiterbare Bibliothek kommen beliebige Algorithmen aus dem Bereich maschinelles Sehen in Frage.

Allerdings müssen die Parameter der Algorithmen in der Bibliothek des SLS vorab per Hand für jede Aufgabe speziell eingestellt werden. Die Parameter werden vom Algorithmus nicht verändert [12]. In dem von Draper beschriebenen

Experiment wurden die Algorithmen in der Bibliothek außerdem speziell für das Experiment zusammengestellt (Kontur eines Gebäudes).

2.5 Evolutionäre Auswahl von Merkmalen

Bala et al. [3] setzen genetische Algorithmen (GA) in Kombination mit einem entscheidungsbaumbasierten Lernalgorithmus (ID3) [33] bei der Auswahl von vorgegebenen Merkmalen für Klassifizierungsaufgaben ein. Der Ansatz wurde bereits sowohl in einer Pixelklassifizierungsaufge (Satellitenbilder) als auch in einer Bildklassifizierungsaufgabe (Gesichtserkennung) untersucht. Im Falle der Satellitenbilder gilt es, jeden einzelnen Bildpunkt an Hand der multispektralen Werte (vier Bänder) einer 3×3 Nachbarschaft in eine von sechs Geländeklassen einzuteilen. Zur Entscheidung stehen also $4 \cdot 3 \cdot 3 = 36$ Features (Ganzzahlige Werte von 0 bis 255) zur Auswahl. Bei dem Gesichtserkennungsproblem sollen kleine Bildausschnitte von 16×12 Pixeln Größe in eine von drei Klassen (Nase, Auge, Mund) sortiert werden. Hierzu wurden für jedes Bild 105 Featurewerte aus 4×4 Masken, die mit 50% Überlappung über den gesamten Bildausschnitt gelegt wurden, berechnet. Für diese 35 Positionen der 4×4 Masken wurden jeweils der Durschnittshelligkeitswert, die Standardabweichung und die Entropie berechnet.

Somit standen in der Satellitenaufgabe eine Basismenge von 36 und in der Gesichtserkennungsaufgabe eine Basismenge von 105 Features zur Auswahl. Dem Algorithmus wurde eine Menge von Trainings- und Testdaten (Featurevektor und richtige Klasse) zur Verfügung gestellt. Nun sollten an Hand dieser Daten relevante Features ausgewählt und eine Klassifikation gelernt werden. Zur Auswahl von Untermengen der Features wurde ein einfacher genetischer Algorithmus gewählt, bei dem die Strings die ausgewählten Features kodieren. Die so erzeugten Untermengen wurden verwendet, um mittels des ID3 Algorithmus einen Entscheidungsbaum auf der Trainingsmenge zu induzieren. Dieser Entscheidungsbaum wird dann auf der Testmenge getestet, um einen Generalisierungsfehler zu erhalten. Dieser Generalisierungsfehler dient als Fitnesswert der untersuchten Untermenge von Features im genetischen Algorithmus. Mit diesem Verfahren konnte die Anzahl der berücksichtigten Features in beiden Fällen bei gleichzeitiger Verbesserung der Fehlerquote um über 50% reduziert werden.

Der von Bala et al. entwickelte hybride Algorithmus, der Evolution und lokales Lernen kombiniert, und vor allem die Vorgehensweise (Samples der Zielfunktion, Verwendung des Generalisierungsfehlers als Fitness) bilden die Basis für den in dieser Arbeit verfolgten Ansatz. Allerdings geht es bei Bala et al. darum, aus einer relativ kleinen Menge von vorgegebenen (fertigen), aufgabenspezifischen subsymbolischen (oder "low-level") Features die relevanten auszusuchen. Die hierbei erreichte Reduktion ist mit 50% genau wie die Ausgangsmenge der möglichen Features vergleichsweise klein.

In der vorliegenden Arbeit werden nicht aufgabenspezifisch die möglichen Features bereits fertig vorgegeben, sondern nur parametrisierte Berechnungs-

methoden (Operatoren) bereitgestellt, die erst konfiguriert und dann eingesetzt werden müssen, um symbolische oder subsymbolische Merkmale zu berechnen. Der Suchraum ist damit um Größenordnungen größer als bei den von Bala et al. untersuchten Problemen.

2.6 Evolutiver Netzwerk-Optimierer (ENZO)

In [8] beschreiben Heinrich Braun und Joachim Weisbrod den Einsatz evolutionärer Algorithmen zur Optimierung der Topologie von neuronalen Netzen. Braun und Weisbrod schlagen einen Algorithmus (ENZO) vor, der eine starke Repräsentation von Netztopologien verwendet, um die Struktur neuronaler Netze für ein bestimmtes Lernproblem zu optimieren. Generell ist es so, dass die optimale Topologie eines neuronalen Netzes sehr stark von der Lernaufgabe abhängig ist und nur schwer ad-hoc bestimmt werden kann. Die Aufgabe, durch empirische Tests eine günstige Topologie zu finden, wird vom ENZO Algorithmus übernommen. Die vom evolutionären Algorithmus erzeugten Kandidatenstrukturen werden hierzu mittels lokalen Trainings auf den Trainingspattern getestet.

Der Benutzer gibt für ein bestimmtes Problem nur eine maximale Netztopologie vor, die den Suchhorizont definiert. ENZO erzeugt dann eine initiale Population von Kandidaten, bei denen einzelne Verbindungen fehlen. Diese Topologien werden dann mittels lokalen Trainings (z.B. per Backpropagation) trainiert und getestet. Als Fitnesskriterien kommen Trainings- und Generalisierungsfehler, aber auch beliebige andere Maße, wie die Anzahl der Verbindungen, in Frage. Zur Erzeugung einer Nachfolgegeneration werden sowohl Mutation (Entfernung / Hinzufügung von Verbindungen) als auch Rekombination verwendet. Der Rekombinationsoperator, der zwei Topologien kombiniert, ist bei ENZO recht komplex und wählt nicht einfach nur Verbindungszustände (an oder aus) von dem einen oder anderen Netz aus, sondern versucht, funktional ähnliche Verbindungsstrukturen (die durch sehr unterschiedliche Strukturen realisiert werden können) in den beiden Vorfahren zu identifizieren und zu berücksichtigen. Darüber hinaus gibt es beim ENZO Algorithmus noch viele speziell an Netztopologien angepaßte Einstellungsmöglichkeiten. So können zum Beispiel Mindestwerte für die Verbindungsgewichte angegeben werden. Fällt ein Gewicht unter diesen Schwellwert, wird die Verbindung gelöscht. Die so evolvierten und trainierten Netze können eine bessere Generalisierungsleistung als von Hand ausgewählte Topologien erreichen.

Da bei ENZO auch alle Verbindungen von einem Eingabeneuron gelöscht werden können, ist der Algorithmus selbst auch geeignet, Features aus dem Eingabevektor auszuwählen und so die Dimensionalität zu reduzieren. In einer späteren Arbeit kombinierten Braun und Ragg [9] den ENZO Algorithmus zudem erfolgreich mit Reinforcement Learning.

Der ENZO Algorithmus ist für die vorliegende Arbeit interessant, weil auch hier das Problem der Auswahl einer geeigneten Topologie für das neuronale Netz der Kontrollschicht besteht. Bei der Erstellung des Algorithmus ist nicht

nur nicht die Aufgabe bekannt, es ist darüber hinaus sogar unbekannt, welche und wie viele Features überhaupt die Eingabevektoren bilden. Da sich die Größe und die Komplexität des Featurevektors von Kandidat zu Kandidat unterscheidet, ist es unmöglich, ad-hoc eine gut geeignete Topologie für alle Kandidaten anzugeben. Daher ist es erforderlich, in die evolutionäre Erzeugung der Kandidaten einen dem ENZO Ansatz ähnlichen Algorithmus zu integrieren, der die Netztopologie an die Featurevektoren der Kandidaten anpassen kann.

2.7 Instanzbasierte Kategorienbildung

Luc Steels und Frederic Kaplan [40] verwenden einen modifizierten AIBO Roboter, um in einem sogenannten "Language Game" zu untersuchen, wie erste Begriffe für Objekte gelernt und Kategorien gebildet werden können. In den durchgeführten Experimenten muss der Roboter in einem "social learning" genannten Vorgang die richtigen Bezeichnungen für drei verschiedene, in die Kamera gehaltene Objekte (Ball, Smiley, Spielzeug) durch eine multimodale Interaktion mit einem Mediator lernen. Das Kontrollprogramm besteht dabei aus einer komplexen Zusammenstellung von verschiedenen, vorgefertigten Handlungsschemas. Den Kern bildet dann ein adaptiver, lernender Algorithmus, der für die Kategorienbildung und Wiedererkennung zuständig ist. Im Prinzip dienen die Schemas nur dazu, aus der komplexen "sozialen Interaktion" wieder einfache Eingabe-Ausgabepaare zu bilden.

Konkret liefern die Interaktion und die Schemas einzelne Paare von Bildern und zugehörigen Wörtern (Klassen). Die Frage ist nun, wie aus diesen Zuordnungen gelernt (generalisiert) werden kann. Die Autoren entscheiden sich gegen eine Segmentierung der Bilder. Eine vollständige und korrekte Segmentierung ist generell, falls überhaupt, zuverlässig nur möglich, wenn Objektschablonen (bzw. a priori Wissen über die Problemdomäne) vorhanden sind [38, S. 123]. Eine Segmentierung, die nur "bottom up" funktioniert, kann alleine nie vollständig und korrekt sein (man denke z.B. an Schatten oder sich ändernde Beleuchtungsverhältnisse). Die Objektkategorien sind aber in der untersuchten Aufgabenstellung eben nicht vorab bekannt, sondern sollen ja erst noch gebildet werden.

Daher wählen sie einen instanzbasierten Algorithmus, der möglichst wenig (keine) Vorverarbeitung vornimmt. Weil der Vergleich von kompletten Bildern aber in der Praxis nicht handhabbar ist, entscheiden sich Steels und Kaplan für die Erstellung von zweidimensionalen Farbhistogrammen, die dann als einzige Information gespeichert und verglichen werden. Gelernt wird also eine Kategorisierung von Farbhistogrammen (und nicht von Bildern). Der Aufbau des lernenden Subsystems besteht also im Sinne dieser Arbeit aus einer fest verdrahteten Vorverarbeitungsschicht und einem lernenden "Kontroller", der im Prinzip sogar durch überwachtes Lernen trainiert wird[2].

[2]Drumherum befinden sich zwar einige Schichten "reinforcement learning" (allerdings nur von "one-step-actions") und das von Steels und Kaplan untersuchte "social learning", im Kern reduziert es sich aber auf das überwachte Lernen einer Klassifikation von Histogrammen

Interessant ist, dass sich Steels und Kaplan (zumindest in dem zitierten Paper), "nur" aus praktischen Gesichtspunkten (kein Objektwissen vorhanden) gegen eine Segmentierung und für eine (biologisch eher unplausiblere) Histogrammberechnung entschieden haben und dass diese Entscheidung nicht aus ideologischen Gründen (holistisches gegenüber analytischem Vorgehen) getroffen wurde. Es sei schlicht nicht möglich, eine korrekte Segmentierung ohne a priori Wissen vorzunehmen.

An dieser Stelle kann nun der Ansatz dieser Arbeit helfen: Dadurch, dass die Segementierung (oder auch eine andere Vorverarbeitung) zum Gegenstand des Lernprozesses selber gemacht wird, muss Wissen über die zu erkennenden Objekte nicht a priori gegeben sein. Es wird vielmehr während der Lernphase implizit gewonnen und angewendet, um die "Einstellung" der Segmentierung bzw. die Auswahl der Features (dieser Begriff wurde von Steels und Kaplan vermutlich wohlwissend vermieden) zu lenken.

Vermutlich kann man den verwendeten Instanzalgorithmus und die Bestimmung des Histogramms durch den in dieser Arbeit entwickelten Algorithmus austauschen und dennoch ähnliche, bzw. vielleicht sogar bessere Ergebnisse erzielen. Der Beweis wäre natürlich noch durch eine Überprüfung an den in der Interaktion gewonnenen Originaldatensätzen zu erbringen. Auf die Interaktion in jedem einzelnen Trainingsschritt könnte verzichtet werden, da sich der hier vorgestellte Algorithmus wie folgt leicht in das bestehende Interaktionsschema einbetten ließe: Man würde einfach die während der Interaktion hinzugewonnenen Pattern wie gehabt sammeln und den Algorithmus kontinuierlich, parallel auf diesen Daten trainieren. Da es sich hier um einen "any-time" Algorithmus handelt, könnte man die Interaktion bereits fortsetzen, auch wenn das globale Optimum noch nicht erreicht wurde. Interessant wäre die Frage, ob der hier vorgestellte Algorithmus mit der gleichen (niedrigen) Anzahl von Bildern wie der Instanzalgorithmus auskommen oder wesentlich mehr Bilder benötigen würde, um eine Quote von über 80% richtig klassifizierter Testbilder zu erreichen.

2.8 Diskussion

Mit den diskutierten Arbeiten wurde exemplarisch gezeigt, welche Ansätze zum Lernen eines Kontrollers für Navigationsaufgaben im Bereich des überwachten und des evolutionären Lernens existieren und welche Probleme diese Ansätze haben. Eine Kombination phenotypischen und genotypischen Lernens wäre für das Problem der Entscheidungsfindung auf visuellen Reizen erstrebenswert. Auch in Richtung solch hybrider Algorithmen existieren verschiedene Ansätze. Für den Bereich visueller Reize wurden einige Bemühungen, insbesondere in der Bildvorverarbeitungsschicht-Kontrollschicht-Architektur vorgenommen. Allerdings ist bei den diskutierten Ansätzen keiner dabei, der völlig ohne zusätzliches a priori Wissen eine beliebige Aufgabe lernen kann und ein vollständig gelerntes Kontrollprogramm erzeugt. Martin hat die Zwischenrepräsentation vorgeschrieben und aufgabenspezifische Features selbst ausgewählt. Im Mittelpunkt stand die automatische Programmierung passender Extraktoren. Belpaeme hat

zudem die mögliche Aufgabenorientierung in seiner Arbeit herausgestellt, aber leider nur ein aufgabenunabhängiges Maß verwendet und die Generalisierung nicht getestet. Drapa hat zwar einen kompletten zweischichtigen Kontroller eingelernt, muss aber die Parameter der verwendeten Bildverarbeitungsbausteine für jede Aufgabe vorab per Hand einstellen.

Am Beispiel von Steels und Kaplans Arbeit sieht man, wie eine Einbeziehung der Verarbeitung der Bilder in den Lernprozess selber helfen kann, so grundlegende Probleme wie die Entstehung erster Kategorien durch multimodale, soziale Interaktion anzugehen.

Kapitel 3

Der Algorithmus

3.1 Evolution von Lösungskandidaten

3.1.1 Überwachtes Lernen

Alle in dieser Arbeit untersuchten Lernprobleme, die von dem vorgestellten Algorithmus gelöst werden können, wurden als Probleme überwachten Lernens formuliert. Es gilt eine unbekannte Zielfunktion

$$f : R^n \mapsto R^m$$

zu approximieren. Die Eingabevektoren werden im Folgenden, wenn nicht anders definiert, mit $\mathbf{v}$, die Zielvektoren mit $f(\mathbf{v}) = \mathbf{t}$ und die Ausgabevektoren der Approximation f' zur besseren Unterscheidung mit $f'(\mathbf{v}) = \mathbf{y}$ bezeichnet.

Zum Lernen dieser Funktion steht eine endliche Menge S von m Beispielen (Samples) $\mathbf{s}$ der Zielfunktion mit

$$s = (\mathbf{v}, f(\mathbf{v}))$$

zur Verfügung. Die Menge S wird auch als Trainingsmenge bezeichnet.

In der Regel wird ein Fehlerterm, oftmals der quadratische Fehler (tss: Total Sum of Squares) zwischen der korrekten Ausgaben $\mathbf{t_p}$ der Zielfuntion f und der Ausgabe $\mathbf{y_p}$ der Approximation f'

$$E = 1/2 \sum_{p=1}^{m} \|\mathbf{t}_p - \mathbf{y}_p\|^2,$$

definiert und durch einen geeigneten Algorithmus (einen sogenannten Funktionsapproximator) minimiert. Die Güte der dabei approximierten Funkion f' kann imHinblick auf ihre Generalisiserungsfähigkeit an einer weiteren Menge von Punkten der Zielfunktion, die nicht in der Menge der Trainingspattern enthalten sind, getestet werden. Je geringer der Fehler der gelernten Funktion auf

dieser sogenannten Testmenge ausfällt, desto bessere Generalisierungsfähigkeit wird dem verwendeten Funktionsapproximator zugesprochen.

Da die exakte Zielfunktion in der Regel nicht bekannt ist, werden sowohl die Beispiele der Trainings- als auch der Testmenge durch einzelne, diskrete Beobachtungen der Zielfunktion (Sampling) gefunden. Hierbei wird die Ausgabe $f(\mathbf{v})$ der Funktion für einen Eingabevektor $\mathbf{v}$ gemessen. Beim Sampling der Funktion können z.B. aufgrund von Messfehlern bei der Bestimmung des Eingabezustandes oder der Ausgabe des Systems Fehler entstehen, die sich als Rauschen in den Trainings- und Testdaten niederschlagen. Erst durch die Fähigkeit, trotz solcher Fehler gute Approximationen zu erreichen, wird ein Funktionsapproximator interessant. Diese Eigenschaft wird als Robustheit gegenüber Fehlern / Rauschen auf den Trainingsdaten bezeichnet.

Im konkreten Fall gilt es, eine Funktion zu lernen, die für ein Bild $\mathcal{B} \in [0,255]^{3^{k\times l}}$ einen Funktionswert $\mathbf{y}$ bestimmt. Hier ist ein Bild eine $k \times l$-Matrix, deren Einträge $\mathbf{b}_{i,j}$ eine Repräsentierung des Farbwertes des Bildpunktes (Pixel) an der Stelle i, j beinhaltet. Der Farb- oder Helligkeitswert eines Bildpunktes ist ein Tupel, das die Koordinaten in einem ein- oder mehrdimensionalen Farbraum angibt. Eine Übersicht über verschiedene Farbräume und gebräuchliche Notation kann man z.B. in [14] finden. In dieser Arbeit werden an den Schnittstellen ausschließlich Bilder mit Farbtupeln aus dem RGB Farbraum (drei Kanäle für Rot, Grün und Blau) verwendet. Dabei gilt für die Einträge der Matrix $\mathcal{B}$ implementationsbedingt $\mathbf{b}_{i,j} \in [0,255]^3$. Der Ausgabevektor $\mathbf{t}$ der zu lernenden Zielfunktion f kann beliebige Dinge kodieren: Es kann sich zum Beispiel um die Kodierung einer Klassenzugehörigkeit oder eines Steuersignals für die Aktoren eines Roboters handeln. Auch die Ausgabe eines weiteren Bildes, bzw. einer anderen ikonischen Darstellung gewonnener Informationen ist hier denkbar (z.B. könnte ein Regionen- oder Kantenbild erzeugt werden). Die zu approximierende Funktion hat jedenfalls die Form:

$$f : [0,255]^{3^{k\times l}} \mapsto [0,1]^n$$

3.1.2 Struktur der Kandidaten

In den vorherigen Abschnitten wurde argumentiert, dass das Lernen einer solchen Funktion direkt auf den hochdimensionalen Bilddaten mit den derzeit zur Verfügung stehenden Algorithmen aus rein praktischen Gesichtspunkten scheitert.

Daher wird die Funktion untergliedert und eine Architektur aus getrennter Vorverarbeitungsschicht und nachgeschalteter höherer Schicht gewählt. Die Vorverarbeitungsschicht realisiert das Low-Level Vision, während die höhere Schicht sowohl Funktionen des High-Level Vision als auch die Generierung des Ausgabesignals vereint:

$$f(\mathcal{B}) = g \bullet h(\mathcal{B})$$

Wobei die Vorverarbeitungsschicht

$$h : [0,255]^{3^{k\times l}} \mapsto R^q$$

einen q dimensionalen Vektor $\mathbf{v}'$ aus dem Bild $\mathcal{B}$ berechnet und

$$g : R^q \mapsto R^n$$

auf den von der Vorverarbeitungschicht berechneten Daten die Ausgabe $\mathbf{y} \in R^n$ berechnet. Die Vorverarbeitungsschicht bringt die ursprünglichen Bilder in eine vereinfachte Form. Dies kann durch eine Reduktion der Dimension und Berechnung eines sogenannten Feature Vektors, durch eine Vereinfachung der ikonischen Darstellung (z.B. durch die Berechnung eines binären Kantenbildes) oder durch eine Kombination beider Techniken geschehen. In der Regel gilt hier $q \ll k \cdot l$. Die Funktion g wird in der gewohnten Weise durch einen Funktionsapproximator – hier durch ein einfaches neuronales Netz – realisiert.

Die durch h realisierte Vorverarbeitungsschicht selber soll nun aber im Gegensatz zu den "herkömmlichen" Verfahren nicht aufgabenspezifisch per Hand erstellt und "fest verdrahtet" werden, sondern durch einen evolutionären Ansatz während der Trainingsphase optimiert werden. Zu diesem Zweck wird die Vorverarbeitungsschicht weiter untergliedert.

Die Vorverarbeitungsschicht setzt sich aus einer endlichen (zumeist kleinen) Menge von parallel angeordneten, handkodierten, parametrisierten Bausteinen zusammen (siehe Abbildung 3.1). Ein solcher Baustein, im Folgenden Operator genannt, berechnet einen Vektor $\mathbf{u}$ für eine Bildmatrix $\mathcal{B}$:

$$o(\mathcal{B}) = \mathbf{u}\,.$$

Als Operatoren kommen hier einfache Filter wie Gauß'sche Weichzeichner, Point-of-interest Detektoren wie Eckendetektoren oder sogar komplexe Programme wie Segmentierungsalgorithmen in Frage. Auf die Operatoren wird weiter unten genauer eingegangen.

Die Ausgabe der Operatoren für ein Bild wird parallel berechnet und anschließend zur Gesamtausgabe der Vorverarbeitungsschicht konkateniert:

$$h(\mathcal{B}) = \mathbf{x}' = (o_1(\mathcal{B}), o_2(\mathcal{B}), \ldots, o_n(\mathcal{B}))$$

3.1.3 Veränderbare Parameter der Kandidaten

Die Arbeitsweise und die Ausgabe der Operatoren kann durch unterschiedlich viele Parameter beeinflusst werden. Für die Operatoren muss in der vorgeschlagenen Architektur gelten:

$$o_{\mathbf{p}}(\mathcal{B}) \neq o_{\mathbf{p}'}(\mathcal{B}) \Rightarrow \mathbf{p} \neq \mathbf{p}'$$

wobei $\mathbf{p}$ ein beliebiger "passender" Parametervektor der richtigen Dimension sei. Während Operatoren also deterministisch sein müssen und auch keinen internen Zustand berücksichtigen dürfen, gilt die Umkehrung der Bedingung nicht: Ein Operator kann für das gleiche Bild trotz unterschiedlich eingestellter Parameter unter Umständen die gleiche Ausgabe liefern. Diese Art der Operatordefinition

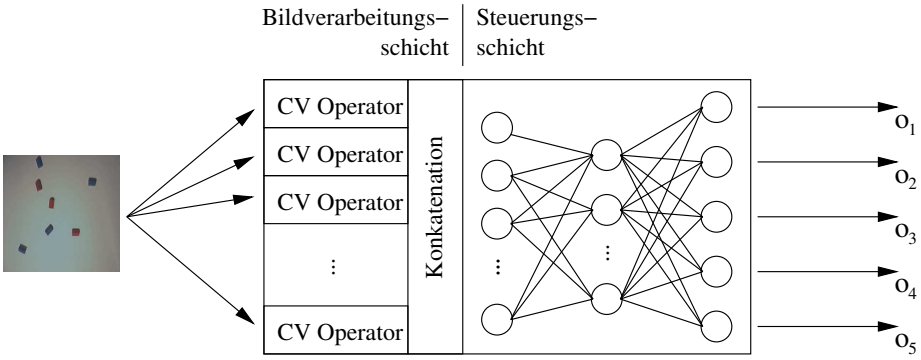

Abbildung 3.1: Die Struktur eines Kandidaten. Der Kandidat besteht aus einer Bildverarbeitungsschicht und einem nachfolgenden Feedforward Netz, das die Kontrollschicht realisiert. Die Bildvorverarbeitungsschicht besteht aus kleineren Bausteinen (Operatoren), die alle direkt auf die Bildmatrix zugreifen. Die einzelnen Ausgaben werden zur Gesamtausgabe der Schicht konkateniert

ist in Verbindung mit dem nachgeschalteten, nicht rekurrenten neuronalen Netz insbesondere für die Lösung von Markov'schen Entscheidungsprozessen (MDP) [5] ausgelegt . Eine Erweiterung für darüber hinausgehende Probleme, bei denen zum Beispiel eine Bildfolge eine Rolle spielt (Berechnung der Geschwindigkeit), ist an verschiedenen Stellen denkbar, wird an dieser Stelle aber nicht weiter verfolgt.

Die Parameter untergliedern sich in zwei Gruppen. Alle eingesetzten Operatoren benötigen mindestens einen Parameter, der den Algorithmus selber beeinflußt und den Operator so an die Umgebungsverhältnisse anpasst. Außerdem haben die Operatoren eine Reihe von Parametern, mit denen ihre Ausgabe angepasst werden kann. So können mittels boolscher Parameter einzelne Werte, die die Operatoren berechnen und ausgeben können, an- oder abgewählt werden. Oftmals können auch die Menge oder die Ordnung der für die Ausgabe zu berücksichtigenden, gefundenen Merkmale, nach denen der Operator im Bild gesucht hat, durch Parameter verändert werden.

Zusammenfassend gibt es also zwei Arten, die Vorverarbeitungsschicht zu verändern: Zum einen können die Parameter der Operatoren verändert und damit deren Arbeitsweise und Ausgabe beeinflusst werden. Zum anderen kann die Zusammenstellung der Operatoren selber angepasst werden.

Von der Zusammenstellung der Operatoren in der Vorverarbeitungsschicht der Individuen ist aber auch die Dimension der Eingabeschicht des die Funktion g realisierenden neuronalen Netzes abhängig. Es liegt nahe, dass sich auch die weitere Topologie des neuronalen Netzes an die veränderten Eingabevektoren anpassen sollte.[1] Daher bietet es sich an, auch die Topologie des Netzes

[1] Theoretisch wäre auch denkbar, die Eingabeschicht so zu dimensionieren, dass sie für alle Fälle groß genug ist. Eventuell freie Stellen würde man dann mit 0 auffüllen. Die anderen

nicht festzulegen, sondern während der Trainingsphase, in Anlehnung an den ENZO Algorithmus (siehe Abschnitt 2.6) mitzulernen. Allerdings können keine einzelnen Verbindungen und auch nicht die Neuronen der Eingabeschicht wie bei ENZO geprunt werden. Die Breite der Eingabeschicht wird allein durch die Ausgabe der Vorverarbeitungsschicht bestimmt. Das Pruning nicht relevanter Eingabewerte wird in der vorliegenden Architektur auf Seiten der Operatoren der Bildverarbeitungsschicht durch Veränderung der die Ausgabe beeinflussenden Parameter vorgenommen. Die Anzahl der Ausgabeneuronen wird ebenfalls durch die Trainingsdaten (die Dimension der Ausgabevektoren $\mathbf{t}$) festgelegt. Die Anzahl der versteckten Netzschichten und der dort enthaltenen Neuronen sowie die Entscheidung, ob Shortcut Verbindungen erstellt werden sollen oder nicht, wird aber durch den gleichen evolutionären Algorithmus, der auch die Operatoren und deren Parameter auswählt, optimiert.

Ein Individuum x aus der Menge der "Kandidaten" K besteht also aus einer Auswahl von Operatoren samt zugehörigen Parametervektoren und einem weiteren Parametervektor

$$\mathbf{p}_{net} \in N \times N \times \{0,1\}\,,$$

der die Anzahl der Neuronen pro versteckter Netzschicht, die Anzahl der versteckten Schichten und das Vorhandensein von Shortcutverbindungen bestimmt.

3.1.4 Der evolutionäre Algorithmus

Die äußere Schleife des hybriden Algorithmus ist ein evolutionärer Algorithmus. Zum Start wird eine Population X mit N Individuen zufällig initialisiert. Dann wird eine (μ, λ)-evolutionäre Strategie mit einer elitären Fortpflanzung verfolgt. Die Population nach t Iterationsschritten wird auch t-te Generation der t-ten Epoche genannt. Die "Verbesserung" wird als Maximierung einer Fitnessfunktion $f : K \mapsto R$, K sei hier die Menge der möglichen Kandidaten, definiert, wobei $f(x) > 0$ die "Fitness des Individuums x" genannt wird. Beim hier definierten überwachten Lernproblem wird die Fitnessfunktion weiter unten an Hand des Fehlers der Netzausgabe auf einer Menge von bereitgestellten Sampeln definiert werden. Die Berechnung der Fitness der Kandidaten einer Population X_t durch überwachtes Training eines Feedforward Netzes bildet die innere Schleife des Algorithmus.

Der evolutionäre Algorithmus untergliedert sich in die fünf Schritte Selektion, Rekombination, Mutation, Evaluation und Wiedereinfügung (siehe Abbildung 3.2). In der Selektionsphase werden μ Vorfahren ausgewählt und bilden in der Rekombinationsphase μ Nachfahren. Diese werden anschließend mutiert und dann in der Evaluationsphase bewertet. Die λ besten Nachfahren ersetzen in der Wiedereinfügungsphase die λ schlechtesten Individuen der Generation X_t um die nachfolgende Generation X_{t+1} zu formen.

Je nach Selektions- und Vereinigungmethode kann sich die Fitness der Population durchaus verschlechtern. Auf lange Sicht wird aber die Wahrscheinlich-

Schichten würden dann für den "Maximalfall" angelegt werden.

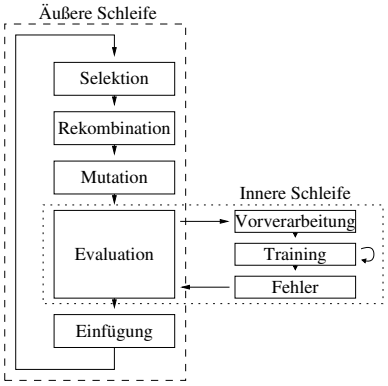

Abbildung 3.2: Die verschiedenen Schritte des Algorithmus. Die äußere Schleife besteht aus fünf verschiedenen Schritten. Die innere Evaluationsschleife bewertet die Kandidaten, indem deren Bildverarbeitungsschichten zur Vorverarbeitung der Trainingsbilder verwendet wird. Das neuronale Netz wird dann auf den so erhaltenen vorverarbeiteten Trainingspattern einige Epochen trainiert. Die dabei gemessenen Trainings- und Testfehler werden zur Berechnung der Fitness herangezogen.

keit, dass sich ein optimales Individuum – also ein Individuum mit einer maximalen Fitness – in der Population befindet, maximiert. Es gibt auch Verfahren, bei denen die Fitness der Population monoton steigt. Dies ist zum Beispiel dann der Fall, wenn im Vereinigungsschritt immer die n besten Individuen aus der Vereinigung der Population X_t und der Menge der Nachfahren für die nächste Generation ausgewählt werden.

Selektion

In der Selektionsphase werden μ Individuen zur Fortpflanzung aus den n Individuen der Population X_t zufällig per "Roulette-Wheel-Selection" ausgewählt. Dabei ist die Wahrscheinlichkeit $P(\mathbf{x}_i)$, das Individuum $\mathbf{x}_i$ als k-tes Elternteil auszuwählen, proportional zu dessen Fitness $f(\mathbf{x}_i)$:

$$P(\mathbf{x}_i) = \frac{f(\mathbf{x}_i)}{\sum_{\mathbf{x}_j \in \mathbf{X}_t} f(\mathbf{x}_j)} \quad .$$

Ein einzelnes Individuum kann hierbei auch mehrmals zur Fortpflanzung ausgewählt werden. Die gezogenen Individuen werden in einer nach der Ziehungsreihenfolge geordneten Liste an den Rekombinationsschritt weitergegeben.

Rekombination

In der Rekombinationsphase wird aus jedem Vorfahren und dem direkt nach ihm gezogenen Vorfahren ein neues Individuum gebildet. Das i-te Individuum

wird somit sowohl mit dem $i-1$-ten Individuum als auch mit dem $i+1$-ten Individuum gepaart.

Bei der Rekombination werden aus der Vereinigung der Operatormengen O der beiden Elternteile $1 \geq n \geq \mid O_i \mid + \mid O_{i+1} \mid$ Operatoren für den Nachfolger ausgewählt. Die Parameter der ausgewählten Operatoren werden hierbei nicht verändert. Die Topologie des neuronalen Netzes wird vom ersten Elternteil unverändert kopiert.

Mittels einer einstellbaren Rekombinationsrate $r_{\text{rekombination}}$ kann beeinflußt werden, ob alle oder nur ein Teil der Eltern in der dargestellten Weise kombiniert werden. Für Werte mit $r_{\text{rekombination}} < 1$ beträgt die Wahrscheinlichkeit, dass ein bestimmtes Elternteil bis zum Mutationsschritt unverändert bleibt $1 - r_{\text{rekombination}}$.

Mutation

Die durch Rekombination entstandenen Nachfahren, bzw. die unverändert weitergegebenen Vorfahren werden nun einzeln zu einer gewissen Wahrscheinlichkeit r_{mutation} (Mutationsrate) mutiert. Die Mutation kann dabei die Auswahl der Operatoren, die Parametervektoren und das neuronale Netz verändern.

Es kann zur Menge der Operatoren der Bildverarbeitungsschicht ein zusätzlicher, zufällig ausgewählter und initialisierter Operator aus der Menge der vorhandenen Operatoren hinzugefügt werden. Ebenfalls kann ein Operator aus der Schicht entfernt werden.

Einige Einträge der Paramtervektoren der einzelnen Operatoren werden durch Addition einer normalverteilten Zufallsvariablen mit einem Durchschnitt von 0 verändert. Die Wahrscheinlichkeit, dass der Eintrag p_i mutiert wird, wird durch einen weiteren Wert $r_{parameter}$ bestimmt. Falls der Operator eine flexible Anzahl von Parametern besitzt, können auch noch weitere zufallsinitialisierte Parameter hinzugefügt oder entfernt werden.

Der Parametervektor des neuronalen Netzes, der dessen Topologie bestimmt, wird in der gleichen Form wie die Parmatervektoren der Operatoren mutiert. Nach der vollständigen Mutation eines Individuums wird geprüft, ob die vorgenommenen Veränderungen noch im Rahmen der erlaubten Einstellungen liegen. Falls ein Fehler passiert ist und zum Beispiel der letzte Operator aus der Vorverarbeitungsschicht gelöscht wurde oder die Parameter unzulässige Werte oder Wertekombinationen angenommen haben (die zulässigen Wertebereiche werden von den Operatoren selber bestimmt), werden alle Änderungen an diesem Nachfahren zurückgesetzt und die Mutation wird noch einmal von vorne begonnen.

Fitnessfunktion

Die bis zu dieser Schicht erhaltenen mutierten Nachfahren werden einzeln auf ihre Fitness getestet. Zu diesem Zweck werden die bereitgestellten Pattern $(\mathcal{B}, f(\mathcal{B}))$ verwendet. Die Bilder werden einzeln durch die Vorverarbeitungsschicht des zu testenden Kandidaten verarbeitet. Die dadurch erhaltenen Pattern $(h(\mathcal{B}), f(\mathcal{B}))$ haben die für die Eingabeschicht des neuronalen Netzes pas-

sende konstante Größe. Auf einem Teil der Pattern (Trainingsset) wird das neuronale Netz nun für eine fest vorgegebene Anzahl von Epochen (in allen Experimenten 500 Epochen) mittels Resilient Propagation (Rprop) [34] trainiert. Nach Abschluß des Trainings wird auf den Trainingspattern der Trainingsfehler des fertig trainierten Netzes bestimmt. In den Experimenten hat sich gezeigt, dass dieser Wert sehr schnell sehr niedrig ist (siehe zum Beispiel Abb. 4.8). Es bietet sich daher an, auch die aussagekräftigere Generalisierungsleistung des Netzes zu testen. Hierzu wird eine von der Trainingsmenge disjunkte Testmenge von Pattern verwendet. Auf dieser Menge wird der gleiche Fehlerterm bestimmt.

Wie gesagt, es gibt schon zu Beginn der Evolution zumeist ein Individuum, dessen Vorverarbeitungsschicht einen – zumeist hochdimensionalen – Featurevektor ausgibt, der dazu geeignet ist, die Trainingsmenge auswendig zu lernen. Der Erfolg auf den ungesehenen Beispielen der Testmenge ist dann aber deutlich niedriger, weil die Strukturen nicht wirklich herausgearbeitet wurden. Die Generalisierungsleistung ist daher anfangs sehr gering und steigt während der Evolution nur langsam an. Da der Generalisierungsfehler niedrig bleibt, solange in der Vorverarbeitungsschicht nicht die aufgabenspezifischen Merkmale und Gemeinsamkeiten herausgearbeitet werden, ist er ein gutes Maß für die Qualität des Bildverarbeitungs-Subsystems und deshalb eine zentrale Komponente der berechneten Fitness. Auch der weniger aussagekräftige Trainingsfehler wird in dem Fitnesswert berücksichtigt, um die Selektion gerade zu Anfang der Evolution zu lenken, wo der Generalisierungsfehler noch bei allen Individuen niedrig ist.

In den Experimenten hat sich die Berücksichtigung einiger weiterer Komponenten als günstig erwiesen. So wird die von der Vorverarbeitungsschicht benötigte Rechenzeit zur Analyse eines Bildes gemessen. Schnellere Schichten werden belohnt. Größere Netzeingabeschichten und mehr Neuronen in den versteckten Netzschichten werden bestraft.

Der innere Evaluationszyklus ist der aufwändigste Schritt des Evolutionszyklus. Es sind für jeden Operator eines jeden Kandidaten unter Umständen mehrere hundert Bilder zu analysieren. Je nach ihrer Einstellung können die einzelnen Operatoren mehrere Hundert bis Tausende von einzelnen Werten zurückliefern. Das Training eines neuronalen Netzes mit entsprechend vielen Eingabeneuronen und Verbindungen kann dementsprechend lange dauern. Daher helfen die Einbeziehung der benötigten Rechenzeit und Netzgröße in die Fitness der Kandidaten, die Evolution bereits zu Anfang in Bezug auf die Laufzeit des Evaluationsschrittes erheblich zu beschleunigen.

Alle Komponenten der Fitnessfunktion werden normiert und so angeordnet, dass größere Fitnesswerte bessere Leistungen (schnellere Operatoren, kleinere Netze) bedeuten. Aus den normierten Komponenten wird dann eine gewichtete Summe gebildet, die die Fitness des Individuums ist. Hierbei werden die Fehlerterme wesentlich stärker ($> 5:1$) als die anderen Komponenten gewichtet.

Die konkreten Werte der Gewichte wurden nicht mit ausführlichen Tests optimiert, sondern ad hoc gesetzt und blieben in allen Experimenten gleich. Es ist wichtig, dass die Werte der Parameter so einfach eingestellt werden und in allen Experimenten gleich bleiben können. Ansonsten hätte man bloß die

Beschreibung der einen Parameter (Operatoren) durch die Bestimmung anderer Parameter (evolutionäre Algorithmus) ersetzt. Das Ziel ist es ja aber gerade, einen Algorithmus zu liefern, der keinerlei Eingriffe von außen benötigt.

Einfügung

Die nachfolgende Generation wird nun aus den rekombinierten und mutierten Nachfahren und der alten Population gebildet. Dabei werden die λ besten Individuen unter den Nachfahren bestimmt, die die schlechtesten λ Individuen der Population X_t ersetzen. Wenn λ kleiner als die Populationsgröße $\mid X_t \mid$ ist, dann überleben die $\mid X_t \mid -\lambda$ besten Individuen (die Elite) der Population. Dadurch kann sichergestellt werden, dass besonders gute Lösungen nicht verlorengehen.

Während der Verfolgung dieser elitären evolutionären Strategie kann die durchschnittliche Fitness der Individuen einer Population durchaus von einer Generation zur nächsten sinken. Ist λ kleiner als die Populationsgröße, ist aber sichergestellt, dass die Fitness des besten Individuums von Generation zu Generation monoton steigt. Wenn man nicht in jedem Fall die λ schlechtesten Individuen aus der Population ersetzt, sondern einfach die besten Individuen aus der ursprünglichen Population und den Nachfahren auswählt, kann man erreichen, dass auch die Durchschnitts-Fitness monoton anwächst.

Unabhängig von diesen Einstellungen und den möglichen "lokalen" Verschlechterungen nimmt aber die Wahrscheinlichkeit mit Voranschreiten der Evolution zu, dass sich der, bzw. einer der optimalen Kandidaten in der Population befindet. In der Praxis kann man aber beobachtet, dass bei dem beschriebenen Algorithmus die Population oftmals zu früh von einzelnen "suboptimalen" Operatoren dominiert wird.

Beginn mit separaten Populationen

Die beobachtete frühe Dominanz nicht optimaler Operatoren und das damit verbundene Festhängen in lokalen Maxima verhinderten oftmals, dass die optimale Lösung gefunden wurde. Zwar besteht auch bei einer solchen Situation eine von null verschiedene Wahrscheinlichkeit, dass das Optimum gefunden wird, allerdings ist diese Wahrscheinlichkeit so gering, dass in der Praxis in den allermeisten Fällen innerhalb endlich vieler Schritte nur eine suboptimale Lösung gefunden werden kann.

Es stellt sich nun die Frage, warum es zu Beginn der Evolution zur Auswahl der falschen Operatoren kommen kann und so das Finden optimaler Lösungen verhindert wird.

Der Hauptgrund ist, dass die Operatoren am Anfang sehr unterschiedliche Ergebnisse liefern. Die verwendeten globalen Operatoren sind zum Beispiel nicht so sehr von der Einstellung ihrer Parameter abhängig, die nur die "Granularität", nicht aber die Arbeitsweise verändern. Generell ist ihr Parameterraum eher klein. Bei manch anderem Operator ist der Parameterraum selber aber schon riesig und hat ganz erhebliche Einflüsse auf den Operator. So werden hier oftmals mehrere Epochen benötigt, bis der Operator überhaupt auch nur ansatz-

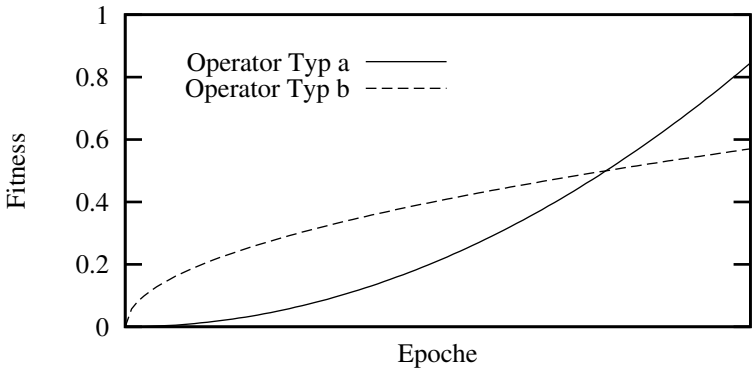

Abbildung 3.3: Getrennte evolutionäre Optimierung der Parametervektoren zweier verschiedener Operatortypen (hypothetisch). Die Fitness des Operators a steigt zwar am Anfang nur langsam an, erreicht aber am Ende höhere Werte als die des Operators b. Werden Vorverarbeitungsschichten mit diesen beiden Typen von Operatoren evolviert, kann es sein, dass der Operator b die gesamte Population zu früh dominiert und Operatoren des Typs a völlig verdrängt. Optimale Lösungen können dann nicht in annehmbarer Zeit gefunden werden (siehe Text).

weise hilfreiche Informationen liefert. Wenn nun ein globaler Operator bereits zu Beginn Informationen liefert, die bei einigen Trainingsbeispielen hilfreich sind, kann es passieren, dass die schwerer einzustellenden Operatoren verdrängt werden. Dies passiert vor allem deshalb, weil die verwendeten Populationsgrößen aus Gründen der Rechenzeit sehr klein (30-60 Individuen) sind.

Problematisch wird es, wenn die am Anfang verdrängten Operatoren zwar nur langsam in ihrer Fitness steigen, aber am Ende einen höheren Wert als die schnell gestarteten Operatoren erreichen (siehe Abb. 3.3). Dann ist die Wahrscheinlichkeit hoch, dass die optimale Lösung nicht in einer endlichen Zahl von Epochen gefunden wird. Ob und wenn ja welcher der bereitgestellten Operatoren andere, am Ende bessere Operatoren aus der Population verdrängt, ist von der zu lernenden Aufgabe abhängig und kann vorab nicht auf einfache Weise festgestellt werden.

In der Literatur findet man verschiedene Methoden, die helfen können, diesem Effekt entgegenzuwirken. Zum Beispiel kann man die komplette Evolution in verschiedenen, voneinander getrennten Subpopulationen parallel ablaufen lassen [41]. Die Wahrscheinlichkeit steigt dann, dass die dominierenden Lösungen in jeder Population unterschiedlich sind und vielleicht eine optimale Lösung dabei ist – selbst dann wenn jede Population von wenigen, ähnlichen Kandidaten dominiert wird. Zwischen den Populationen werden nach einem vorgegebenen Schema von Zeit zu Zeit – zumeist die besten – Kandidaten ausgetauscht.

Hier wurde eine einfachere und weniger aufwändige Lösung gewählt. Weni-

ger aufwändig bezieht sich hier auf die Anzahl der zu evaluierenden Kandidaten und die damit verbundene Rechenzeit. So wird die Evolution in getrennten Subpopulationen gestartet. Die Vorverarbeitungsschichten der Individuen einer Population dürfen alle nur Operatoren eines Typs aus dem Operatorpool enthalten. Für jeden im Pool vorhandenen Operator wird eine eigene Population angelegt und für eine bestimmte Anzahl von Epochen (30 Epochen haben sich als ausreichend erwiesen, siehe auch Tab. 4.2) evolviert. Es werden während dieser Startphase keine Individuen zwischen den Subpopulationen ausgetauscht. Nach dieser Startphase ist sichergestellt, dass auch die Operatoren mit größerem Parameterraum zumindest grob auf die Aufgabe eingestellt sind.

Anschließend werden aus den einzelnen Subpopulationen je n Individuen für die heterogene Startpopulation der Hauptevolutionsphase gezogen. Dabei ist die Ziehungswahrscheinlichkeit wieder proportional zur Fitness (siehe Abschnitt 3.1.4).

3.2 Die eingesetzten Bildverarbeitungsoperatoren

Die Bildverarbeitungssubsysteme der Kandidaten werden aus mehreren Operatoren, deren Ausgaben für die Gesamtausgabe aneinander gehängt werden, zusammengesetzt. Hierzu wird dem evolutionären Algorithmus ein Pool von gut untersuchten, fertigen Algorithmen aus dem Bereich maschinelles Sehen bereitgestellt. In Frage kommen hier vor allem Algorithmen, die ausreichend schnell sind, um eine Verarbeitung in Echtzeit zu ermöglichen. Nur so kann das Ziel eines echtzeitfähigen Kontrollprogramms erreicht werden. Die Schnelligkeit spielt auch bei der Evolution eine entscheidende Rolle. Es müssen schließlich bei jedem Evolutionsschritt alle Kandidaten auf der Menge von Trainingsbildern trainiert und getestet werden. Hierzu müssen pro Kandidat alle Bilder der Trainings- und der Testmenge mit dessen Vorverarbeitungsschicht, die aus mehreren Operatoren bestehen kann, analysiert werden. Von daher hilft es, wenn die einzelnen Operatoren möglichst viele Bilder pro Sekunde bearbeiten können.

Die Operatoren haben alle den gleichen Aufruf: Sie erhalten eine einzelne Bildmatrix und liefern einen reellwertigen Vektor zurück. Außerdem können sie mit einer Menge von Parametern konfiguriert werden. Hierbei sind zwei Gruppen von Parametern zu unterscheiden: Die eine Gruppe dient dazu, den Algorithmus einzurichten und an die Umgebungsbedingungen anzupassen. Die andere Gruppe hat keinen direkten Einfluss auf den Algorithmus. Parameter dieser zweiten Gruppe beeinflussen den Aufbau des Ausgabevektors und wählen einzelne Features, die der Algorithmus berechnen kann, an oder ab. Zum Beispiel kann über diese Parameter beim CVTK-Operator eingestellt werden, dass nur Werte der n größten gefundenen Flächen ausgegeben und alle anderen Flächen ignoriert werden.

Die reellwertigen Parameter werden von den Operatoren als einer der drei Typen int (ganze Zahl), bool (Wahrheitswert) oder double (Fließkommazahl)

interpretiert. Die Umwandlung der reellwertigen Parameter zu int und bool geschieht durch Abschneiden der Nachkommastellen (nicht durch Runden!). Parameter können neben diesen einfachen Zahlen auch Tupel fester Länge sein (z.B. ein dreikanaliger Farbwert). In der Regel wird jeder einzelne, so definierte und intern mit einer Semantik versehene Parameter nur einfach angegeben. In manchen Fällen kann aber auch eine Liste von solchen Parametern übergeben werden. Der CVTK Operator benötigt zum Beispiel eine Liste von beliebig vielen Farbbeispielen, deren Einträge je ein Tupel mit vier Einträgen für die Werte des roten, grünen und blauen Kanals und der zugehörigen Farbklasse sind.

Generell sollte der Pool möglichst viele verschiedene, flexible und spezielle Operatoren umfassen und die komplette Bandbreite der verfügbaren Algorithmen abdecken. Für den Anfang wurden fünf verschiedene Operatoren angepasst oder implementiert. Später sollen mehr hinzukommen. Im Folgenden werden die einzelnen Operatoren erklärt und ihre Parameter aufgeführt.

3.2.1 Regionen mit CVTK

Der CVTK Operator hat von den verwendeten Operatoren den mit Abstand größten Parameterraum und muss sehr genau an die Umgebung und die Aufgabenstellung angepasst werden. Normalerweise wird diese Kalibrierung von einem Menschen vorgenommen, der durch zwei mitgelieferte interaktive Tools bei dem relativ komplexen und fehleranfälligen Vorgang unterstützt wird.

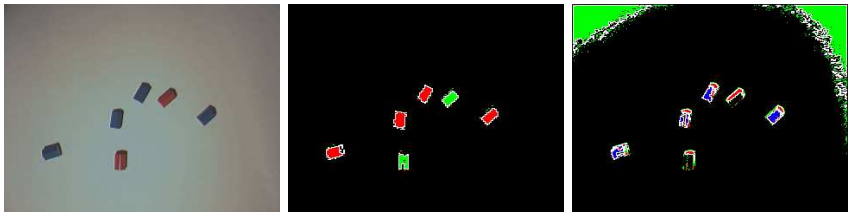

Abbildung 3.4: Originalbild (links) und Ergebnis der Segmentierung mit verschiedenen Parametereinstellungen. Parameter Mitte: Farbbeispiele: (56, 71, 90, 1), (124, 66, 69, 2), (141, 140, 129, 0), (141, 160, 159, 0), Farbraum: UV, Schwellwert: 10. Parameter rechts: Farbbeispiele: (58, 76, 55, 3), (68, 66, 55, 1), (104, 96, 79, 2), (111, 170, 119, 0), Farbraum: RGB, Schwellwert: 40.

Der CVTK Operator wurde aus den low-level Computer Vision Schichten des Comptuer Vision Tool Kits erstellt. Das Tool Kit implementiert eine vollständige Verarbeitungskette, die es ermöglicht, farblich markierte Objekte zu unterscheiden und im zweidimensionalen Raum zu verfolgen (tracken) [23]. Die Verarbeitungskette wurde für die Small-Size League des Robocups entwickelt, bei der eine Kamera über dem Spielfeld angebracht wird. Die Bilder dieser globalen Kamera werden zur Steuerung des gesamten Teams verwendet.

Die für den CVTK Operator verwendeten beiden ersten Schichten der Verarbeitungsschicht nehmen eine Farbsegmentierung des ursprünglichen Bildes vor

Tabelle 3.1: Parameter des CVTK Operators

	Parameter	Typ	Menge	Beschreibung
Algorithmus	max_color	int	1	Anzahl der Farbklassen des Bildvordergrundes. Die Klasse 0 für den Hintergrund kommt noch hinzu
	(r,g,b,c)	int[4]	1..n	Klassifiziertes Farbbeispiel
	color_space	int	1	Auswahl des Farbraumes, der für die Berechnung des Farbabstandes verwendet werden soll (RGB, YUV, UV, CIE L*a*b*)
	t	double	1	Schwellwert für den maximalen Abstand in der Nächster-Nachbar-Klassifizierung
	min_size	double	1	Minimale Fläche der Regionen. Kleinere Flächen werden gelöscht
Ausgabe	n_regions	int	1	Anzahl der pro Farbklasse auszugebenden Regionen
	order	boolean	1	Ab- oder aufsteigende Sortierung der Regionen nach der Fläche
	area	boolean	1	Fläche ausgeben
	position	boolean	1	Koordinaten des Schwerpunkts ausgeben
	color	boolean	1	Farbklasse ausgeben
	compactness	boolean	1	Kompaktheit ausgeben
	bounding_box	boolean	1	Bounding Box ausgeben

und berechnen eine konturbasierte Beschreibung der gefundenen Regionen. Aus dieser Beschreibung können dann einfach die Region beschreibende Deskriptoren wie Fläche, Schwerpunkt, Bounding Rectangle und Kompaktheit berechnet werden. Diese Deskriptoren werden dann als Analyseergebnis ausgegeben, während die Kontur selbst nur der internen Verarbeitung dient.

Die erste Schicht der CVTK Verarbeitungskette nimmt eine einfache, pi-

xelbasierte Farbsegmentierung vor. Dabei wird jedes Pixel entsprechend einer Klassifikationsfunktion f einer Farbklasse zugeordnet, die allein vom Farbwert des zu klassifizierenden Pixels bestimmt wird:

$$f : [0, 255]^3 \mapsto [0, N]$$

wobei N die Anzahl der Farbklassen des Vordergrundes und 0 die Klasse des Hintergrundes ist (Visualisierung des Ergebnisses in Abb. 3.4). Die Umgebung des Pixels bleibt unberücksichtigt. Bei dieser Form der Segmentierung handelt es sich um eine sehr einfache Segmentierungsmethode, die dem globalen Thresholding sehr ähnlich ist.

Die Klassifikationsfunktion wird dem Segmentierungsalgorithmus von außen bereitgestellt. Beim globalen Thresholding wird sie zum Beispiel mittels einer Analyse des Histogramms des zu segmentierenden Bildes gefunden [16]. Im Fall der CVTK Bibliothek wird sie aber in einem "Kalibration" genannten Prozess von einem Menschen bereitgestellt. Um als parametrisierter, automatisch einstellbarer Operator in dieser Arbeit verwendet werden zu können, wird diese zentrale Funktion nun von den Parametern des Operators bestimmt.

Zur Einstellung wird eine beliebig große Menge von Beispielen (Samples) dieser Klassifizierungsfunktion gegeben. Ein Sample s hat dabei die Form $(s_{\text{rgb}}, s_{\text{class}})$ mit $s_{\text{rgb}} \in [0, 255]^3$ und $s_{\text{class}} \in [0, N]$. Für die Segmentierung selber wird nun eine einfache Nächster-Nachbar-Klassifikation angewendet. c_1 und c_2 seien die Komponentenfunktionen, die die beiden Komponenten eines Samples liefern. Es gelte $c_1(\mathbf{s}) = s_{\text{rgb}}$ und $c_2(\mathbf{s}) = s_{\text{class}}$. Das Klassenlabel s_{class} eines jeden Pixels b_{ij} des zu segmentierenden Bildes $\mathcal{B}$ wird bestimmt, indem zuerst das nächstgelegene Sample s_{closest} gefunden wird:

$$s_{\text{closest}} = argmin_s d(c_1(s), b_{ij}) \ .$$

Falls der Abstand $d(c_1(s), b_{ij})$ nicht größer als ein Schwellwert t ist, wird dem Pixel b_{ij} das Klassenlabel $c_2 s_{\text{closest}}$ dieses nächsten Samples zugewiesen. Ist der Abstand größer als der Schwellwert t, so wird dem Pixel die Klasse des Hintergrundes zugewiesen.

Da diese Minimierung des Abstandes für jedes einzelne Pixel jedes Bildes der Trainingsmenge sehr rechenaufwändig wäre, wird die Funktion f im Voraus berechnet. Prinzipiell ist es so, dass der verwendete Farbraum von den Samples und der Abstandsfunktion in die einzelnen Samples umgebende, disjunkte Regionen eingeteilt wird. Alle Punkte innerhalb einer Region sind dabei dem gleichen, in ihrem Zentrum gelegenen Farbsample am nächsten. Pixel mit in der gleichen Region gelegenen Farbwerten erhalten auch das gleiche Farblabel (solange sie nicht weiter als t vom Zentrum entfernt sind). Eine beispielhafte Aufteilung eines zweidimensionalen Raumes in die durch drei Samples und die Abstandsfunktion bestimmte Voronoi-Polytope ist in Abb. 3.5 zu sehen. Sind die Grenzen eines Polytops weiter als vom Schwellwert t zugelassen vom Zentrum entfernt, wird dieser Bereich von der das Zentrum umgebenden Hyperkugel mit dem Radius t abgeschnitten und mit dem Label Null versehen. Die Aufteilung

des gesamten Farbraumes kann z.B. in einer Nachschlagetabelle vorab berechnet und gespeichert werden.

Der Operator kann für die Aufteilung des Farbraumes verschiedene Abstandsoperatoren (Kernel) d verwenden. Die vorhandenen Implementierungen verwenden immer den Euklidschen Abstand, transformieren die RGB-Farbwerte aber vorher in andere Farbräume. Es sind per Parameter auswählbare Kernel für den RGB, YUV [14], UV und CIE 1976 L*a*b* Farbraum vorhanden, wobei der UV Farbraum eine einfache Projektion des YUV Farbraums auf eine UV Hyperebene (mit $Y = c$) darstellt, bei der alle Helligkeitsinformation verloren geht.

Das durch die Segmentierung entstandene Regionenbild, bei dem jeder Eintrag aus einer natürlichen Zahl, die die Farbklasse benennt, besteht, wird anschließend von der zweiten Schicht der CVTK-Verarbeitungskette verarbeitet. Hier wird das gesamte Bild noch einmal durchlaufen und die Konturen der verschiedenfarbigen Regionen abgeschritten. Eine Region sind hierbei alle Pixel mit dem gleichen Farblabel, die im Sinne der 4er- oder 8er-Nachbarschaft zusammenhängen. Die abgeschrittenen Konturen werden als Kettenkodierungen nach Freeman [15] abgespeichert und für die Weiterverarbeitung verwendet. Hierbei werden nur Regionen berücksichtigt, deren Fläche (Anzahl der zugehörigen Pi-

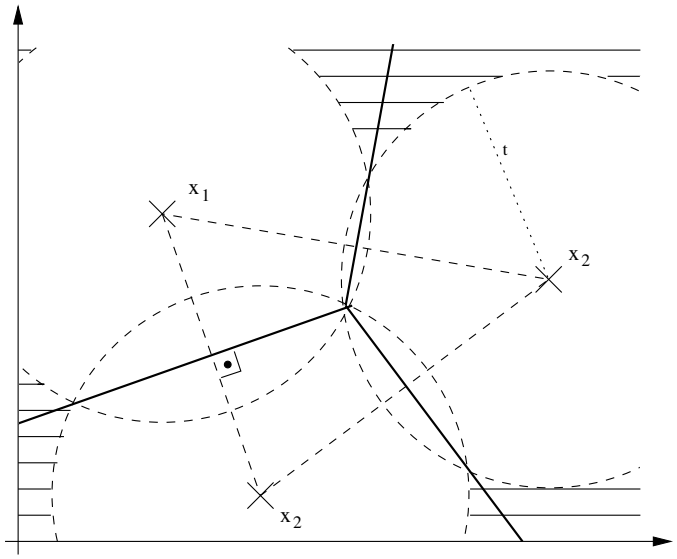

Abbildung 3.5: Aufteilung eines zweidimensionalen Farbraumes in um die projizierten Farbbeispiele x_1, x_2 und x_3 gelegene Voronoi-Polytope (begrentzt von durchgezogenen, dicken Linien). Punkte eines Voronoi-Polytops, die weiter als der Schwellwert t vom Zentrum entfernt sind, werden dem Hintergrund (waagerecht schraffiert) zugeordnet.

xel) größer als ein einzustellender Minimalwert ist. Das ikonische Regionenbild selbst wird nach der Kodierung in die kompakteren Kettenkodes nicht mehr benötigt und gelöscht.

Aus den Kettenkodierungen der einzelnen Regionen können nun die benötigten Merkmale (Koordinaten des Schwerpunkts, Kompaktheit, Fläche, etc.) berechnet werden. Welche Merkmale tatsächlich berechnet werden, wird von den aktuellen Werten der Ausgabeparameter des Operators bestimmt. Eine Auflistung aller Parameter ist in Tabelle 3.1 zu finden.

3.2.2 Histogramme

Die Idee für die Verwendung eines Histogramme berechnenden Operators an dieser Stelle stammt aus einem Artikel von Luc Steels und Frederic Kaplan [40, S. 16f]. In dem Artikel beschreiben Steels und Kaplan die Durchführung verschiedener Experimente mit einem modifizierten AIBO Roboter, der das "Kategorisierungsspiel", eine Abart des "language games" [39], spielen muss. Dort werden zweidimensionale Histogramme der nach der Helligkeit (R+G+B) normalisierten Farbwerte als einzige Repräsentation der Bilder in einem assoziativen Speicher (Bild + Wort) abgelegt und mittels eines einfachen, instanzbasierten Verfahrens Kategorien gebildet. Die Ergebnisse belegen, dass die Aufnahmen von drei verschiedenen Objekten alleine an Hand dieser Histogramme unterschieden, bzw. kategorisiert werden können.

Hier in dieser Arbeit wird ein ähnlicher Operator mit 3 Parametern (siehe Tabelle 3.2). verwendet. Und zwar bildet der Operator ein zweidimensionales Histogramm aus den RGB-Farbwerten, indem die Farben zuerst in den YUV Farbraum transformiert und dann orthogonal auf die UV Ebene projiziert (einfache Entfernung des Helligkeitswertes) werden. Die Granularität, also die Anzahl der verschiedenen UV Werte, die auf den gleichen Histogrammeintrag fallen, kann durch einen Parameter bestimmt werden und reicht von 256×256 bis hin zu 2×2 Histogrammeinträgen.

Ein solches globales Histogramm vom ganzen Bild enthält keinerlei Information mehr über die Position einzelner Objekte. Wird ein einzelnes Objekt vor einem uniformen Hintergrund verschoben, ändert sich das Histogramm nicht. Um das mögliche Einsatzgebiet etwas breiter zu fassen, wurde der Operator um die Möglichkeit erweitert, das Bild in mehrere disjunkte Regionen einzuteilen, von denen getrennte Histogramme bestimmt werden. Durch einen Parameter kann bestimmt werden, in wie viele Bereiche die x und die y Bildachsen eingeteilt werden sollen. Die Einteilung geht von 1 bis 6 Bereiche je Bildachse, was maximal $6^2 = 36$ separat zu untersuchende Bildbereiche bedeutet. Die 6^2 resultierenden Histogramme werden einfach zeilenweise Eintrag nach Eintrag aneinandergehängt und ausgegeben.

Da bei der Verwendung einer feinen Granularität und einer hohen Anzahl von Bildbereichen ein recht langer Vektor (mit deutlich mehr als tausend Einträgen) entstehen kann, in der Regel aber nur einige wenige Einträge im Histogramm von Interesse sind (z.B. das Verhältnis zwischen Kaminrot und Himmelblau), wurde die Möglichkeit geschaffen, einzelne Einträge des Histogramms für die

Tabelle 3.2: Parameter des Histogramm-Operators

	Parameter	Typ	Menge	Beschreibung
Algorithmus	windows	int	1	Anzahl der Bereiche auf jeder Bildachse. Es werden windows^2 Histogramme disjunkter Bildregionen berechnet
	quantization	int	1	Spezifiziert die Granularität der Histogramme. Die UV Koordinaten werden um diesen Wert bitweise nach rechts verschoben. Die Histogramme haben dann $n = (256/2^{\text{quantization}})^2$ Einträge
Ausgabe	print_not	int	1..n	Gibt je einen Histogrammeintrag an, der nicht ausgegeben werden soll. Dabei werden die Histogrammeinträge spalten- und zeilenweise durchnummeriert

Ausgabe zu "löschen". Es können per Parameter beliebig viele Einträge aus dem Ausgabevektor entfernt werden. Diese Angaben gelten immer einheitlich für die Histogramme aller Bildbereiche.

3.2.3 Regionen mit CSC

Der von Priese und Rehrmann in [32] beschriebene "Color Structure Code" (kurz: CSC), ist eine hierarchische Region-Growing-Methode. Bei dieser Art von Segmentierungsmethoden werden ausgehend von sogenannten Saatpunkten Regionen iterativ vergrößert, indem zu einer bestehenden Region benachbarte Punkte hinzugefügt werden, wenn sie einen festgelegten Grad an Ähnlichkeit zu bestehenden Regionen besitzen. Oftmals wird zusätzlich noch ein Homogenitätskriterium für die Regionen verwendet, das auch noch von der vergrößerten Region erfüllt sein muss [38, S. 176ff]. Beim CSC geschieht die Entwicklung der verschiedenen Regionen parallel. Das Segmentierungsergebnis ist im Gegensatz zu vielen der anderen, schnellen Region-Growing-Methoden unabhängig von der, z.B. durch Saatpunkte bestimmten, Verarbeitungsreihenfolge.

Der CSC läßt die Regionen auf einer speziellen hexagonalen Topology anwachsen. Die Struktur besteht aus sogenannten Inseln verschiedener Hierarchieebenen. Die Inseln der untersten Ebene bestehen aus einem zentralen Pixel und

Abbildung 3.6: Originalbild (links) und Ergebnis der Segmentierung mit dem CSC Operator mit verschiedenen Werten für den Schwellwert t des Farbabstandes. Schwarze Punkte wurden keiner Region zugeordnet. Mitte: $t = 10$, rechts: $t = 40$.

den sechs in der hexagonalen Topologie benachbarten Pixeln. Dabei überlappen sich benachbarte Inseln immer in genau einem Pixel; die Randpixel sind folglich immer in genau zwei Inseln enthalten. Die Inseln der nächsthöheren Ebene $h + 1$ werden in gleicher Weise aus der Inselstruktur der Ebene h gebildet (eine zentrale Insel und die sechs Nachbarn dieser Insel der Ebene h). Da Bilder in der Regel in einem orthogonalen Gitter aufgezeichnet werden, wird die hexagonale Struktur nur als logische Struktur verwendet. Die Hexagone der Ebene 0 werden gebildet, indem jede zweite Zeile der logischen hexagonalen Struktur um ein halbes Pixel nach links verschoben wird.

Auf der so entstandenen hierarchischen Struktur fällt die Verschmelzung benachbarter Regionen wesentlich leichter. Der Algorithmus besteht hierbei aus drei Phasen.

In der Startphase (engl: initialization phase) wird das Bild in kleine, zusammenhängende Regionen innerhalb der Inseln der Ebene 0 aufgeteilt. Dazu wird jede einzelne Insel getrennt von den anderen daraufhin untersucht, welche der sieben Pixel benachbart und ähnlich zueinander sind und somit zu einer Region (beschrieben durch sogenannte Codeelemente) verschmolzen werden können.

In der Verknüpfungsphase (engl.: linking phase) werden die Inseln der Ebene $h+1$ daraufhin untersucht, welche der Codeelemente ihrer sieben Subinseln der Ebene h verschmolzen werden können. Es werden Codeelemente verschmolzen, die miteinander verbunden sind und deren Durchschnittsfarben sich ausreichend ähneln. Die Überprüfung, ob zwei Codeelemente benachbarter Subinseln miteinander verbunden sind, ist wegen der verwendeten, besonderen Struktur recht einfach: Sie müssen ein gemeinsames Subelement haben (jedes Subelement, das kein zentrales Element ist, ist Element zweier Inseln der nächst höheren Ebene). Dieser Verknüpfungsschritt wird für jede Ebene bis zur höchsten Ebene, die aus einer einzigen Insel besteht, wiederholt. Dabei werden für jedes neue Codeelement der Ebene h Zeiger zu den zugehörigen Subcodeelementen der Ebene $h - 1$ gespeichert. Die Subcodeelemente erhalten zur Vereinfachung späterer Berechnungen ebenfalls einen Zeiger zu dem übergeordneten Codeelement.

Aufgrund der überlappenden Inselstruktur können Codeelemente jeweils bis zu zwei "Elternzeiger" besitzen. Diese Tatsache kann für eine Beschleunigung

Tabelle 3.3: Parameter des CSC Operators

	Parameter	Typ	Menge	Beschreibung
Algorithmus	t	double	1	Schwellwert für den Farbabstand in der Verknüpfungsphase
Ausgabe	n_regions	int	1	Anzahl der auszugebenden Regionen
	level	boolean	1	Die Verknüpfungsebene der Region ausgeben
	island_coords	boolean	1	Koordinaten der zugehörigen Insel ausgeben
	feature	boolean	1	Durchschnittsfarbe der Region ausgeben
	bounding_box	boolean	1	Bounding Box ausgeben
	real_coords	boolean	1	Koordinaten des Schwerpunkts ausgeben

der Verknüpfungsoperation verwendet werden: Anstatt überlappende Subelemente zu suchen, werden einfach die Codeelemente der Ebene $h-1$ daraufhin untersucht, ob sie zwei Elternzeiger besitzen. In einem solchen Fall sind die Eltern der Ebene h miteinander verbunden.

Die nach Abschluß des Verknüpfungsschrittes gefundenen Regionen können unter Umständen inhomogen sein. Da immer nur die aktuelle Durchschnittsfarbe der gerade zu verschmelzenden Regionen betrachtet wird, kann es passieren, dass sich die Durchschnittsfarbe kontinuierlich in eine Richtung verschiebt und sich weit von der Farbe der einzelnen Subregionen der untersten Ebenen entfernt. Gerade wenn eine Kette von Pixeln, deren Farbe sich von Nachbar zu Nachbar nur leicht ändert, zwischen zwei Farbextremen existiert, kann am Ende eine Region einzelne Pixel mit sehr unterschiedlichen Farben beinhalten. Daher wird in der Aufteilungsphase (engl.: splitting phase) neben der bei der Verknüpfung berücksichtigten lokalen Information ein globales Homogenitätskriterium überprüft. Und zwar werden alle Codeelemente einer verknüpften Region auf ihre Farbabstände zueinander untersucht. Ist der Abstand größer als ein Schwellwert, wird die Region wieder aufgeteilt.

Zur Berechnung der Farbähnlichkeiten wird beim CSC der HSV Raum verwendet. Zur Vermeidung von Problemen bei der Berechnung der Farbähnlichkeit verwenden Priese et al. sich mit der Position des Farbwertes im Farbraum ändernde Schwellwerte für den erlaubten Abstand. Die Probleme entstehen wegen der starken Schwankungen des Wertes für H nahe der Singularität der V-Achse, wo die Sättigung gleich null ist. Die genauen Schwellwerte wurden experimentell bestimmt und in einer Nachschlagetabelle gespeichert.

Für Bilder mit einer Größe von weniger als 512×512 Pixeln sind laut Priese

ungefähr $4 \cdot N$ Farbähnlichkeitsvergleiche nötig, wobei N die Anzahl der Pixel sei. Die Anzahl der Überprüfungen, ob zwei Regionen miteinander verbunden sind, sei ebenfalls linear zur Anzahl der Pixel. Die Anzahl der Aufteilungsvorgänge ist hingegen vom Bildinhalt und der damit verbundenen Verknüpfungshöhe und der Anzahl der Regionen abhängig. Allerdings macht diese Phase weniger als 5% der Rechenzeit aus und hat daher keinen entscheidenden Einfluss auf das ansonsten zur Bildgröße lineare Laufzeitverhalten.

Nach eigenen Messungen ist der CSC Algorithmus zumindest für Bilder der Standardgröße 320×240 Pixel auf aktueller PC-Hardware echtzeitfähig und damit für die Verwendung in dieser Arbeit geeignet.

Der wichtigste Parameter für die in dieser Arbeit verwendete Implementierung des CSC-Algorithmus ist der Schwellwert für den maximal erlaubten Farbabstand zweier in der Verknüpfungsphase verschmolzenen Regionen. Alle anderen Parameter des CSC-Operators beeinflussen lediglich, wie und mit welchen Eigenschaften die gefundenen Regionen für den Ausgabevektor kodiert werden sollen (siehe Tabelle 3.3).

3.2.4 Ebene der Gaußpyramide

Dieser Operator mit nur einem Parameter reduziert die Auflösung des Originalbildes. Eine solche Reduzierung kann nicht einfach durch die Abtastung jedes iten Pixels geschehen, da dadurch ein fehlerhaftes Muster entstehen würde [19, S. 143 f, S. 227 ff]. Stattdessen muss die Reduktion mit einer Glättung kombiniert werden.

Wenn man ein Originalbild in dieser Weise mehrmals hintereinander mit der Abtastrate 2 (Breite und Höhe werden halbiert) bearbeitet, erhält man eine Bildserie von kleiner werdenden Bildern, die Gaußpyramide genannt wird [19, S. 143 f].

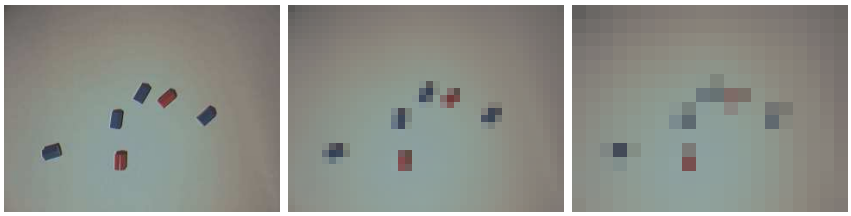

Abbildung 3.7: Originalbild (links) und die Ebenen drei (mitte) und vier (rechts) der zugehörigen Gaußpyramide.

Genau diese Pyramide kann von dem Operator berechnet werden. Mittels eines Parameters wird eingestellt, welche Ebene q ausgegeben werden soll. Das Bild der Ebene $q = 0$ der Gaußpyramide ist das Originalbild. Für die Ausgabe werden die Zeilen der resultierenden Bildmatrix aneinander gehängt, so dass ein Vektor mit der Länge $\frac{b}{2^q} \cdot \frac{h}{2^q} 3$ zurückgegeben werden kann. b sei hierbei die Breite und h die Höhe des dreikanaligen Originalbildes.

Tabelle 3.4: Parameter des Operators zur Berechnung einer Ebene der Gaußpyramide

	Parameter	Typ	Menge	Beschreibung
Algorithmus & Ausgabe	level	int	1	Auszugebende Ebene der Gaußpyramide. Die Dimension des Bildes dieser Ebene beträgt $\frac{b}{2^q} \times \frac{h}{2^q}$ Pixel, wobei h und b die Höhe und Breite des Originalbildes seien. Das Level muss hierbei größer als zwei sein

Um eine minimale Reduktion der Dimension zu erreichen, sind die Auswahl der ersten beiden Ebenen und des Originalbildes durch den Parameter ausgeschlossen; es können nur Ebenen ab der dritten Ebene ($q > 2$) ausgegeben werden.

3.2.5 Featureextraktion mit SUSAN

Der SUSAN-Operator sucht zweidimensionale Features im Bild und gibt deren Position aus. Der Operator findet nicht nur Ecken, sondern auch normale und T-Kreuzungen. Kreuzungen entstehen im Bild zum Beispiel, wenn mehrere Flächen aneinanderstoßen. Das Besondere an dem SUSAN-Operator ist, dass er zur Berechnung dieser auf dem Verlauf von Kanten basierenden Features keinen Gradienten berechnet und die mit dieser Methode verbundenen Probleme umgeht [37].

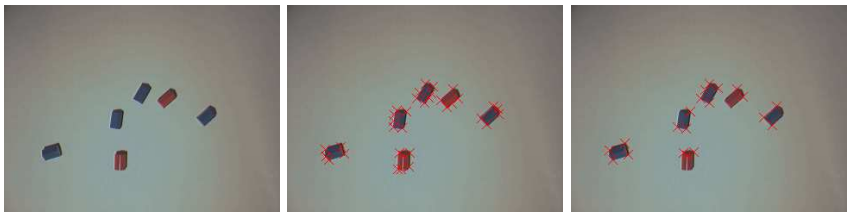

Abbildung 3.8: Originalbild (links) und Ergebnisse des SUSAN Eckendetektors für zwei verschiedene Schwellwerte t für den maximal erlaubten Farbabstand "ähnlicher" Pixel und unterschiedliche Kanäle. Die gefundenen Ecken sind im Bild mit Kreuzen markiert. Mitte: $t = 14$, Helligkeit. Rechts: $t = 50$, roter Kanal.

Der Algorithmus besteht aus zwei Phasen: Zuerst wird ein spezieller Kernel, der USAN (Univalue Segment Assimilating Nucleus) für jedes Pixel angewen-

det. Anschließend erfolgt eine Non-Maxima-Suppression (Unterdrückung aller Werte, bis auf lokale Maxima). Das Ergebnis ist in Abb. 3.8 zu sehen.

Zur Berechnung des Wertes des USAN für ein Pixel wird eine radiale Umgebung herangezogen. Es werden in dieser Umgebung die zusammenhängenden Pixel gezählt, die die gleiche, bzw. eine ähnliche Farbe wie das Zentrum besitzen. Befindet sich das Zentrum des USAN in der Nähe einer Kante einer Region, ist die Anzahl kleiner, als wenn sich der Nukleus vollständig innerhalb einer Region befinden würde. Befindet sich das Zentrum nun am äußersten Punkt einer Ecke der Region, dann ist die Anzahl der zusammenhängenden Pixel im spitzeren Winkel dieser Ecke auf jeden Fall kleiner als die Hälfte aller Punkte in der kreisförmigen Maske. Dieses Kriterium kann direkt zum Auffinden von Ecken herangezogen werden. Hierzu wird die Anzahl der gefundenen Pixel von einem geometrischen Schwellwert (z.B. die Hälfte der vom Kernel abgedeckten Pixel) abgezogen und als initialer "Eckenausschlag" verwendet. Wird eine potentielle Ecke mit einem positiven Ausschlag gefunden, werden noch zwei weitere Tests durchgeführt, um falsch positive Kandidaten auszuschließen. Und zwar wird der Schwerpunkt der zusammenhängenden Fläche berechnet. Liegt der Schwerpunkt nahe am Zentrum des Kernels, handelt es sich nicht um die Ecke einer Region (dann wäre der Schwerpunkt nicht nahe am Zentrum), sondern z.B. um eine dünne Linie durch den Kernel (hier ist die einfarbige, zusammenhängende Fläche auch klein). Sollte dieser Test bestanden werden, wird die Verbindungslinie zwischen Kernelzentrum und Schwerpunkt daraufhin untersucht, ob alle auf ihr liegenden Bildpunkte eine ähnliche Farbe wie das Zentrum haben und somit ebenfalls zur Fläche gehören. Nur wenn auch dies der Fall ist, handelt es sich bei dem untersuchten Punkt um eine potentielle Ecke. Wird eines dieser Kriterien nicht erfüllt, wird die Ausgabe des Operators für dieses Pixel auf 0 gesetzt.

Interessant ist hierbei, dass nicht der Gradient der Farbänderung, sondern die zusammenhängende Fläche selber zur Detektion herangezogen wird. Es hat sich erwiesen, dass dieses Verfahren sehr stabil gegenüber Rauschen ist und keine Glättung vor der Anwendung des Kernels erfordert. Auch deshalb können die Ecken und Schnittpunkte wesentlich genauer als mit anderen Methoden, die eine Glättung erfordern, lokalisiert werden.

Anschließend werden in einem 5×5 Pixel großen Fenster die maximalen Ausschläge gesucht (Non-Maxima Suppression), die dann vom SUSAN Edge Detector als Ecken ausgegeben werden.

Der Algorithmus besitzt im wesentlichen zwei Parameter: Den geometrischen Schwellwert g und den Schwellwert t für den erlaubten Farbabstand ähnlicher Pixel. Der Wert g beeinflusst, was für Ecken gefunden werden. Je niedriger man ihn setzt, desto spitzer sind die gefundenen Ecken. Mit t kann bestimmt werden, wie stark die Farbunterschiede (Schattierung, Rauschen) sein dürfen, um noch zur gleichen Fläche gehören zu können. Da der SUSAN-Algorithmus für monochromatische Bilder implementiert ist, kann mit einem weiteren Parameter eingestellt werden, in welchem Farbkanal des Bildes nach den Ecken gesucht werden soll (Rot, Grün, Blau und Grauwert).

Neben diesen beiden den Algorithmus beeinflussenden Parametern gibt es

Tabelle 3.5: Parameter des SUSAN Ecken-Operators

	Parameter	Typ	Menge	Beschreibung
Algorithmus	t	double	1	Schwellwert für den Farbabstand
	g	double	1	Geometrischer Schwellwert, je kleiner, desto spitzere Ecken
	channel	int	1	zu untersuchender Kanal (R, G, B oder Y)
Ausgabe	n_features	int	1	Anzahl der auszugebenden Ecken
	position	boolean	1	Koordinaten des Features ausgeben
	intensity	boolean	1	Intensität (Auschlag des Operators) ausgeben
	gradient	boolean	1	Gradienten berechnen und ausgeben
	sort_criterion	int	1	Sortierung nach Intensität oder dem Produkt der Gradienten, auf- oder absteigend

für den implementierten SUSAN-Operator noch einige Parameter zur Beeinflussung des Ausgabevektors. Es kann festgelegt werden, welche der berechenbaren Eigenschaften für jede gefundene Ecke ausgegeben werden sollen. Die Anzahl der maximal auszugebenden Ecken kann ebenso geändert werden wie das Kriterium, nach dem die Ecken für die Ausgabe sortiert werden. Eine vollständige Übersicht findet sich in Tab. 3.5.

Kapitel 4

Experimente

4.1 Problemstellungen

Der Algorithmus wurde an Hand von verschiedenen Experimenten in fünf verschiedenen Problemstellungen mit unterschiedlicher Schwierigkeit getestet. Dabei handelt es sich sowohl um Klassifikationsaufgaben, die sich besonders gut als Benchmark eignen, als auch um Navigationsaufgaben. Bevor die einzelnen Experimente und die Ergebnisse beschrieben werden, werden hier die verschiedenen Problemstellungen erläutert.

In allen durchgeführten Experimenten wurden ausschließlich echte, unbearbeitete Bilder verwendet. Es wurden keine Tests mit künstlichen Bildern durchgeführt. Außerdem wurden keine besonderen Anstrengungen unternommen, für eine besonders starke, gleichmäßige oder schattenfreie Beleuchtung zu sorgen.

Für jede Aufgabe werden mindestens zwei Patternmengen bereitgestellt. Dabei befindet sich kein Bild in beiden Mengen. Eine Menge wird zum Training des neuronalen Netzes verwendet (Trainingsmenge), die andere zum Test der Generalisierungsfähigkeit nach dem Austrainieren des neuronalen Netzes (Testmenge).

Im Falle der Klassifikationsaufgaben wird noch eine dritte Menge (zweite Testmenge) bereitgestellt, mit der die Generalisierungsfähigkeit des besten Kandidaten nach Abschluss der Evolution getestet werden kann. Diese Bilder wurden dem Algorithmus in keiner Phase der Evolution gezeigt. Im Falle der Navigationsaufgaben wird das beste Individuum nach Abschluß der Evolution direkt auf dem Roboter getestet.

4.1.1 Post-its zählen

Bei der einfachsten untersuchten Aufgabe wurde eine Bildserie von gelben Post-it Aufklebern vor weißem Hintergrund angefertigt. Im Bild befanden sich kein, ein oder zwei Post-its. Die Post-its überlappen oder berühren sich in keinem der Bilder. In der Testmenge befinden sich auch Bilder von zwei Teilen eines zerrissenen Post-its (sihe Abb. 4.1).

Abbildung 4.1: Auszug aus der Bildmenge im Experiment 4.1.1.

Die Aufgabe besteht darin, die Bilder entsprechend der Anzahl der abgebildeten Post-its richtig zu klassifizieren (0, 1, 2). Die Bilder haben eine Größe von $x \times y$ Pixel. Im Trainingsset befinden sich 20, im Testset 12 und im zweiten Testset 8 Aufnahmen.

4.1.2 Würfel ablesen

Die nächstschwierigere Aufgabe befasst sich mit dem Ablesen der Augenzahl eines Würfels. Die Trainingsmenge besteht aus 46 Aufnahmen eines weißen Würfels mit schwarzen Augen. Der Würfel ist auf allen Bildern in ungefähr dem gleichen Abstand von der Kamera zu sehen. In der Testmenge für die innere Evaluationsschleife befinden sich 12 andere Bilder desselben Würfels. In der zweiten Testmenge sind neben 11 Bildern vom gleichen Würfel auch 4 Bilder von einem anderen, größeren Würfel enthalten (siehe Abb. 4.2).

Abbildung 4.2: Auszug aus der Bildmenge im Experiment 4.1.2. Bilder vom letzten, größeren Würfel sind nur in der zweiten Testmenge enthalten.

4.1.3 Subtraktion von Spielsteinen

Diese Aufgabe ähnelt der Zählen-Aufgabe, erfordert aber die Detektion und Unterscheidung von zwei Objektklassen. Im Bild befinden sich null bis acht blaue und rote Spielsteine. Dabei sind von jeder Farbe maximal vier Steine und immer mindestens so viele blaue wie rote Spielsteine im Bild. Die Aufgabe besteht darin, die Differenz zwischen blauen und roten Steinen zu bilden. Mit den oben genannten Randbedingungen bewegt sich die auszugebende Zahl im Bereich zwischen Null und Vier.

In der Trainingsmenge befinden sich 98 Bilder, in der ersten Testmenge 27 und in der zweiten Testmenge 9 Bilder.

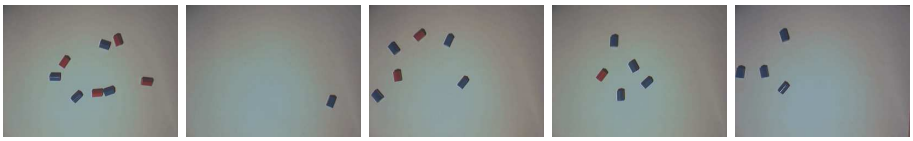

Abbildung 4.3: Auszug aus der Bildmenge im Experiment 4.1.3.

4.1.4 Spielfiguren klassifizieren

Bei diesem Experiment müssen Aufnahmen von Spielzeugfiguren in drei verschiedene Klassen eingeteilt werden. Zu diesem Zweck wurden 66 Makroaufnahmen von je fünf verschiedenen Figuren von drei verschiedenen Figurtypen aus verschiedenen Winkeln angefertigt. Um die Aufgabe zu erschweren, wurden die Figuren vor dem gleichen Hintergrund mit einer festen Kameraposition aufgenommen.

Bei den Figuren handelt es sich um aktuelle Überraschungseifiguren, Warhammer Menschen und Warhammer Orks. Die Aufnahmen sollen nun analog in drei Klassen (1 = Überraschungsei, 2 = Mensch, 3 = Ork) einsortiert werden. Hierbei wird die Aufgabe dadurch weiter erschwert, dass sich die Figuren aus den Überraschungseiern zum Teil sehr stark unterscheiden und so eine inhomogene Klasse bilden.

In Abb. 4.4 sind alle Figuren der drei Klassen aus einem Drehwinkel zu sehen. Je Klasse wurde eine Figur ausgewählt, deren Bilder nicht in die Trainingsmenge, sondern nur in die Testmengen aufgenommen wurden. Die Trainingsmenge beinhaltet 32 , die Testmenge 26 und die zweite Testmenge 8 Bilder.

4.1.5 Roboter zum Ball steuern mittels gerichteter Kamera

In diesem Experiment besteht die Aufgabe darin, einen Roboter mit omnidirektionalem Antrieb zu einem Ball zu fahren. Es sind für jedes von einer am Roboter befestigten, nach vorne gerichteten Kamera aufgenommenes Bild passende Steuersignale zu generieren. Ein Steuersignal besteht hierbei aus einem Richtungsvektor im egozentrischen Koordinatensystem des Roboters und einem Wert für die Drehgeschwindigkeit um die eigene Achse.

Mittels einer Joysticksteuerung wird der Roboter mehrere Male von verschiedenen Positionen zum Ball gefahren. Dabei werden Bilder mit einer Frequenz von 7,5 Hz aufgezeichnet und zusammen mit dem über den Joystick eingegebenen Steuerbefehl abgespeichert. Von je 10 Bildern gelangen 2 in das erste und 2 in das zweite Testset. Die restlichen sechs Bilder werden in der Trainingsmenge gespeichert.

Bei der Durchführung wurde darauf geachtet, dass sich der Ball möglichst immer im Sichtfeld der Kamera befunden hat. Die Performance wird nur qualitativ verifiziert, indem das gelernte Steuerprogramm auf den Roboter selber

Abbildung 4.4: Auszug aus der Bildmenge im Experiment 4.1.4. In jeder Reihe sind fünf verschiedene Figuren einer Klasse abgebildet. Von der fünften, abgesetzten Figur sind keine Bilder in der Trainingsmenge enthalten.

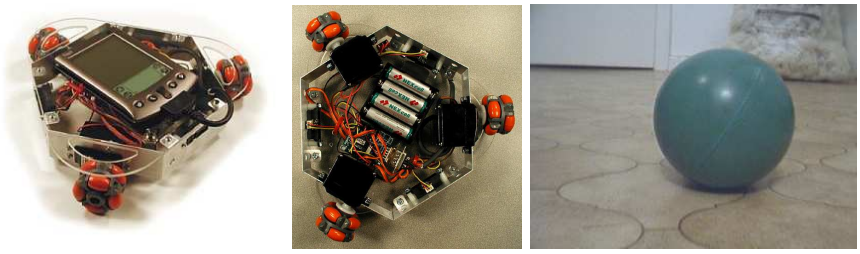

Abbildung 4.5: Links: Aconames PPRK. Mitte: Unterseite des PPRK mit omnidirektionalen Antrieb. Rechts: Bild des grünen Balles aus Sicht des mit einer Kamera anstelle des PalmPilot ausgestatteten PPRK. (Produktbilder von Acroname Inc., Boulder, Colorado, USA)

aufgespielt und getestet wird.

Bei dem eingesetzten Roboter handelt es sich um ein PalmPilot Robot Kit (PPRK), das von der CMU entwickelt und von Acroname Inc. vertrieben wird (siehe Abb. 4.5). Das PPRK ist mit einer einfachen IEEE1394 Digital Camera (nach dem IIDC Standard, siehe [1]) versehen worden. Gesteuert wird der Roboter von einem stationären Computer, weil er selbst keinen leistungsfähigen Microcontroller mitführt. Der Roboter zieht daher ein Kamerakabel und ein Kabel für die Kommunikation über die serielle Schnittstelle hinter sich her.

4.1.6 Navigationsaufgaben mittels omnidirektionaler Kamera

Weitere Navigationsexperimente wurden auf dem "Brainstormers Tribot" (siehe Abb. 4.6, rechts) durchgeführt, wobei die Version verwendet wurde, die bei den RoboCup German Open 2004 [35] erfolgreich war. Dieser Roboter ist wesentlich leistungsfähiger als der kleine PPRK. Er führt einen eigenen, vollwertigen PC in Form eines Subnotebooks mit sich und besitzt einen deutlich stärkeren omnidirektionalen Antrieb, der Geschwindigkeiten bis zu $1,5m/s$ ermöglicht [2,6].

Anstelle einer gerichteten Kamera besitzt der Tribot einen "omnidirektionalen" Kameraaufbau: Die Kamera ist hierbei senkrecht nach oben so auf einen hyperbolischen Spiegel gerichtet, dass im Spiegel die gesamte Umgebung wahrgenommen werden kann (siehe Abb. 4.6, links). In einem solchen Spiegelbild treten nicht-triviale, nicht-lineare Verzerrungen auf, die sich zwar theoretisch wie bei einer gerichteten Kamera herausrechnen (kalibrieren) lassen [17], in der Praxis wegen der ungenauen und äußerst instabilen Positionierung der Kamera und des Spiegels auf dem Tribot aber nur näherungsweise bestimmt werden [2]. Die Berechnung eines Steuerbefehls aus einem solchen Bild stellt somit auch für "triviale Orientierungsaufgaben" ein echtes Problem dar. Daher ist der Nachweis, dass der hier vorgestellte Algorithmus fähig ist, auch für diesen Aufbau

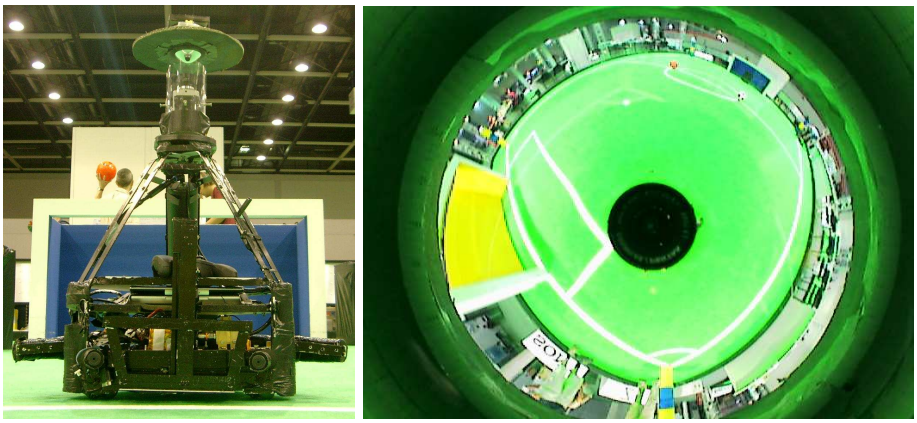

Abbildung 4.6: Links: Brainstormers Tribot in der Version vom Robocup in Padua 2003 Rechts: Von der Kamera des Roboters aufgenommenes Bild einer Spielsituation.

funktionierende Kontroller einzulernen, besonders interessant.

Die Experimente wurden auf einem kleinen "Spielfeld" mit einem gelben Tor durchgeführt. Außerdem befand sich immer ein Spielball auf dem Feld.

Auf dem Tribot wurden mittels der vom Acroname Roboter bekannten Joysticksteuerung drei kleine Aufgaben eingelernt und die resultierenden Kontroller untersucht. Die Aufgabenstellungen wurden so gewählt, dass immer mindestens eine der markanten, in jedem Bild vorhandenen "Landmarken"keine Rolle spielt und nicht mit den Steuersignalen korreliert. Es gilt nachzuweisen, dass eine solche Landmarke trotz ihrer Auffälligkeit von den resultierenden Kontrollern ignoriert und nicht verarbeitet wird.

Zum Ball ausrichten

Ziel ist es, den an einer Seite des Roboters angebrachten Kickmechanismus zum Ball auszurichten und dann Bewegungen des Balles fortlaufend zu verfolgen. Hierbei darf der Roboter sich ausschließlich drehen und möglichst nicht fahren; das Zentrum des Roboters sollte immer auf der gleichen Position verbleiben.

Zum Tor fahren

Der Roboter soll von einer beliebigen Spielfeldposition zum gelben Tor fahren. Die Ausrichtung des Roboters spielt hierbei keine Rolle.

Zum Ball fahren

Der Roboter soll bis in Kickerreichweite zum Ball fahren. Dabei muss der Kicker am Ende zum Ball hin ausgerichtet sein. Damit ist die Aufgabe komplexer als

die “Fahre zum Tor” Aufgabenstellung.

4.2 Ergebnisse

4.2.1 Parametereinstellungen

Bei allen Experimenten, deren Ergebnisse hier präsentiert werden, wurden die Evolutionsparameter gleich eingestellt. Eine Population besteht jeweils aus 60 Individuen. Für die beiden Parameter μ und λ wurde $\mu = 20$ und $\lambda = 15$ gewählt. Die Populationsgröße wurde im Verhältnis zum gewählten Wert von μ eher klein gewählt (Standard wäre bei $\mu = 20$ eine Population von 140 oder mehr Individuen), weil gerade die Auswertung der zufällig initialisierten und wenig optimierten Individuen zu Beginn verhältnismäßig lange dauert. Bei den untersuchten Aufgaben reicht diese kleine Population aber aus, um zuverlässig zu guten Ergebnissen zu kommen. Sollte die Evolution bei einer Aufgabe konsistent in lokalen Maxima stecken bleiben, könnte eine Erhöhung der Anzahl der Individuen helfen.

Die Evolution wurde jeweils für 200 - 500 Epochen durchgeführt. Das neuronale Netz wurde in der inneren Schleife für jeweils 500 Epochen trainiert, unabhängig davon, wie groß die Trainingsmengen waren. Hier könnte man später vielleicht noch ein Abbruchkriterium setzen, für den Fall, dass eine für den Fehler vorgegebene Schranke sehr schnell unterschritten wird oder sich über viele Epochen hinweg keine Verbesserung des Fehlers beobachten lässt. Dadurch könnte vermutlich noch eine beachtliche Menge an Rechenzeit eingespart werden.

Die Anzahl der Operatoren in einer einzelnen Vorverarbeitungsschicht wurde aus Gründen der verfügbaren Rechenzeit auf zehn beschränkt. Setzt man keine solche Obergrenze, werden die Vorverarbeitungsschichten gerade zu Beginn der Evolution übermäßig groß, bevor die kleineren Schichten anfangen, sich aufgrund der besseren Generalisierungsfähigkeit durchzusetzen. Die gesetzte Obergrenze ist aber mit Sicherheit größer, als für ein optimales Lösen der hier gestellten Aufgaben nötig ist, so dass noch genug Spielraum für die evolutionäre Optimierung der Operatorenanzahl vorhanden bleibt.

4.2.2 Optimierung der Parameter eines CVTK Operators

Zum Test des Algorithmus und um einen ersten, praktischen Beleg für die Leistungsfähigkeit zu erbringen, wurden zuerst Experimente durchgeführt, in denen die Parameter eines einzelnen CVTK Operators evolutionär bestimmt wurden. Zur Erinnerung: Dieser Operator besitzt den mit Abstand größten Parameterraum und muss ansonsten mit der Hilfe eines grafischen Tools für jede Aufgabe und auf veränderte Umgebungsbedingungen von Hand neu eingestellt werden.

In der so durchgeführten Evolution besteht die Vorverarbeitungsschicht aller Individuen aus nur einem einzigen CVTK Operator. Damit unterscheiden sich die Vorverarbeitungschichten der einzelnen Individueen nur bei den Parametervektoren. Der Rekombinationsschritt wird zwar auch in diesen Experimenten

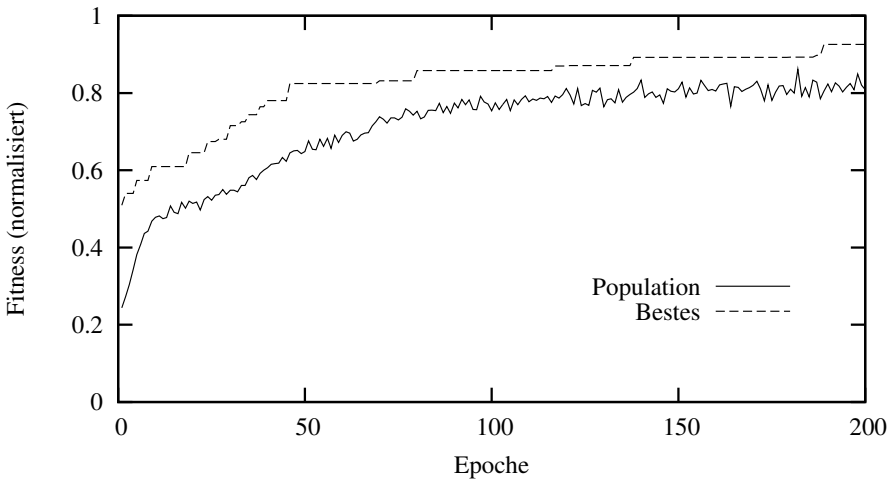

Abbildung 4.7: Fitness des besten Individuums und gemittelte Fitness der Gesamtpopulation über 200 Evolutionsepochen in der Subtraktionsaufgabe bei Beschränkung der Vorverarbeitungsschichten auf einen einzelnen CVTK-Operator.

aufgerufen, hat aber keinerlei Auswirkungen, da es keinen zweiten Operator zum Austauschen gibt. Die evolutionäre Veränderung beschränkt sich also auf eine Mutation der Parametervektoren.

Der CVTK Operator eignet sich für diese Experimente nicht nur weil er einen großen Parameterraum hat, sondern auch, weil sich seine Ergebnisse einfach visualisieren lassen und die Güte der evolvierten Vorverarbeitungsschichten an Hand der von ihnen erzeugten Regionenbilder leicht zu beurteilen ist.

In Abb. 4.7 ist die Entwicklung der durchschnittlichen Fitness der Population und des jeweils besten Individuums eines einzelnen Durchlaufes über 200 Epochen hinweg aufgetragen. Abbildung 4.8 zeigt den Hamming Abstand[1] des jeweils besten Individuums, gemessen auf der Trainings- und auf der Testmenge.

Wie zu sehen ist, wird die Trainingsmenge schnell gelernt; der Hamming Abstand auf dieser Menge ist bereits zu Beginn der Evolution sehr niedrig (siehe Abb. 4.8). Dies liegt daran, dass sehr viele Kombinationen der möglichen RGB-Tupel immerhin zu einem Ergebnis (mindestens eine vom Hintergrund verschiedene Fläche gefunden) der Segmentierungsschicht führen. Gerade schlecht ausgewählte Farbbeispiele können zu vielen kleinen Flächen führen. An Hand der resultierenden, zumeist sehr hochdimensionalen Daten können die Trainingsbeispiele leicht auswendig gelernt werden. Die so gefundenen und für die Diskriminierung herangezogenen Regionen stimmen aber zu einer sehr ho-

[1] Der Hamming Abstand gibt die Anzahl der Zeichen (oder “Bits”) an, in denen sich zwei Strings von Nullen und Einsen unterscheiden.

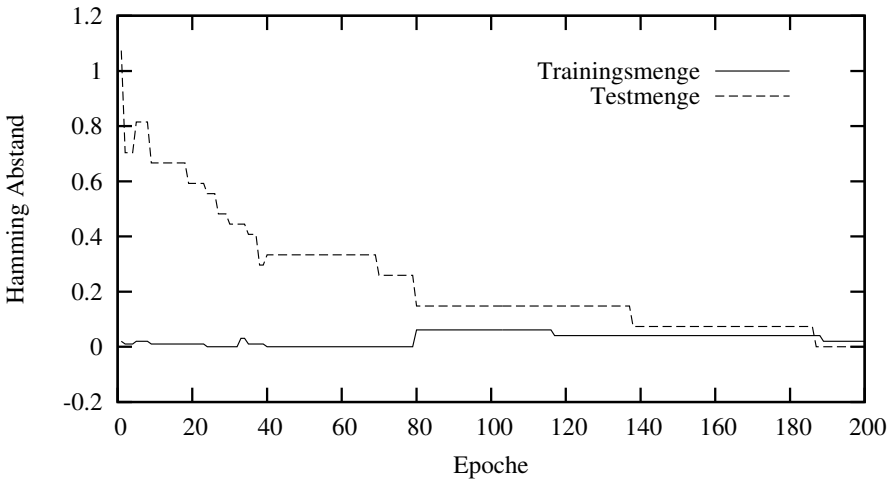

Abbildung 4.8: Hamming Distance des besten Individuums auf der Trainings- und auf der Testmenge in der Subtraktionsaufgabe bei Beschränkung der Vorverarbeitungsschichten auf einen einzelnen CVTK-Operator. Für die Darstellung wurde der Wert hinsichtlich der Anzahl der Pattern und der Neuronen in der Ausgabeschicht normiert.

hen Wahrscheinlichkeit noch nicht mit den tatsächlich relevanten Objektflächen überein, so dass der Generalisierungsfehler anfangs sehr groß ist und wesentlich langsamer absinkt. Erst nach ca. 180 Epochen ist im dargestellten Durchlauf aber auch der Generalisierungsfehler sehr niedrig.

In Abb. 4.7 sieht man, dass sich bei der gewählten (μ, λ)- evolutionären Strategie die Fitness des besten Individuums und die gemittelte Fitness aller Individuen einer Epoche parallel verbessern. Da niemals die besten Individuen von einer Epoche zur nächsten ersetzt werden, steigt die Fitness des besten Individuums monoton an, während sich die Population in ihrem Mittel auch verschlechtern kann. Zu beobachten ist, dass das beste Individuum nicht kontinuierlich in jeder Epoche, sondern in Sprüngen verbessert wird.

In den Abbildungen 4.9, 4.11 und 4.10 sind für drei Durchläufe die besonders deutlichen Verbesserungssprünge in Form der von den Operatoren erzeugten Regionenbilder dokumentiert.

An dem in der Abb. 4.10 dargestellten Durchlauf kann man besonders gut erkennen, dass vom evolutionären Prozess nicht nur ein einziger Kandidat verbessert wird. Es werden vielmehr mehrere Ansätze parallel entwickelt. In der vierten Epoche wird der beste Operator von einem Operator einer anderen "Familie" abgelöst, der die Farbklassen anders zuordnet, etwas hellere Blautöne und mehr von den roten Flächen erkennt. Allerdings markiert dieser Operator im Gegensatz zum abgelösten Operator sämtliche auf den blauen Spielsteinen gefundene Punkte mit demselben Farbmarker wie die Punkte auf den roten Spielsteinen.

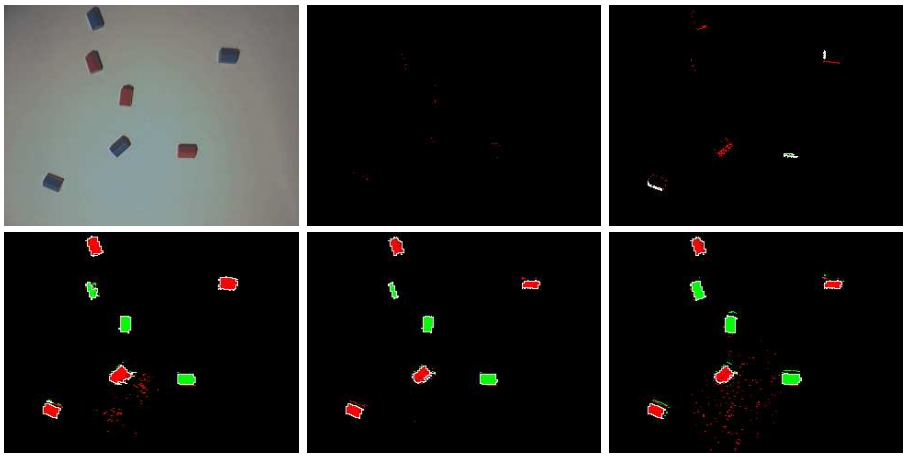

Abbildung 4.9: Originalbild aus der Testmenge der Subtraktionsaufgabe und vom Operator des jeweils besten Individuums der Populationen X_t, erzeugtes Regionenbild. Von oben nach unten, von links nach rechts ist die Ausgabe nach $t = 0$, $t = 12$, $t = 25$, $t = 50$, und $t = 150$ Epochen zu sehen.

In Generation acht überholt daher ein Nachkomme des ersten Operators diese Interimslösung wieder, nur um in Generation neun von einem anderen, nicht verwandten[2] Operator überholt zu werden, der an den Rändern der Flächen wesentlich genauer und damit anscheinend robuster arbeitet. Nach weiteren 33 Generationen wird aber auch dieser Ansatz wieder von einem optimierten Nachkommen des ursprüglichen Operators abgelöst, weil dieser nunmehr auch helle Streifen auf den roten Spielsteinen (zu sehen am unteren roten Spielstein) erkennt und damit seltener einen einzigen roten Spielstein in zwei Flächen aufteilt.

Während bei dem in Abb. 4.10 dokumentierten Durchlauf die Nachkommen des zu Beginn erfolgreichsten Operators nur zwischenzeitlich von anderen Ansätzen übertroffen werden und sich am Ende durchsetzen, wird bei dem Durchlauf der Abb. 4.11 der zu Beginn verfolgte Ansatz erst relativ spät in der 74. Generation von einem völlig anderen Operator abgelöst, der sich dann bis zum Ende aufgrund wesentlich besserer Ergebnisse behaupten kann. Interessant ist an diesem Durchlauf, dass sich der erste Operator so lange gegenüber dem auch bereits vor der Epoche 74 in der Population vorhandenen und später erfolgreicheren "zweifarbigen" Ansatz behaupten kann, obwohl er rote und blaue Spielsteine in die gleiche Farbklasse einordnet. Schaut man sich die Segmentierungsergebnisse (z.B. der 21. und 30. Generation) und die zugehörigen Paramtereinstellungen genauer an, stellt man fest, dass die erkannte Fläche der roten Spielsteine bei den meisten Bildern der Trainings- und Testmenge deut-

[2]Stammen zwei Individuen über beliebig viele Generationen von einem gemeinsamen Vorfahren ab, werden sie hier als "verwandt" bezeichnet.

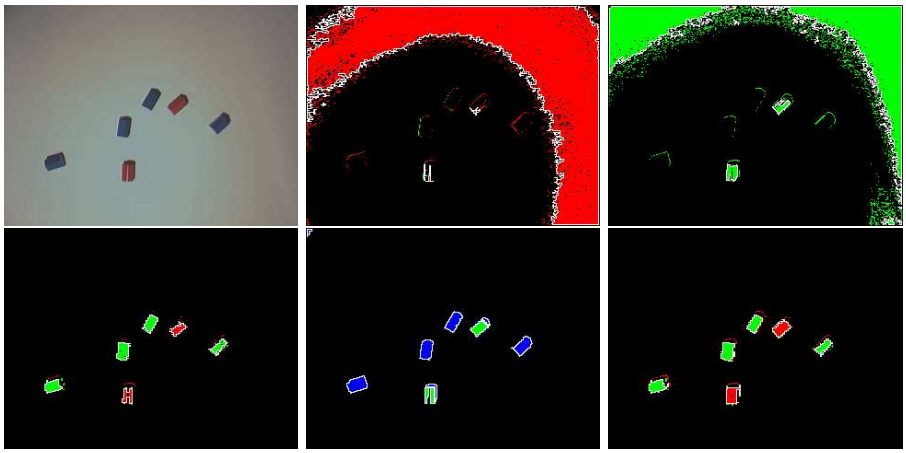

Abbildung 4.10: Originalbild aus der Testmenge der Subtraktionsaufgabe und vom Operator des jeweils besten Individuums der Populationen X_t, erzeugtes Regionenbild. Von oben nach unten, von links nach rechts ist die Ausgabe nach $t = 0$, $t = 4$, $t = 8$, $t = 9$, und $t = 42$ Epochen zu sehen.

lich kleiner ist als die Fläche der blauen Spielsteine. Da bei diesem Operator auch der Wert für die Fläche der gefundenen Regionen in den Ausgabevektor eingefügt wird, liegt die Vermutung nahe, dass das neuronale Netz die Größe der Flächen zur Unterscheidung blauer und roter Spielsteine verwendet (eine andere Information zur Diskriminierung ist zumindest für das Netz nicht verfügbar). Auch mit diesem überraschenden (kreativen?) Lösungsansatz ist bereits eine sehr gute, weit über dem Zufallsniveau liegende Generalisierung möglich.

Eine Inspektion der Populationen verschiedener Epochen und Durchläufe zeigt, dass vermutlich aufgrund der geringen Populationsgröße die Anzahl der unterscheidbaren, innerhalb einer Population parallel verfolgten Lösungsansätze eher gering ist (zumeist nur 2-3 vergleichbar erfolgreiche Ansätze). Oftmals stellt über die ganze Evolution hinweg ein und derselbe Lösungsansatz das beste Individuum aller Epochen, dessen Fitness, wie bei dem in Abb. 4.9 dargestellten Durchlauf, nie von einem Individuum eines gänzlich anderen Ansatzes übertroffen wird. Häufig dominiert dieser Ansatz die Population schon frühzeitig völlig und läßt kaum Raum für eine parallele Verfolgung anderer Ansätze. Falls zur Lösung einer komplexeren Aufgabenstellung notwendig, könnte hier vermutlich eine Erhöhung der Populationsgröße (in der Literatur findet man oftmals eine Populationsgröße von ca. $7 * \mu$ Individuen) oder die Evolvierung getrennter Subpopulationen helfen.

In Abb. 4.12 sind die evolvierten RGB-Farbtupel des besten Individuums des in Abb. 4.11 dargestellten Durchlaufs aufgeführt. Alle Tupel einer Farbklasse befinden sich im selben Rechteck. Auf den ersten Blick ist es nicht leicht zu erkennen, welche Tupel für welche Klasse stehen, da sich sowohl in der Bei-

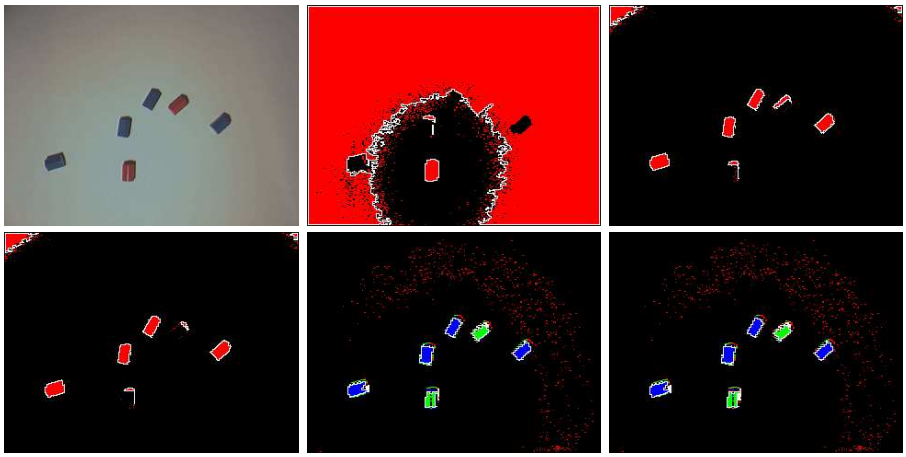

Abbildung 4.11: Originalbild aus der Testmenge der Subtraktionsaufgabe und vom Operator des jeweils besten Individuums der Populationen X_t, erzeugtes Regionenbild. Von oben nach unten, von links nach rechts ist die Ausgabe nach $t = 0$, $t = 21$, $t = 30$, $t = 74$, und $t = 171$ Epochen zu sehen.

spielmenge für Rot als auch für Blau rote und blaue Töne finden. Man selbst würde vermutlich intuitiv besonders "typische" Beispielfarben – also rötliche Töne für die Farbklasse "Rot" – zur Definition der Farbklassen wählen. Bei dem zur Segmentierung verwendeten Algorithmus spielen aber insbesondere die Grenzen der Farbkategorien eine wichtige Rolle. Daher ist es wichtiger, Farben in den Grenzbereichen einer Kategorien zu wählen, um eine saubere Abgrenzung gegenüber "benachbarten" Kategorien zu erreichen. Befinden sich schon einige Farben nahe an der Grenze, haben weitere Beispielfarben in der Clustermitte keinen Einfluss auf diese Grenzen und verändern das Segmentierungsergebnis nicht. Ein Beispiel für diese Methodik sind die beiden RGB-Tupel (20, 51, 107) (blau) und (53, 55, 103) (rot), die eng beieinander liegen. Das Tupel aus der roten Klasse ist zwar selbst ein bläulicher Farbton, grenzt aber den Bereich, der das Tupel (20, 51, 107) umgibt, eng gegenüber den Rottönen ab, die in der Richtung von (53, 55, 103) liegen.

Neben dieser systematischen Ursache gibt es noch einen weiteren Grund dafür, dass die Diversität bei den Farbbeispielen recht hoch ist. So wird in der derzeitigen Architektur eine größere Menge von RGB-Tupeln nicht gegenüber einer kleineren Menge, die das gleiche Ergebnis liefert, bestraft. Bei der Berechnung der Fitness wird nur die für die Segmentierung benötigte Rechenzeit gemessen, die wegen der verwendeten Lookuptabelle nicht von der Anzahl der Beispieltupel abhängt. Eine größere Anzahl von Tupeln wirkt sich nur negativ auf die bei der Initialisierung des Operators benötigte Laufzeit aus, die derzeit aber nicht bewertet wird. Daher gibt es keinen evolutionären Druck, Farbtupel, die keinen Einfluss auf die Farbzuordnung haben, weil sie vollständig von an-

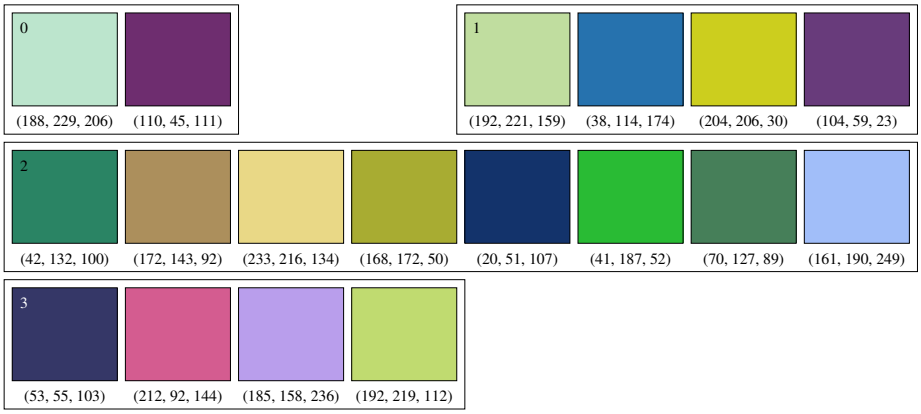

Abbildung 4.12: Nach der zugeordneten Farbklasse gruppierte Farbtupel des besten Individuums des in Abb. 4.11 dargestellten Durchlaufs. Die Farbklassen sind wie folgt verteilt: 0 = Hintergrund, 1 = Hintergrund, 2 = rote Spielsteine, 3 = blaue Spielsteine.

deren Beispielen für die gleiche Farbklasse umgeben sind, zu löschen. Gleiches gilt für Zuordnungsbeispiele, die zwar die Klassifikation in einem sie umgebenden Farbbereich ändern, deren beinflusster Farbbereich aber überhaupt nicht in den Bildern auftaucht. Treten in den Bildern keine grünen Pixel auf, kann dieser Farbbereich beliebig unterteilt und klassifiziert werden, ohne die Fitness zu verändern.

Neben den Farbtupeln, die den Segmentierungsalgorithmus beeinflussen, werden bei der Evolution auch noch die für den Ausgabevektor ausgewählten Eigenschaften optimiert. Beim CVTK-Operator können zu jeder gefundenen Bildregion die Position, die Fläche, die Kompaktheit und die Farbklasse ausgegeben werden. Einerseits hilft die Auswahl von vielen Eigenschaften den Trainingsfehler zu verringern. Andererseits kann die Auswahl zu vieler (irrelevanter) Eigenschaften zu schlechten Resultaten bei der Generalisierung führen und wird zudem durch den "Größenfaktor" in der Fitnessfunktion bestraft. Im Idealfall sollten daher am Ende der Evolution nur noch die Eigenschaften ausgewählt sein, die bei der Lösung der Aufgabe helfen und auch vom Menschen bei der Lösung der Aufgabe berücksichtigt worden wären.

Im Falle der Subtraktionsaufgabe spielt nur die Anzahl der Spielsteine einer jeden Farbe eine Rolle. Die Position, die Form und die Größe haben für das Ergebnis keine Bedeutung. Die kompakteste mögliche Kodierungsform wäre es demnach, nur die Farbklasse der gefundenen Regionen zurückzugeben. Für jede Region müsste so nur ein einziges Eingabeneuron vorgesehen werden.

In Abb. 4.13 sind die Features, die beim besten Individuum einer jeden Epoche berücksichtigt werden, für einen einzelnen Durchgang markiert. Es ist deutlich zu erkennen, dass zu Beginn der Evolution im Schnitt mehr Features

Tabelle 4.1: Evolvierte Parameterwerte des besten Individuums des in Abb. 4.11 dargestellten Durchlaufs. Die Werte der 18 Farbbeispiele sind in Abb. 4.12 zu sehen.

	Parameter	Wert
neuronales Netz	hidden layers	1
	neurons in hidden layer	11
	shortcuts	false
CVTK Operator	min_size	4
	color_classes	4
	t	71
	n_regions	10
	order	
	position	false
	area	false
	color	true
	compactness	false
	color_space	UV (YUV, Y nicht beachtet)

als am Ende verwendet werden. Gerade die Position wird bis zur 90. Epoche durchgehend von jedem Individuum berücksichtigt, obwohl sie bei der Kodierung sogar so viel Platz wie zwei andere Eigenschaften benötigt und die Ausgabe der Position deshalb auch genau doppelt so stark wie die anderen zur Verfügung stehenden Eigenschaften bestraft wird. Es wurde ja bereits festgestellt, dass die Individuen gerade zu Beginn der Evolution die Trainingsbilder eher auswendig lernen und noch wenig generalisieren. So gesehen ist es äußerst plausibel, dass möglichst viele Features und auch die Position berücksichtigt werden, da gerade die Position und auch die Kombination vieler Eigenschaften eine genaue Unterscheidung der einzelnen Bilder ermöglichen. Dazu müssen auch nicht die eigentlich relevanten Flächen gefunden werden; es reicht schon, wenn in jedem Bild an ein paar Stellen Flächen gefunden werden (ob im Hinter- oder im Vordergrund spielt dabei keine Rolle). Es müssen sich bloß die Positionen dieser "zufällig" gefundenen Flächen in den einzelnen Bildern unterscheiden.

Die Farbklasse bleibt bei dem hier aufgezeichneten Durchlauf und auch bei vielen anderen Durchläufen gerade zu Beginn häufig unberücksichtigt. Die Einstellung der richtigen Beispielfarben ist keine einfache Aufgabe und benötigt einige Epochen, bevor auch die relevanten Farbklassen voneinander unterschieden werden können. Erst ab diesem Zeitpunkt hat die Berücksichtigung der Farbinformationen einen Vorteil.

Mit der Abnahme der Anzahl der ausgewählten Features geht auch eine Abnahme der Größe der Eingabeschicht des neuronalen Netzes einher.

Mit Hilfe dieses ersten Experiments wurde gezeigt, dass der entworfene Algorithmus Operatoren und Netztopologien findet, die die Zielfunktion erfolgreich

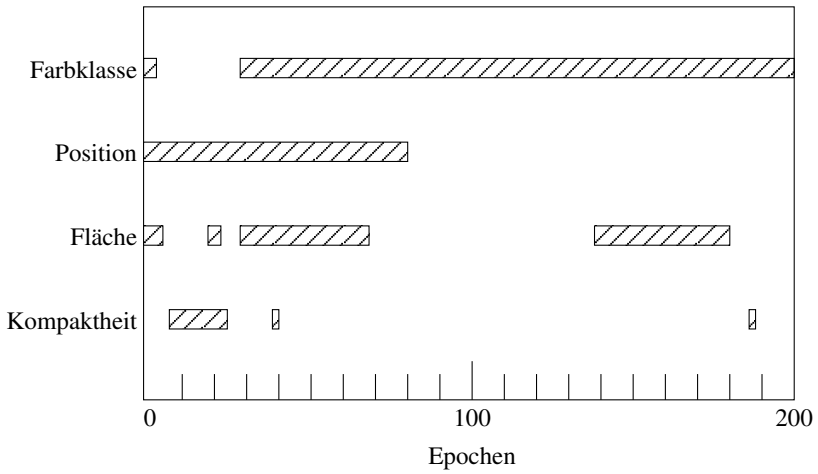

Abbildung 4.13: Die Features, die beim jeweils besten Individuum einer Epoche für den Ausgabevektor verwendet werden. Es handelt sich um den in Abb. 4.10 dargestellten Durchlauf.

lernen können und dass die gefundenen Operatoren die intendierten Eigenschaften (Farbklasse) heranziehen, um die Zielfunktion zu approximieren.

4.2.3 Evolution mit allen Operatoren

Nach diesen ersten Tests wurden nun die eigentlichen Experimente durchgeführt. Die verschiedenen Problemstellungen wurden nun unter Freigabe aller Operatoren gelernt. Jede Vorverarbeitungsschicht durfte nun bis zu 10 Operatoren in beliebiger Zusammensetzung enthalten.

In den ersten Testdurchläufen hat es sich als (empirsich) günstig erwiesen, die Evolution während der ersten 30 Epochen in getrennten Subpopulationen laufen zu lassen (siehe Tab. 4.2).

Eine Introspektion in die resultierenden Kontroller und die "Evolutionsgeschichte" gestaltet sich bei diesen Experimenten wesentlich schwieriger, da einige Operatoren (z.b. Histogramm) schwer zu visualisieren sind und nicht klar ist, wie die Informationen von mehreren verschiedenen Operatoren vom neuronalen Netz kombiniert werden. Die Berechnung der Ausgabe des Kontrollers ließe sich also allenfalls durch eine genauere Untersuchung der Netzstruktur nachvollziehen, die sich bei der Anzahl der erlaubten Neuronen in den versteckten Schichten als sehr aufwändig gestaltet.

Generell stellt sich aber die Frage, ob eine solche Untersuchung überhaupt notwendig und sinnvoll ist. Im ersten Abschnitt wurde durch eingehende Untersuchungen in einem für die Introspektion vereinfachten Setup bereits gezeigt, dass der Algorithmus generell in der Lage ist, stabil die richtigen Features aus

Tabelle 4.2: Anteil richtig klassifizierter Bilder bei unterschiedlich langer Evolution in homogenen Subpopulationen (Experiment 4.1.3). Die Evolution wurde für 0, 10 und 30 Epochen in getrennten Subpopulationen begonnen, die jeweils nur einen Operatortyp enthielten.

	Subtraktion	
	Testmenge	Testmenge 2
keine Isolation	96%	44%
10 Epochen isoliert	96%	56%
30 Epochen isoliert	100%	100%

den Bildern zu extrahieren und die Menge der für die Entscheidung berücksichtigten Daten zu minimieren – zwar nicht unbedingt bis auf ein Optimum, aber zumindest bis auf ein vernünftiges Maß. Nachdem dieser Nachweis erbracht wurde, erübrigt sich eigentlich eine weitere Untersuchung der resultierenden Kontroller: Die ursprüngliche Intention war es ja gerade, einen Algorithmus zu liefern, der selbstständig Bilder so verarbeitet, dass es ihm möglich ist, eine ihm gestellte Aufgabe zu erlernen. Wie dies geschieht und welche Lösung hierzu herangezogen wird, sagt eigentlich im Sinne der Problemstellung nichts über die Qualität des Kontrollers aus, solange die Aufgabe korrekt und zuverlässig gelöst wird. Man möchte sich ja eben nicht mehr mit den verschiedenen Lösungswegen beschäftigen, sondern einfach "visuelle" Aufgaben formulieren. Die Art der von den evolvierten Kontrollern verwendeten Lösung ist hierbei natürlich stark von der Aufgabenstellung abhängig. Von einer Bildverarbeitungsschicht eines Kontrollers, der die Aufgabe 4.1.2 löst, kann man zum Beispiel nicht erwarten, dass sie auch Informationen über die Position des Würfels liefert. Befindet sich der Würfel immer im gleichen Abstand zur Kamera, kann es sogar sein, dass der Würfel als solcher gar nicht erkannt wird, sondern mittels des Histogrammoperators lediglich die Gesamtfläche der schwarzen Augen berechnet und für die Bestimmung der Ausgabe verwendet wird.

Aus diesen Gründen wurden die evolvierten Kontroller in den verschiedenen Aufgabenstellungen im Wesentlichen lediglich auf ihre äußeren Eigenschaften (Generalisierungsleistung, Erprobung auf dem Roboter) untersucht und nur dann, wenn möglich einige interessante Ergebnisse der Evolution aufgeführt. In allen Aufgaben erzielt der Algorithmus Ergebnisse, die deutlich über der Zufallsrate liegen (siehe Tab. 4.3). Bei den Aufgabenstellungen 4.1.2, 4.1.3 und 4.1.4 liegen die Klassifizierungsergebnisse der neuen Bilder aus der zweiten Testmenge bei einer Korrektheit von 100%. Beim Experiment 4.1.1 wird konsistent eines der Bilder aus der zweiten Testmenge falsch klassifiziert. Dieses Bild zeigt einen zerissenen Post-it, der mit einem anderen Kameraabstand und bei einer wesentlich schwächeren Beleuchtung aufgenommen wurde.

Bei den Roboteraufgaben galt es keine Klassifizierungen sondern ein dreidimensionales Steuersignal zu lernen. Gemessen wurde hier der Trainings- und er

Tabelle 4.3: Anteil richtig klassifizierter Bilder in den Klassifikationsaufgaben. Getestet wurde jeweils das beste Individuum nach 200 Epochen. Die Testmenge 2 enthält nur Bilder, die während der Evolution weder zur Berechnung des Trainingsfehlers noch zur Berechnung des Generalisierungsfehlers des neuronalen Netzes verwendet wurden.

	Testmenge	Testmenge 2
Post-its zählen	100%	88%
Würfel ablesen	100%	100%
Subtraktion	100%	100%
Spielfiguren	100%	100%

Generalisierungsfehler. Gute Kontroller wurden anschließend auf den Robotern selbst getestet.

Im Falle der Aufgabe "Fahre zum Ball" auf dem Acroname Roboter wurden einige funktionierende Kontroller gelernt. In Abb. 4.14 ist die Entwicklung des Fehlers in einem dieser Durchläufe dargestellt.

Das Steuerprogramm verhält sich im Test auf dem Roboter weitestgehend richtig; es steuert den Roboter immer auf dem kürzesten Weg gerade zum Ball, wobei ein leichtes, wechselseitiges "Übersteuern" zu beobachten ist. Interessant ist, dass alle der getesteten Kontroller bei einem Abstand vom Ball von ca. 60 bis 30 cm ein identisches, "fehlerhaftes" Verhalten aufweisen: In diesem Bereich fährt der Roboter zwar noch in die richtige Richtung, dies aber nur sehr langsam. Es dauert daher oftmals mehr als 30 Sekunden, bis der Roboter dieses virtuelle "Hindernis", scheinbar widerwillig überwunden hat und wieder Geschwindigkeit aufnimmt.

Da dieses Verhalten konsistent bei mehreren eingelernten Kontrollern auftrat, kann davon ausgegangen werden, dass es sich wahrscheinlich nicht um einen Fehler während des Trainings oder in der Generalisierung handelt, sondern die aufgezeichneten Steuerdaten vielmehr widersprüchlich sind. Solche Widersprüche können zum Beispiel dann auftreten, wenn der Roboter vom Menschen zuerst mit seiner Vorderfront zum Ball zentriert worden ist, bevor er nach vorne gesteuert wurde. Wurde der Roboter hierzu das eine Mal von links nach rechts und ein anderes Mal von rechts nach links gesteuert, kann es vorkommen, dass sich die Bewegungsbeispiele in der "Mitte" (also im nahezu zentrierten Bereich) überlappen. Generell neigt der Mensch dazu, den Roboter mittels des Joysticks zu übersteuern und gerade diese Seitwärtsbewegungen zu spät zu verlangsamen.

Liegen in den Trainingsdaten einmal solche diametral entgegengesetzten Steuerdaten für ein und denselben Zustand vor, müsste ein neuronales Netz, dass den quadratischen Fehler minimiert, genau die Mitte zwischen diesen Steuersignalen wählen, was in diesem Fall mindestens zu einer Verlangsamung oder sogar zu einem Stoppen der Bewegung führen würde.

Eine weitere unerwartete Beobachtung in diesem Experiment mit dem Acro-

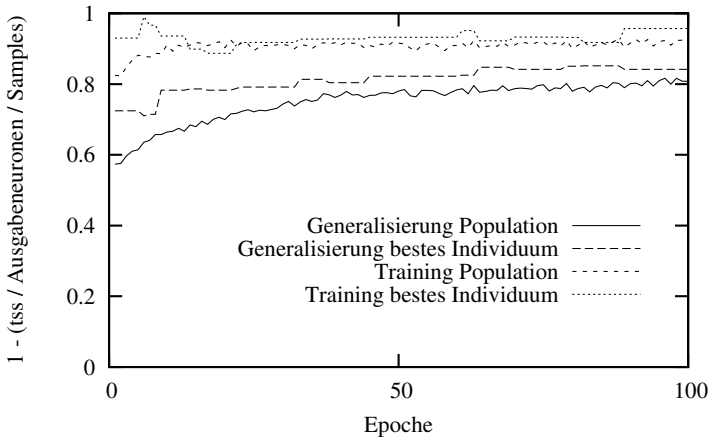

Abbildung 4.14: Entwicklung des Trainings- und Generalisierungsfehlers im Experiment 4.1.5.

name Roboter war, dass die Kontroller, die einen Segmentierungsoperator verwendeten, nicht immer so wie erwartet vorgingen. Würde ein Mensch die Segmentierung des Balles per Hand einstellen, würde er vermutlich in den allermeisten Fällen dafür sorgen, dass möglichst viele Pixel vom Ball und möglichst wenige Pixel vom Hintergrund gefunden werden. Die zum Ball gehörenden Pixel sollten möglichst bis zu den Rändern richtig als "Ball" klassifiziert werden, wobei man in der Regel ein paar einzelne, falsch klassifizierte Pixel im Hintergrund in Kauf nehmen würde.

Einige der evolvierten Segmentierungsoperatoren wurden vom Algorithmus aber gänzlich anders eingestellt: Sie detektieren nur einen dünnen, äquatorialen Streifen auf dem Ball. Der obere und untere Bereich des Balles wird großflächig falsch als Hintergrund klassifiziert. Dafür treten im Hintergrund aber auch so gut wie keine "falsch positiven" Pixel auf. Insgesamt ist die Summe der falsch positiven und der falsch negativen Pixel aber weit von jedem sinnvollen Optimum entfernt – wie auch immer man die Gewichtung zwischen falsch-positiven und falsch-negativen wählen würde.

Interessant ist, dass der Schwerpunkt des gefundenen äquatorialen Streifens nahezu identisch ist mit dem Mittelpunkt des Balles. Weil der Umfang und die Abgrenzung des Balles in der Aufgabenstellung keine Rolle spielen, handelt es sich also um eine adäquate (näherungsweise) Berechnung der benötigten Informationen. Der große Vorteil scheint hier zu sein, dass die üblichen Segmentierungsprobleme, die bei Highlights (Reflektionen der Deckenbeleuchtung im oberen Bereich des Balles) und bei starken Schatten (unterer Bereich des Balles) auftreten, einfach aber effektiv umgegangen werden.

Anschließend wurden die drei Experimente auf dem Tribot durchgeführt. Auch hier konnten gute Kontroller evolviert werden. In Abb. 4.15 ist die Ent-

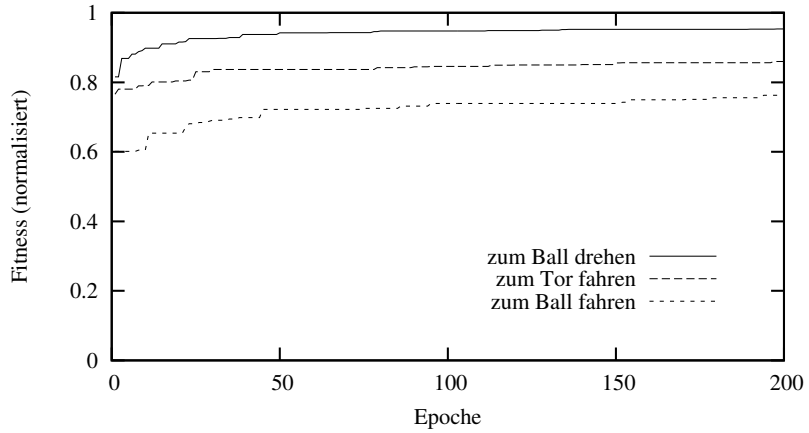

Abbildung 4.15: Verlauf der Fitness, gemessen am besten Individuum der Population, bei den drei auf dem Brainstormers Tribot durchgeführten Experimenten (siehe Abschnitt 4.1.6).

wicklung der Fitness des besten Individuums aufgezeichnet[3].

Die Aufgabe "zum Ball ausrichten" wird vom gelernten Kontroller zuverlässig gelöst. Der Roboter richtet sich sehr genau zum Ball aus, egal wie weit der Ball entfernt ist. Interessant ist, dass der Roboter kurz vor Erreichen der Zentrierung die Drehbewegung vermindert und durch dieses frühzeitige Abbremsen ein Übersteuern verhindert. Dieses Verhalten wurde so nicht mittels des Joysticks vorgeführt, sondern scheint aus den beiden demonstrierten Fällen "Bei Zentrierung keine Bewegung" und "Volle Drehgeschwindigkeit in Richtung Ball, wenn nicht zentriert" "emergiert" zu sein. Weiterhin ist interessant, dass der Roboter – durchaus sinnvoll – mit verminderter Geschwindigkeit rotiert, solange kein Ball im Bild zu sehen ist. Auch dieser Fall tritt so in den Trainingsdaten nie auf.

Während eines Tests des Kontrollers traten aber auch einmal Probleme auf: Plötzlich hörte der Roboter auf, sich zum Ball zu drehen und zitterte nur noch auf der Stelle. Der Grund wurde nach kurzer Zeit gefunden: Eine der im Raum anwesenden Personen war in der Trainingsphase nicht im Raum. Der Roboter hielt aus einer bestimmten Entfernung den grünen Pullover der betreffenden Person für einen orangen Ball. Da die Farbe des Pullovers in keinem Bild der Trainingsmenge auftauchte und auch in der sonstigen Umgebung kein ähnlicher Farbton vorkommt, ist dieser "Fehler" nicht weiter verwunderlich. Da der Farbton nicht in den Trainingsbildern enthalten ist, hat die Farbklasseneinteilung dieser Farbe auch keine Auswirkungen auf die Netzeingabe und damit auch

[3] Auf der Internetseite http://www-lehre.inf.uos.de/~slange/master/ werden Mutlimediadateien bereitgestellt, die das Trainieren und den Test der gelernten Kontroller ausführlich dokumentieren.

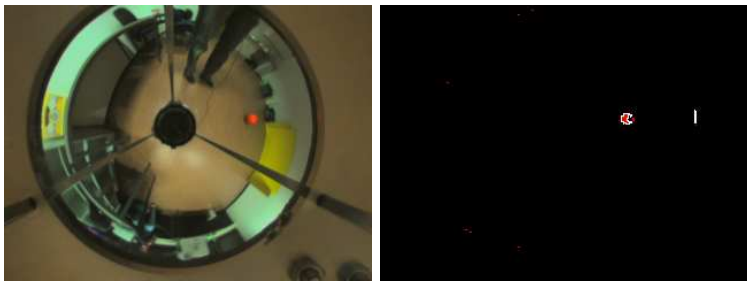

Abbildung 4.16: Testbild und zugehöriges, vom CVTK Operator des besten Individuums erzeugtes Regionenbild im Experiment “zum Ball ausrichten”. Bei diesem Durchlauf waren nur CVTK Operatoren in der Bildverarbeitungsschicht erlaubt.

Abbildung 4.17: Testbild und zugehöriges, vom CVTK Operator des besten Individuums erzeugtes Regionenbild im Experiment “zum Tor fahren”. Bei diesem Durchlauf waren nur CVTK Operatoren in der Bildverarbeitungsschicht erlaubt.

nicht auf die Fitness der Kandidaten.

Die Aufgabe “zum Tor fahren” wird ebenfalls gut gelöst. Der Roboter fährt von verschiedenen Positionen und bei verschiedenen Blickrichtungen zuverlässig zum Tor. Zumeist wird hierbei auch mit maximaler Geschwindigkeit eine gerade Linie verfolgt. Von manchen Stellen fährt der Roboter aber auch einen kleinen Bogen. Es ist aber in jedem Falle deutlich zu erkennen, dass der Roboter ständig zum Tor fährt; im Prinzip ist das Verhalten nicht von der handkodierten Version der Tribotsteuerung zu unterscheiden.

Die Aufgabe “zum Ball fahren” wird im Vergleich zu den beiden vorherigen Aufgaben nicht so gut gelöst. In mehreren Durchläufen zeigte sich immer dasselbe Problem: Das Anfahren des Balls funktioniert nur zuverlässig, wenn die Vorderseite des Roboters in eine ganz bestimmte Richtung gedreht wurde. “Schaut” der Roboter vom Tor aus gesehen nach rechts, dann funktioniert die Anfahrt zum Ball sehr gut. Hierbei spielt die Anfangsposition des Roboters und

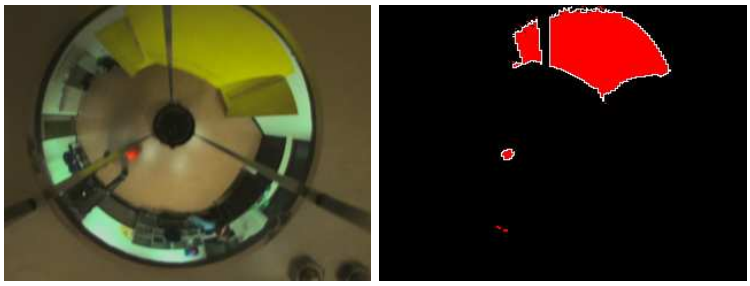

Abbildung 4.18: Testbild und zugehöriges, vom CVTK Operator des besten Individuums erzeugtes Regionenbild im Experiment "zum Ball fahren". Bei diesem Durchlauf waren nur CVTK Operatoren in der Bildverarbeitungsschicht erlaubt.

die Position des Balls keine Rolle: Egal ob der Ball vor, hinter oder neben dem Roboter liegt, er wird immer erreicht. Dabei umfährt der Roboter den Ball – falls notwendig – mit einem Bogen, um eine Kollision zu vermeiden. Generell erscheint das Verhalten sehr stabil und effizient zu sein. Auf den ersten Blick ist kein Nachteil gegenüber der handkodierten Version zu erkennen – ganz im Gegenteil. Dreht man den Roboter nun aber um 180°, so dass er vom Tor aus gesehen nach links schaut, ist plötzlich kein sinnvolles Verhalten mehr zu erkennen. Manchmal verfolgt der Roboter mit einem festen Abstand den rollenden Ball, kommt aber nicht näher. Manchmal scheint der Roboter die richtigen Bewegungen spiegelverkehrt auszuführen.

Um etwas Licht in dieses immer wiederkehrende Problem bei der letzten Aufgabe zu bringen, wurden die drei Aufgaben noch einmal nur mit den gut zu visualisierenden CVTK Operatoren trainiert. In Abb. 4.16 und Abb. 4.17 sieht man ein vom jeweils besten Individuum der ersten beiden Experimente erzeugtes Regionenbild. In beiden Bildern wird nur das aufgabenrelevante Feature extrahiert (Ball respektive Tor). In Abb. 4.18 ist das Ergebnis im Falle des dritten Experiments abgebildet. Wie zu sehen ist, wird hier nicht nur der Ball, sondern auch das Tor in der gleichen Farbklasse segmentiert. Da die Position des Tors somit an das neuronale Netz weitergegeben wird, ist zu erwarten, dass diese Information auch vom neuronalen Netz verwendet wird. Anscheinend befinden sich in der Menge der Trainingsdaten zu wenige Beispiele von unterschiedlichen Positionen des Tores. Da im Training der Roboter tatsächlich in den meisten Fällen vom Tor aus gesehen nach rechts ausgerichtet war, scheint es von Vorteil, bzw. zumindest nicht von Nachteil zu sein, die Position des Tores zu berücksichtigen und so vielleicht bessere Ergebnisse auf den Trainingsdaten zu erreichen.

Diese Beobachtung zeigt noch einmal, wie wichtig die Zusammenstellung der Trainingsdaten ist. Es muss unbedingt versucht werden, eine Strategie vorzuführen, die möglichst nur an Hand der "richtigen" Features hergeleitet werden kann. Wenn ein nicht bedachtes Bildfeature auch nur zufällig mit den Ausgabe-

signalen korreliert, ist die Wahrscheinlichkeit sehr hoch, dass dieses Feature von der Bildverarbeitung auch gefunden und vom Kontroller berücksichtigt wird. Im Falle des zum Teil fehlgeschlagenen letzten Experiments hätte man lediglich für eine größere Diversität bei den Drehwinkeln des Roboters in Kombination mit verschiedenen Ballpositionen sorgen müssen.

4.3 Diskussion der Ergebnisse

Die bereits in den ersten Versuchen (Abschnitt 4.2.2) gewonnenen Daten zeigen deutlich, dass der Testfehler und nicht der Trainingsfehler das relevante Minimierungskriterium in der Evolution darstellt. Ein niedriger Trainingsfehler hat bei der gewählten Architektur der Kontrollprogramme kaum eine Aussagekraft hinsichtlich der zu erwartenden Qualität der Generalisierung und der damit verbundenen Performance des evolvierten Kontrollprogramms. Der Trainingsfehler kann lediglich als initialer Indikator bei der Auwahl der zu Beginn der Evolution noch wenig entwickelten Kandidaten dienen. Wie die Visualisierungen der Durchläufe mit dem einzelnen CVTK Operator zeigen, reicht eine Minimierung des Trainingsfehlers allein nicht aus, um eine gute Segmentierung zu erreichen. Obwohl der Trainingsfehler auch in den ersten Epochen sehr niedrig ist und im weiteren Verlauf nicht mehr wesentlich sinkt, sondern eher wieder ansteigt, werden die für die Aufgabe relevanten Flächen von den Bildverarbeitungsschichten noch nicht sonderlich genau und zuverlässig segmentiert (siehe Epochen 0 und 12 in Abb. 4.9 und Epochen 0 und 4 in Abb. 4.10).

In den Experimenten hat sich die Annahme bewahrheitet, dass in der Formulierung der Aufgabenstellung in Form von Samples der Zielfunktion tatsächlich ausreichend viele Informationen enthalten sind, um die "wesentlichen" Merkmale finden zu können. In jedem Durchlauf der Subtraktionsaufgabe mit dem einzelnen CVTK Operator wurden bisher bis zum Ende der Evolution genau zwei verschiedene Farbklassen für die Spielsteine eingestellt. Informationen über die Position und die Größe der Flächen blieben zumeist unberücksichtigt. Zum Ende wiederholter Evolutionsdurchläufe werden mit hoher Wahrscheinlichkeit die der Aufgabenstellung "inheränten" Eigenschaften gefunden, auch wenn zwischenzeitlich überraschende Lösungen (siehe Abschnitt 4.2.2) verfolgt werden können.

Ob die bei der Aufgabenstellung intendierten Entscheidungsmerkmale gefunden werden, hängt allerdings sehr stark von der Aufgabenstellung ab. Es werden im Allgemeinen nie mehr Informationen als nötig extrahiert. Dies zeigt sich zum Beispiel darin, dass bei der Aufgabe, in der die Augenzahl eines Würfels bestimmt werden soll, oftmals der Histogrammoperator verwendet wird. Diese Lösung erkennt den Würfel überhaupt nicht als Objekt, wie man es intuitiv vermuten würde. Außerdem ist wichtig, dass die Trainingsdaten keine Zweideutigkeiten enthalten. Korreliert ein Bildfeature (z.B. eine Landmarke) zufällig mit dem angelegten Steuersignal, kann es von den evolvierten Kontrollprogrammen zur Berechnung herangezogen werden, so wie es auf dem Tribot in der "zum Ball fahren" Aufgabe passiert ist.

Überraschend ist aber, dass eine genaue (exakte) Verarbeitung anscheinend bei allen Aufgaben eine wichtige Rolle spielt, zum Teil weit stärker, als man intuitiv hätte annehmen können (Beispiel Subtraktion mittels CVTK Operator: Die Spielsteine werden vollständig erkannt und segmentiert). Man selbst erkennt zwar die genaue Fläche und den exakten Verlauf der Ränder, man hätte einem evolvierten Algorithmus aber wohl auch zugestanden, nur einige wenige "sichere" Punkte auf dem Spielstein zu erkennen. Diese Beobachtung gilt auch, wenn die Genauigkeit in der Aufgabenstellung selbst nicht "formuliert" wurde: Im Subtraktionsbeispiel ist die Aufgabe eben nicht "finde die Ränder (damit ich zugreifen kann)", sondern eine bloße Detektionsaufgabe.

Kapitel 5

Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Algorithmus vorgestellt, der in der Lage ist, eine Kontrollfunktion direkt auf Bilddaten zu lernen. Dabei werden von dem Algorithmus geeignete Operatoren für eine Bildverarbeitungsschicht ausgewählt und konfiguriert. Die Bildverarbeitungsschichten werden für die Aufgabe hinsichtlich der Art und der Anzahl der extrahierten Features optimiert. Auf der so erzeugten, kompakten Ausgabe der Bildverarbeitungsschicht wird ein neuronales Netz, dessen Topologie ebenfalls evolutiv an die Aufgabenstellung angepaßt wurde, trainiert, um die Kontrollfunktion zu realisieren. Der Algorithmus wurde erfolgreich in Bildklassifikations- und Navigationsaufgaben getestet.

Der vorgestellte Algorithmus trägt der Idee Rechnung, dass Bildverarbeitung generell zielgerichtet sein sollte. Es ist bisher nicht gelungen, ein ganzheitliches Bildverarbeitungssystem zu entwickeln, dass eine vollständige (symbolische) Beschreibung der Umgebung liefern kann, die für alle denkbaren Aufgaben geeignet ist. Vielmehr beschränkt sich die Disziplin des maschinellen Sehens derzeit vornehmlich auf die Untersuchung spezieller Einzelprobleme und auf die Entwicklung von aufgabenspezifischen Systemen aus diesen gut untersuchten Bausteinen. Der hier verfolgte Ansatz automatisiert die Entwicklung problemspezifischer (zielgerichteter) Bildverarbeitungssysteme. In den Experimenten wurde gezeigt, dass von diesen automatisch erzeugten Systemen "bedeutungsvolle" Features extrahiert werden, die helfen, die Aufgabe zu lösen. Die Ergebnisse legen nahe, dass vom System aber nur genau die Informationen verarbeitet werden, die für die Erreichung des Ziels vonnöten sind. Ein darüber hinaus gehendes "Verständnis" der Wahrnehmungen wird nicht erreicht. Diese Beobachtungen gehen einher mit Beobachtungen in der biologischen Evolution. Auch hier wirken ähnliche Mechanismen und es gibt starke Anzeichen dafür, dass sich auch unser Sehsystem zielgerichtet von einfachen, helligkeitsempfindlichen Sinneszellen bis hin zu unserem heutigen Wahrnehmungsapparat entwickelt hat. In diesem Sinne der Zielgerichtetheit kann ein Bildverarbeitungssystem ohne

eine zugehörige Aufgabenstellung und die damit verbundenen Anforderungen überhaupt nicht "vollständig" und korrekt sein.

Unter diesem Aspekt ist das eigentliche Problem des maschinellen Sehens nicht, ein "vollständiges" (was immer das heißen mag) und allgemeingültiges, losgelöstes Bildverarbeitungssystem zu liefern, sondern zuerst einmal die "richtigen" Aufgabenstellungen zu formulieren. Es gilt nicht, die Leistungsfähigkeit eines losgelösten Bildverarbeitungssystems zu bewerten, sondern das Verhalten des Gesamtsystems (Bildverarbeitung und Entscheidungsfindung) als Ganzes zu betrachten. Ein kleiner Schritt auf dem Weg in diese Richtung wurde hier mit der Fokussierung auf die Aufgabenstellung und der Untersuchung des Erfolges des gelernten Gesamtsystems unternommen.

Bisher handelt es sich bei dem System nur um eine rudimentäre Implementierung. Es stehen nur wenige Operatoren bereit, so dass in Zukunft noch wesentlich mehr Algorithmen integriert werden müssten. Generell sollte es auch nicht bei der einfachen, parallelen Anordnung der Bildverarbeitungsoperatoren bleiben. Es müsste möglich sein, Operatoren hintereinanderzuschalten, um die Ausgaben der vorgeschalteten Operatoren weiterverarbeiten zu können. Wie dies realisiert werden kann, zeigen die Arbeiten von Belpaeme, Martin und insbesondere Draper.

Spätestens wenn man die Anzahl der Operatoren oder die Komplexität der Aufgaben weiter erhöht, sollten Möglichkeiten der Parallelisierung des Algorithmus untersucht und implementiert werden. Prinzipiell kann die Bewertung der Kandidaten einer Population gleichzeitig geschehen, da die Fitness nicht in einer Konkurrenzsituation sondern für jedes Kontrollprogramm einzeln berechnet wird. Ebenfalls können die homogenen Subpopulationen, die zum Evolutionsbeginn verwendet werden, parallel evolviert werden. Darüber hinaus ist auch eine Parallelisierung des gesamten Evolutionsablaufes in mehreren Subpopulationen denkbar. In der Literatur finden sich hier verschiedene allgemeine Ansätze für evolutionäre Algorithmen.

In einem weiteren Schritt könnten die Bildverarbeitungsoperatoren selbst für die eingesetzte Hardware optimiert werden. Zwar verliert man hier einiges an Flexibilität, aber durch den Einsatz von Intel's SSE könnten zum Beispiel bei der Segmentierung im CVTK Operator einige Rechenschritte parallelisiert und um Faktoren beschleunigt werden.

Eine deutliche Beschränkung stellt derzeit die Formulierung als Problem überwachten Lernens dar. Die Kombination mit selbstorganisierten Ansätzen wie z.B. Reinforcement Learning müsste untersucht werden. Allerdings sind auch für den hier beschriebenen Algorithmus praktische Anwendungsmöglichkeiten denkbar. So wurde in Dortmund von Stefan Welker und Artur Merke [24] ein System zur linienbasierten Selbstlokalisation eines mobilen Roboters entwickelt, dass auf den Tribots im Roboterfußball eingesetzt wird. Ein Problem besteht bei diesem Ansatz darin, dass das Spielfeld symmetrisch ist und der Roboter so im Verlaufe des Spiels die Orientierung verlieren kann und sich plötzlich auf der falschen Spielfeldhälfte wiederfindet. Nur an Hand der Linien ist es aufgrund der Symmetrie nicht möglich, zu entscheiden, auf welcher Spielfeldhälfte man sich befindet. Mit Hilfe des hier dargelegten Algorithmus wäre es nun möglich,

neben der linienbasierten ersten Hypothese eine weitere Hypothese zu lernen, die auf anderen, spielfeld- bzw. umgebungsspezifischen Features basiert. Diese zweite, lokale Hypothese könnte ebenfalls eine Position oder nur schlicht eine Vermutung über die aktuelle Spielfeldseite liefern. Geht man davon aus, dass der Roboter am Anfang weiss, auf welcher Seite er sich befindet und diese Information auch über einen kurzen Zeitraum (einige Sekunden) zuverlässig ist, könnten genug Trainingsbeispiele für den hier beschriebenen Lernalgorithmus eingesammelt werden. Das Lernen lokaler Orientierungspunkte (Landmarken) könnte somit vollautomatisch geschehen. Man könnte diesen Algorithmus sogar inkrementell betreiben, um ihn an veränderte Umgebungsbedingungen anzupassen.

Generell bestünde also ein selbstorganisierendes Einsatzgebiet des hier beschriebenen Algorithmus darin, eine bereits vorhandene, allgemein gefasste und / oder rechenaufwändige erste Hypothese durch eine "lokalisierte" und / oder zeitlich optimierte weitere Hypothese zu ergänzen oder sogar zu ersetzen.

Literaturverzeichnis

[1] The 1394 Trade Association: *IIDC 1394-based Digital Camera Specification Version 1.30*, Dokument Nr. 1999023, http://www.1394ta.org (1999)

[2] Arbatzat, M., Freitag, S., Fricke, M., Hafner, R., Heermann, C., Hegelich, K., Krause, A., Krüger, J., Lauer, M., Lewandowski, M., Merke, A., Müller, H., Riedmiller, M., Schanko, J., Schulte-Hobein, M., Theile, M., Welker, S., Withopf, D.: Creating a Robot Soccer Team from Scratch: the Brainstormers Tribots, *Proceedings of Robocup 2003*, Padua, Italy (2003)

[3] Bala, J. , DeJong K., Huang, J., Vafaie, H., Wechsler, H.: Hybrid Learning Using Genetic Algorithms and Decision Tress for Pattern Classification, *14th Int Joint Conf. on Artifical Intelligence (IJCAI)*, Canada (1995) 719-724

[4] Belpaeme, T.: Evolution of Visual Feature Detectors, *Proceedings of the First European Workshop on Evolutionary Computation in Image analysis and Signal Processing (EvoIASP99, Göteborg, Sweden*, University of Birmingham School of Computer Science technical report. (1999)

[5] Bertsekas, D. P.: *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, MA (1995)

[6] *The Brainstormers Tribots*, http://amy.informatik.uni-osnabrueck.de/tribots/ (2004) letzter Besuch: April 2004

[7] Braitenberg, V.: *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, MA (1984)

[8] Braun, H., Weisbrod, J.: Evolving feedforward neural networks, *Proc. of the International Conference on Artificial Neural Nets and Genetic Algorithms* (1993)

[9] Braun, H., Ragg, T.: Evolutionary Optimization of Neural Networks for Reinforcement Learning Algorithms, *ICML96, Workshop Proceedings on Evolutionary Computing and Machine Learning*, Italy (1996) 38-45

[10] McCabe, G.P.: Principal Variables, *Technometrics vol. 26* (1984) 127-134

[11] Draper., B.: *Learning Object Recognition Strategies*, Ph.D. dissertation, Univ. of Massachusetts, Dept. of Computer Science. Tech. report 93-50 (1993).

[12] Draper, B.: Learning Control Strategies for Object Recognition, *Visual Learning*, Ikeuchi, Veloso (Hrsg.), Oxford University Press (1996)

[13] Floreano D., Mondada F.: Hardware solutions for evolutionary robotics, *Proc. of the First European Workshop on Evolutionary Robotics*, Husbands, P., Meyer JA., Hrsg., Springer-Verlag (1998)

[14] Foley, J. D, van Dam, A., Fiener, S. K., Hughes, J. F.: *Computer Graphics Principles and Practice*, Second Edition, Addison-Wesley (1996)

[15] Freeman, H.: On the encoding of arbitrary geometric configuration, *IRE Transactions on Electronic Computers, EC-10(2)*, 1961 260-268

[16] Glasbey, C. A.: An Analysis of Histogram-Based Thresholding Algorithms, *CVGIP: Graphical Model and Image Processing 55(6)* (1993) 532-537

[17] Hübner, K.: *Methods for Range Estimation and Situation Recognition using an Omnidirectional Vision System for Mobile Robots – Symmetry as a Natural Feature*, Diplomarbeit an der Universität Bielefeld (2001)

[18] Hyvärinen, A., Karhunen, J., Oja, E., *Independent Component Analysis*, John Wiley & Sons (2001)

[19] Jähne, B., *Digitale Bildverarbeitung*, Springer-Verlag (1997)

[20] Jolliffe, I.T., *Principal Component Analysis*, Second Edition, Springer-Verlag (2002)

[21] Krzanowski, W.J.: Selection of Variables to Preserve Multivariate Data Structure, Using Principal Component Analysis, *Applied Statistics- Journal of the Royal Statistical Society Series C vol. 36* (1987) 22-33

[22] K-Team Corporation, Hersteller des Khepera und Khepera II, http://www.k-team.com, letzter Besuch: April 2004

[23] Lange, S.: Verfolgung von farblich markierten Objekten in 2 Dimensionen, B.Sc. thesis, Universität Osnabrück, Institut für Kognitionswissenschaft (2001)

[24] Merke, A., Welker, S.: *Line Based Robot Localization under Natural Light Conditions*, bisher unveröffentlicht (2004)

[25] Marek, A.J., Smart, W.D., Martin, M.C.: Learning Visual Feature Detectors for Obstacle Avoidance using Genetic Programming, *GECCO Late Breaking Papers* (2002) 330-336.

[26] Martin, C. M.: *The Simulated Evolution of Robot Perception*, Ph.D. dissertation, Carnegie Mellon University Pittsburgh (2001)

[27] Pasemann, F., Steinmetz, U., Hülse, M., Lara, B.: Robot control and the evolution of modular neurodynamics, *Theory in Biosciences, 120* (2001) 311-326

[28] Pasemann, F., Hülse, M., Zahedi, K.: Evolved Neurodynamics for Robot Control, *European Symposium on Artificial Neural Networks'2003*, Verleysen, M. (Hrsg.), D-side publications (2003) 439-444

[29] Pomerleau, D., ALVINN: An Autonomous Land Vehicle In a Neural Network, *Advances in Neural Information Processing Systems 1.* Morgan Kaufmann (1989)

[30] Pomerleau, D., *A Neural Network Perception for Mobile Robot Guidance*, Ph.D. Dissertation, Carnegie Mellon University (1992)

[31] Pomerleau, D., Neural Network Vision for Robot Driving, *The Handbook of Brain Theory and Neural Networks,* M. Arbib (Editor) (1995)

[32] Priese, L., Rehrmann, V., Schian R., Lakmann, R.: Traffic Sign Recognition Based on Color Image Evaluation, *Proc. Intelligent Vehicles Symposium* (1993) 95-100

[33] Quinlan, J.R.: Learning efficient classification procedures and their application to chess endgames, *Machine learning: An artificial intelligence approach, Vol I*, Michalski, R. S., Carbonell, J., Mitchell, T. M. (Hrsg.) Tioga Press (1983) 463-482

[34] Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: the Rprop algorithm, *Proceedings of the ICNN* (1993)

[35] *RoboCup German Open 2004*, http://www.ais.fraunhofer.de/GO/2004/ (2004) letzter Besuch: April 2004

[36] Schölkopf, B., Smola, A., J., *Learning with Kernels*, MIT Press (2002)

[37] Smith, S. M.: A new class of corner finder, *Proc. 3rd British Machine Vision Conference* (1992) 139-148

[38] Sonka, M., Hlavac, V., Boyle, R., *Image Processing, Analysis and Machine Vision.* Second Edition, P.W.S. Publishing (1999)

[39] Steels, L.: Language games for autonomous robots, *IEEE Intelligent systems* (2001) 17-22

[40] Steels, L., Kaplan, F.: AIBO's first words. The social learning of language and meaning, *Evolution of Communication, vol. 4, nr. 1*, Gouzoules, H. (Hrsg.), John Benjamins Publishing Company (2001)

[41] Tanese, R.: *Distributed Genetic Algorithms for Function Optimization*, Ph.D. thesis, University of Michigan (1989)

[42] Thorpe, C., Carlson, J. D., Duggins, D., Gowdy, J., MacLachlan, R., Mertz, C., Suppe, A., Wang, C.: Safe Robot Driving in Cluttered Environments, *Proceedings of the 11th International Symposium of Robotics Research* (2003)

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

Hilfsmittel

Die Programme für die Experimente wurden auf einem Linux Rechner in C/C++ und Tcl/Tk unter Zuhilfenahme geeigneter Compiler und Interpreter erstellt. Dabei wurde bei der Bildverarbeitung zum Teil auf Implementationen von Priese (Color Structure Code, CSC) und Smith (SUSAN Edge Detector) zurückgegriffen. Für das neuronale Netz und Rprop wurde eine Implementation von Riedmiller (n++) verwendet. Außerdem wurden die GNU Scientific Library (GSL) und die Qt Bibliothek von Trolltech eingesetzt.

Die Experimente wurden auf einem Acroname PPRK und auf einem Brainstormers Tribot durchgeführt. Bei den Experimenten mit dem Tribot wurde bei der Ansteuerung der Roboterhardware auf den bestehenden Sourcecode der Brainstormers Tribots zurückgegriffen. Die Bildserien wurden mit verschiedenen Digitalkameras aufgenommen.

Die Zeichnungen wurden mit xfig, die Diagramme mit gnuplot angefertigt. Die Arbeit wurde mit Latex gesetzt.

Hiermit versichere ich nach §20, Abs. 6 der aktuell gültigen Prüfungsordnung, dass ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Osnabrück, den