hochschule für angewandte wissenschaften
FACHBEREICH ELEKTROTECHNIK UND INFORMATIK hamburg
university of applied sciences

Diplomarbeit

Arne Dieckmann

Verbesserung visueller Objekterkennung für mobile Roboter

Studiengang Technische Informatik
Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Gunter Klemke
Abgegeben am 13. März 2003

BERLINER TOR 3 · D - 20099 HAMBURG

Danksagung

Hiermit möchte ich mich bei meinem Betreuer Kai von Luck bedanken, bei meiner Verlobten Julia für ihre unendliche Geduld und bei allen, die mich in irgendeiner Form unterstützt haben.

DANKE, DANKE, DANKE!!!

Verbesserung visueller Objekterkennung für mobile Roboter

Stichworte Objekterkennung, Roboter, Pioneer, Saphira, Framegrabber

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit der visuellen Objekterkennung für mobile Robotor. In diesem speziellen Fall geht es um die Pioneer-Roboter von ActivMedia. Hierbei soll mit geringem Resourcenaufwand die bestehende Objekterkennung der Sonycamera unterstützt und somit verbessert werden. Der Schwerpunkt liegt dabei in der Formerkennung eines Gegenstandes.

Improvement of visual object recognition for mobile robots

Keywords Recognition, Robots, Pioneer, Saphira, Framegrabber

Abstract

This thesis (diploma) concerns itself with the visual object recognition for mobile Robotor. In this special case it goes around the Pioneer-Robot from ActivMedia. The existing object recognition of the Sonycamera is to be supported and improved thus at small resource expenditure. The emphasis is thereby in the shape identification of an object.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Roboter sind in der heutigen Zeit gar nicht mehr wegzudenken. So gibt es schon diverse Anwendungen im industriellen Bereich, wie Lagerroboter für simple Sortieraufgaben oder Schweißroboter in der Autoherstellung und viele mehr.

Aber auch im privaten Bereich finden die elektronischen und mechanischen Helfer einen immer größer werdenden Zuspruch. Im Garten beispielsweise werden sie als Rasenmäher eingesetzt, wo sie selbstständig ihren Weg über das zu mähende Feld finden, um welches ein Draht gespannt ist, an dem sich der Roboter mit Hilfe von Berührungssensoren orientieren kann. Er stellt somit eine ideale Unterstützung für den etwas ”fauleren“ Gartenfreund dar.

Andere Robotor sind hier schon sehr viel besser ausgestattet, was die Fähigkeit der Wahrnehmung ihrer Umwelt angeht. So besitzen sie zum Beispiel Sonar- oder Laserscanner für Entfernungsmessungen oder Kameras, um Objekte zu erkennen bzw. zu verfolgen. Hiermit lassen sich dann schon sehr einfache ”Gehilfen“ realisieren, die dann als Bürobote zum Post abholen oder zum Kaffee servieren eingesetzt werden können.

Hier setzt auch diese Diplomarbeit an. Da ich schon einige Erfahrungen mit den Robotern des Robot-Labs der HAW-Hamburg und der aufgesetzten Kamera

sammeln konnte, die nicht immer alle sehr positiv waren, habe ich mir überlegt, die Bildverarbeitung für den Roboter zu verbessern. Hierbei geht es mir vor allen Dingen darum, Objekte mit einer höheren Trefferrate zu erkennen und möglicherweise unterscheiden zu können. Hierbei soll kein komplett neues System entstehen, sondern vielmehr die Anwendung von Bild verarbeitenden Algorithmen die bestehenden Fähigkeiten der Kamera erweitern.

Kapitel 2

Rahmenbedingungen

2.1 Robot–Lab der Hochschule für Angewandte Wissenschaften Hamburg

Seit 1996 wird an der Hochschule für Angewandte Wissenschaften Hamburg im Fachbereich für Elektrotechnik und Informatik das Projekt IKS [10] betrieben. IKS steht für ”Integration Kognitiver Systeme“. Der Schwerpunkt dieses Projektes ging aber sehr bald in Richtung kognitiver Roboter. Im Rahmen dieses Projektes entstand an der Hochschule für Angewandte Wissenschaften Hamburg ein Robotik-Labor für Ausbildungszwecke. Dieses Labor macht heute den größten Teil des IKS Projektes aus.

Das Robot–Lab besteht mittlerweile aus verschiedenen Teilen. Im Rahmen des Labores ist ein eigenständiges Board, das Krabat Board [11], für Robotik-Zwecke entwickelt worden, auf dem u.a. ein embedded Linux läuft. Der Teil, der sich mit Lego Robotern beschäftigt, ist inzwischen zweigeteilt. Ein Teil setzt das bereits 1989 am MIT entwickelte 6.270 Board [15] ein, das nach der Kursnummer am MIT benannt worden ist. Der andere Teil benutzt die LEGO Mindstorms [9]. Ein weiterer Teil des Labores beschäftigt sich mit den weit verbreiteten Pioneer Robotern der Firma Activ Media [1]. Diese sind wesentlicher Bestandteil dieser Arbeit.

2.2 Hardware

2.2.1 Pioneer 2–DX

Den Mittelpunkt dieser Arbeit stellt der Pioneer 2–DX Robotor der Firma Active Media dar, sowie die dazugehörige Steuersoftware Saphira [2], die aber in einem eigenen Abschnitt näher betrachtet wird.

Der Pioneer 2–DX Robotor stammt aus der Pioneer Roboter Familie der Firma Activ Media, die mit diesem Paket ein sehr funktionelles und universell einsetzbares Einstiegssystem auf dem Markt anbietet.

Abbildung 2.1: Pioneer 2–DX

Der Pioneer 2–DX besitzt zwei vordere Antriebsräder und ein hinteres Stützrad. Durch diese Konstruktion ist es ihm möglich, auf der Stelle zu drehen. Des Weiteren ist er mit einem an der Vorderseite gelegenen Sonarring ausgestattet. Zusätzlich besitzen die zwei an der HAW Hamburg eingesetzten Roboter noch folgende Komponenten:

- einen Greifer

- Sony Farb-Kamera
- Cognachrom Vision System

Herz des Roboters ist ein Board mit einem mit 20 MHz getakteten Siemens 88C166 basierenden Mikrocontroller. Gesteuert wird der Roboter üblicherweise durch einen externen Rechner, auf dem die Steuerungssoftware Saphira läuft. Mit diesem Rechner ist er über ein serielles Kabel oder Funkmodem verbunden. Es entsteht dann eine Client/Server-Architektur, wobei der Client durch die externe Steuerungssoftware und der Server durch das auf dem Roboter laufende Betriebssystem P2OS dargestellt wird. Für den in dieser Arbeit verwendeten Roboter ”R2D2“ kommt als Steuerungsrechner ein Notebook zum Einsatz, auf dem das Betriebssystem Windows 2000 von Microsoft [13] installiert ist.

In den nächsten Abschnitten werden die wichtigsten Komponenten noch einmal näher beschrieben.

2.2.1.1 Positionsbestimmung

Der Pioneer 2 DX ist mit Shaft-Encodern ausgerüstet, mit denen die Umdrehungen beider Antriebsräder ausgewertet werden. Mit Hilfe dieser Daten ist dann eine relative Positionsbestimmung möglich und es lassen sich Drehwinkel berechnen.

Auf längeren Strecken, die der Roboter zurücklegt, kommt es dann durch bauliche Schwächen zu Fehlerakkumulationen. Gerade bei ruckartigen Drehungen kommt es durch den Schlupf der Reifen zu Ungenauigkeiten, und der Roboter verliert langsam seine Orientierung. Auf kurzen Strecken ist diese Abweichung noch relativ gering. Je länger der Roboter aber unterwegs ist, desto größer wird diese Abweichung.

2.2.1.2 Sonar

Der Pioneer 2–DX ist an der Vorderseite mit einem Sonar Array, bestehend aus acht Sonars, ausgestattet. Sechs der Sonars sind nach vorne mit einem Abstand

von 20 Grad ausgerichet. Zwei weitere liegen an den Seiten des Pioneer 2–DX. Somit ist der Roboter in der Lage, einen Bereich von 180 Grad abzutasten.

Abbildung 2.2: Sonar Array

Die Sonars schicken einzelne Impulse ab und warten dann auf das erste Echo der Signale, welches entsteht, wenn die Impulse auf ein Hindernis treffen [21]. Hierbei wird die Zeit zwischen abschicken und empfangen gemessen, womit dann eine ziemlich genaue Entfernungsmessung durchgeführt werden kann.
Die Impulsrate der Sonars beträgt 25 Hz und es können mit ihnen Objekte in einem Abstand zwischen 10 und 500 cm registriert werden.

2.2.1.3 Greifer (Gripper)

Unterhalb des Sonarrings befindet sich noch ein Greifer, mit dem der Roboter Objekte aufnehmen kann. Zusätzlich zum motorbetriebenen Greifen mit den Greifpaddles kann der ganze Greifer noch in der vertikalen Richtung bewegt werden. Hiermit kann der Pioneer 2–DX dann bis zu 2kg schwere Objekte 9 cm anheben.

In den Greifpaddles selber befinden sich zwei Lichtschranken, um Gegenstände zwischen den Paddles sicher detektieren zu können. Durch Abfrage der Lichtschranke kann dann das Greifen eingeleitet werden.

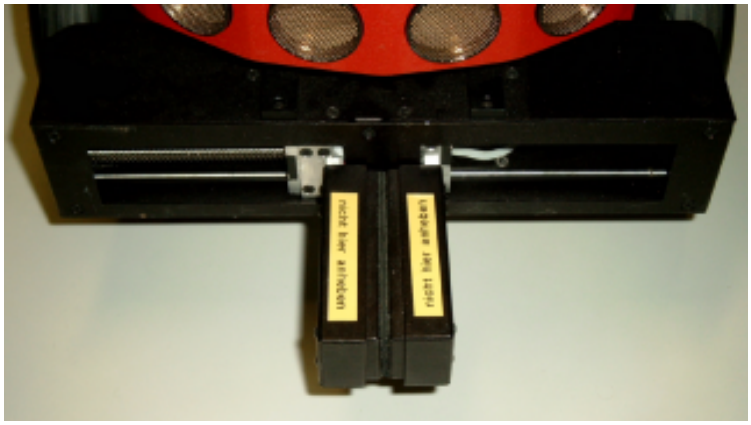

Abbildung 2.3: Greifer

2.2.1.4 Sonycamera

Eine weitere wichtige Komponente in dieser Diplomarbeit ist die auf dem Pioneer 2–DX installierte Kamera von Sony EVI-D30 [20] mit einer maximalen Auflösung von 768 x 485 Pixel. Diese besitzt eine integrierte Pan-Tilt-Einheit, mit der sich die Kamera in einem Winkel von 100 Grad in der horizontalen und 25 Grad in der vertikalen Richtung einstellen lässt. Außerdem verfügt die Kamera über einen 12fach Zoom.

Abbildung 2.4: Sony EVI-D30

Neben der Funktion zum Erkennen von farbigen, zusammenhängenden Flächen im aufgenommenen Bild, gibt es die Möglichkeit des Autotrackings. Hiermit

lassen sich schon recht gut Objekte automatisch verfolgen, wobei diese sich aber deutlich vom Hintergrund des Bildes hervorheben müssen.

2.2.1.5 Cognachrome Vision System

Die in diesem Abschnitt beschriebene Cognachrome Vision System [17] Karte von Newtonlabs [18] ist ein weiterer optionaler Bestandteil des Pioneer 2–DX. Mit dieser Karte lassen sich Bilder auf einer sehr hardwarenahen Ebene bearbeiten.

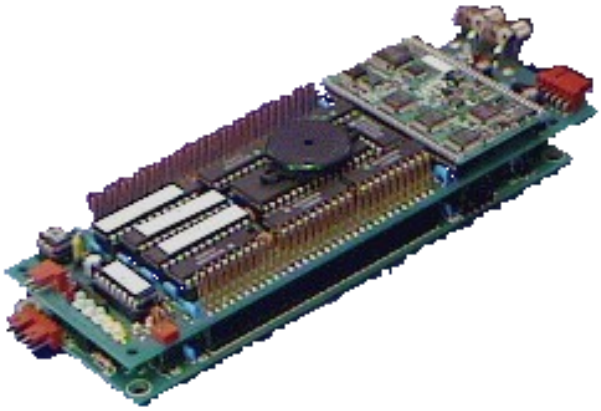

Abbildung 2.5: Cognachrome Vision System

Das System besteht zum einem aus einem Prozessor Board, welches auf einem Motorola 68332 basiert und mit 256k bestückt ist. Auf ihm sind diverse I/O Ports, sowie asynchrone und synchrone serielle Schnittstellen integriert.

Der zweite Teil ist ein Farb Prozessor Board, welches das anliegende NTSC Signal digitalisiert und ein 24Bit RGB Bild erzeugt. Mit dem System können bis zu drei Farben trainiert werden, die dann z.B. für Colortracking benutzt werden können. Das Digitalisieren der Bilder wird mit einer Frequenz von 60 Hz und für Bildgrößen von 200x250 Pixeln ausgeführt.

Mit der Entwicklungsumgebung ARC C [16] von Newtonlabs ist es möglich, eigene Programme für das Cognachrome Board zu entwickeln. Dabei ist ARC C

eine ähnliche Umgebung wie das weit verbreitete Interactive C. Bei der Programmiersprache handelt es sich hier um C, was einen einfachen Einstieg in diese Umgebung verspricht. Über die serielle Verbindung zwischen dem Host- und dem Targetsystem können die Programme dann interaktiv getestet werden. Auf der Seite von Newtonlabs stehen mehrere kostenlose Testversionen für Linux und Windows zur Verfügung, mit denen man sich einen sehr guten Überblick über dieses System verschaffen kann.

2.2.2 USB - Grabberkarte

Eine weitere Möglichkeit, Bilder einer Kamera zu verarbeiten, ist eine Framegrabberkarte in Verbindung mit einem PC. Die hier verwendete Framegrabberkarte ist von der Firma Dazzle Europe [6] und lässt sich über den USB Port an jeden beliebigen PC anschließen.

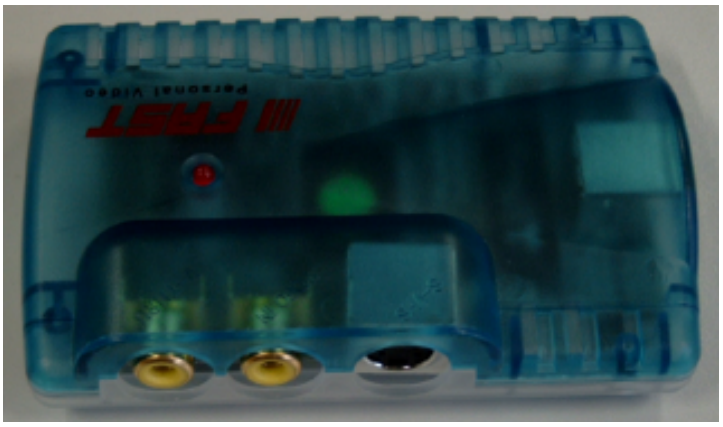

Abbildung 2.6: Dazzle Fast PV.Master

Es lassen sich dabei Einzelbilder mit einer Auflösung von bis zu 1600x1200 Pixel aufnehmen, die dann in fast jedem beliebigen Format abzulegen sind. Auch Videosequenzen lassen sich problemlos mit der Grabberkarte erzeugen, allerdings nur mit einer Größe von 176x120 Pixeln. Das soll diese Arbeit aber in keiner Weise einschränken. Da das Arbeitsnotebook nur über eine USB-1.1[19] Schnittstelle verfügt, ist die Datenrate auf 12 MBit/s beschränkt, was eine größere Auflösung von vornherein ausschließt.

Um in diesem Projekt dem großen Datenstrom und somit dem Flaschenhals entgegenzuwirken, wird die Auflösung der Kamera so gering wie möglich gehalten.

2.3 Software

2.3.1 Saphira 6.2

Maßgeblich an der Entwicklung der Systemarchitektur der Pioneer-Roboter ist Kurt Konolige[8] beteiligt. Neben der Roboterplattform hat er dabei auch ein Steuerungskonzept auf Basis einer Client/Server-Architektur entworfen. Ein Ergebnis dessen ist die bereits erwähnte Saphira-Entwicklungsumgebung [2], welche Funktionen zur Erstellung von Client-Anwendungen beinhaltet. Den Server stellt der Roboter selbst bzw. das dort laufende Betriebssystem dar. Er hat dann die Funktion einer lowlevel-Steuerung und kümmert sich z.B. um die direkte Motoransteuerung oder das Abschicken und wieder Auffangen von Sonar-Impulsen. Hierbei führt er die vom Client gewünschten Funktionen aus und stellt diesem wiederum Statusinformationen zur Verfügung. Der Client selbst braucht somit nicht die genauen Details des Roboterservers kennen, was bedeutet, dass dort Steuerungsroutinen auf einer höheren Ebene entwickelt werden können. Dadurch sind die Client-Programme theoretisch unabhängig von der eigentlichen Roboterhardware. Genutzt wird diese Unabhängigkeit beim im Saphira-Paket enthaltenen Pioneer-Simulator.

Saphira ist für verschiedene Betriebssysteme erhältlich und wird standardmäßig mit der in Abbildung 2.7 zu sehenden Oberfläche ausgeliefert. In der Oberfläche werden die Positionen des Roboters und der erkannten bzw. definierten Objekte (Artefakte) sowie die Sonarmessungen grafisch dargestellt. Zusätzlich enthält sie einen Kommandozeilen-Interpreter, über den direkt Steuerungsbefehle an den Roboter gesendet werden können.

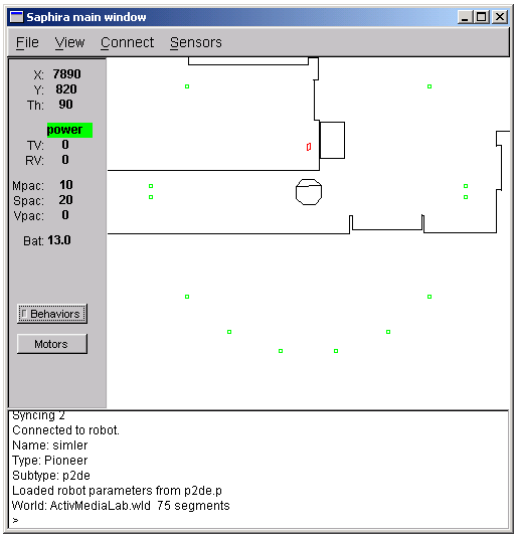

Abbildung 2.7: Saphira 6.2 Benutzeroberfläche

2.3.1.1 Architektur

Das Saphira-System kann man sich als aus zwei übereinander liegenden Architekturen zusammengesetzt vorstellen. Die untere Ebene ist dabei die Saphira-System-Architektur und die obere die Roboter-Kontroll-Architektur (näheres dazu im Saphira-Manual [2]).

In der Saphira-System-Architektur werden dabei die grundlegenden Funktionen zur Roboteransteuerung zur Verfügung gestellt. Hier befindet sich zunächst ein synchrones Betriebssystem, das die registrierten saphira-eigenen und benutzerdefinierten Routinen in einem 100 ms-Zyklus verwaltet. Die hiervon verwalteten Routinen haben dabei die Struktur von endlichen Automaten.

Neben diesen synchronisierten Routinen ist es auch möglich, vom Saphira-Zyklus asynchron laufende Funktionen zu starten, um beispielsweise komplexere Planungsaufgaben erledigen zu können. Des Weiteren wird von dieser Ebene die gesamte Paket-Kommunikation mit dem Roboter-Server gesteuert.

Als drittes befindet sich hier auch der sog. State-Reflector, der den momentanen Status des Roboters (z.B. Sensor-, Bewegungs- und Positionsinformationen)

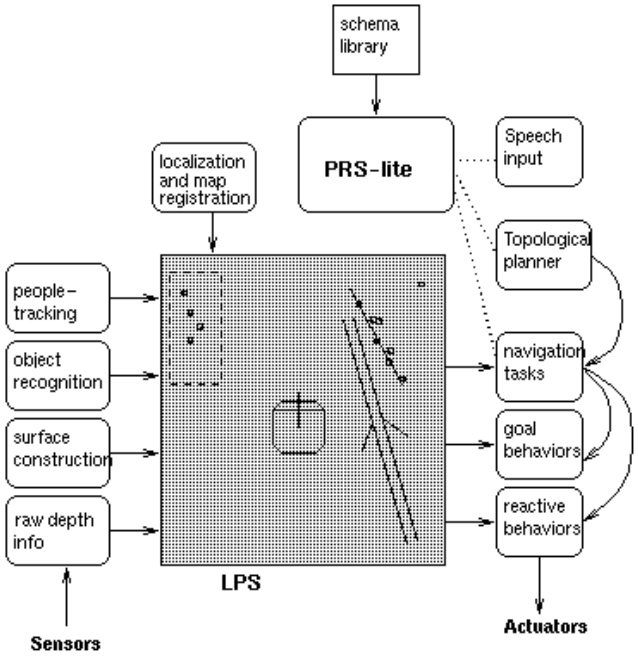

Abbildung 2.8: Saphira 6.2 Architektur

auf dem Client zur Verfügung stellt und sich um das Aktualisieren der Werte kümmert. Hier können dann auch motion setpoints definiert werden, die zur Steuerung des Roboters notwendig sind und zu ihm übertragen werden. Es kann z.B. ein Wert für die Geschwindigkeit festgelegt werden, wobei von der Steuerung dann versucht wird, diesen Wert zu erreichen (beispielsweise durch Beschleunigen).

Über dem State-Reflector setzt nun die Roboter Kontrollarchitektur auf. Dort befinden sich Routinen zur Verarbeitung von Sensordaten sowie Routinen zur Steuerung des Roboters. Erstere können noch unterteilt werden in Funktionen zur Interpretation von Sensordaten, Funktionen zur Aktualisierung der Roboterposition in der Karte und Funktionen zur grafischen Darstellung der Ergebnisse in Saphira.

Saphira arbeitet hierbei mit zwei verschiedenen Koordinatensystemen, dem Local Perceptual Space und dem Global Map Space. Ersteres ist eine vom Roboter aus egozentrische (lokale) Karte mit geringen Ausmaßen, während Zweiteres eine globale Karte der Welt darstellt. Vom Roboter mit den Sensoren erfaßte und von

den Interpretierungsroutinen klassifizierte Objekte können dann in diese Karte als Artefakte eingetragen werden. Für diese Artefakte gibt es bestimmte vordefinierte Typen, z.B. Punkte, Korridore, Türen. Des Weiteren ist es möglich, in Saphira sog. Map-Dateien zu laden, die bereits eine Menge von Artefakten enthalten und eine Karte einer bestimmten Umgebung, beispielsweise eines Raumes, darstellen. Der Roboter kann diese Karte dann als Navigationshilfe benutzen und versuchen, die von ihm erkannten Objekte mit den in der Karte enthaltenen abzugleichen und seine Position im Raum festzustellen.

Neben den sensorabhängigen Routinen befinden sich in der Kontroll-Architektur noch die von Saphira benötigten Routinen, die zur Verwaltung von Steuerungsprogrammen notwendig sind. Grundsätzlich gibt es in Saphira zwei Arten von Programmen, die erstellt werden können, Behaviors und Activities. Behaviors werden dabei von einem Fuzzy- Controller gesteuert, während Activities von der Colbert-Executive verwaltet werden. Es können in Saphira beliebig viele Activities und Behaviors gestartet werden, die dann parallel abgearbeitet werden. Beide Arten unterliegen dann aber dem 100 ms Saphira-Zyklus, d.h. alle 100 ms wird das entsprechende Programm aufgerufen und kann seine Abarbeitung für eine bestimmte Zeit fortsetzen.

2.3.1.2 Behaviors

Behaviors sind einfache kleine Steuerungsprogramme, die mit Fuzzy-Regeln implementiert werden und jeweils nur eine bestimmte Aufgabe erfüllen. Einige Behaviors werden mit Saphira schon mitgeliefert, z.B. zur Kollisionsvermeidung. Eine Besonderheit ist, dass Behaviors mit einer Priorität versehen werden können, welche bei der Auswertung der Fuzzy-Regeln vom Fuzzy-Controller berücksichtigt wird. Behaviors können allerdings keine anderen Programme aufrufen.

2.3.1.3 Activities

Als Zweites können in Saphira noch Activities definiert werden. Diese sind dann auch für komplexere Steuerungs- und Planungsaufgaben geeignet. Erstellt werden

Activities in einer in Saphira definierten Programmiersprache mit dem Namen COLBERT. COLBERT ist an die Sprache C angelehnt, bietet aber nur einen Teil der dort enthaltenen Sprachkonstrukte. Die grundlegende Struktur der Activities basiert auf endlichen Automaten (finite state machines).

Im Gegensatz zu Behaviors können Activities andere Activities oder Behaviors aufrufen, besitzen allerdings keine Prioritäten. Zusätzlich ist es noch möglich, von COLBERT aus externe Bibliotheksfunktionen aufzurufen, die für MS Windows beispielsweise in Dynamic Link Libraries (DLLs) zur Verfügung gestellt werden. Dieser Vorgang ist geeignet, um aufwendige Berechnungen auszulagern (da Activities zunächst nicht kompiliert, sondern von der Colbert Executive interpretiert werden) oder die in COLBERT enthaltenen Einschränkungen zu umgehen. Zum Beispiel sind die zur Ansteuerung von Greifer und Kamera zur Verfügung gestellten Funktionen in solchen Bibliotheken untergebracht.

2.3.2 Visual C++ 6

Visual C++ ist eine Entwicklungsumgebung von Microsoft, die zum Erstellen von MS Windows-Anwendungen geeignet ist. Dazu stellt sie z.B. die Microsoft Foundation Class Library [14] bereit, die die entsprechenden benötigten Elemente (Fenster, Schaltflächen...) in C++ Klassen verpackt. Zum Einsatz für die Programmierung des Pioneer 2 kommt diese Umgebung bei der Erstellung von in Saphira ladbaren Bibliotheken, die z.B. rechenaufwendige Funktionen enthalten. Diese sind dann allerdings in Standard C zu programmieren. Des Weiteren wurde mit dieser Umgebung und unter Verwendung der MFC auch die Bildverarbeitungssoftware erstellt.

2.3.3 Video OCX (SDK)

Um die mit der Framegrabber ”gegrabbten“ Bilder in den PC zu bekommen und verarbeiten zu können, wird eine SDK benötigt. Die hier verwendete VideoOCX [12] gibt es als kostenlose Probeversion auf der Homepage des Herstellers.

Die SDK stellt zum einen einfache Befehle zur Verfügung, mit der die Bilder in den Speicher des Rechners gebracht werden können. Zum anderen gibt es schon ein paar Befehle zur weiteren Verarbeitung der Bilder, wie z.B. Binarisierung und Kantenfinden. Die vorhandenen Befehle lassen sich dann sehr einfach für eigene Grafik Routinen verwenden.

2.3.4 Betriebssysteme

Als Betriebssystem für den Steuerrechner kommt Windows 2000 zum Einsatz, da alle weiteren Tools wie SDK, Saphira und Visual C++ unter diesem Betriebssystem laufen. Die ausschlaggebenden Gründe liegen aber an der Framegrabber Karte, da diese nur Treiber für das Microsoft Betriebssystem zur Verfügung stellt und die USB Unterstützung von Linux immer noch nicht so ganz ausgereift scheint.

Auf dem Pioneer ist als Betriebssystem die aktuellste Version 1.J des P2OS installiert, wobei die Versionsnummer aber nicht weiter ins Gewicht fällt, da es hier keine wesentlichen, für die Diplomarbeit wichtigen Verbesserungen in den einzelnen Versionen gibt.

Kapitel 3

Anforderungen an das System

Ein wesentliches Merkmal von mobilen Robotern ist, dass sie sich in ihrer Umgebung bewegen. Für mit Kamera ausgestattete Roboter führt dies dazu, dass in der Bildverarbeitung ständig neue Bilder mit veränderten Objektpositionen zu betrachten sind. Die Folge davon kann sein, dass die Bildverabeitungskomponente gerade ein Bild analysiert und die Daten an die Steuerungskomponente übergeben hat, inzwischen sich der Roboter aber schon soweit bewegt hat, dass die erarbeiteten Daten nicht mehr aktuell sind und eine in der neuen Situation falsche Reaktion bewirken. Die Bildverarbeitung steht also vor dem Problem, dass sie zum einen richtig funktionieren soll, also alle wichtigen Objekte erkennt, und zum anderen, dass sie auch in einer ausreichenden Geschwindigkeit arbeitet, so dass das Robotersteuerungsprogramm mit den von der Bildverarbeitung gelieferten Daten auch noch etwas anfangen kann.

Es heißt also oftmals, Kompromisse zu schließen. Die Fahrgeschwindigkeit des Roboters sowie die Bewegungen der Kamera müssen auf die Bildverarbeitungsdauer abgestimmt werden. Um nun aber gerade bei komplexeren Bildanalysen den Roboter nicht im Schneckentempo navigieren zu lassen, muss die Bildverarbeitung noch auf Geschwindigkeit hin optimiert werden. Dies wird aber meist auf Kosten der Genauigkeit geschehen. Einfache Möglichkeiten dazu sind z.B. die Verwendung von Grauwert- anstatt von Farbbildern oder eine Verringerung der Auflösung.

Am Ende soll eine Anwendung entstehen, die mit einer relativ geringen Anzahl von verarbeiteten Bildern eine stabile Umgebung für den Roboter bietet. Gehen wir zum Beispiel davon aus, dass wir pro Sekunde ein Bild verarbeiten, der Roboter in der Zeit aber schon einen Meter gefahren ist, dann ist die Wahrscheinlichkeit sehr groß, dass wichtige Details in der Umwelt verloren gehen. Schaffen wir aber fünf Bilder pro Sekunde und lassen den Roboter nur mit halber Geschwindigkeit fahren, erhöht sich die Aufnahmefähigkeit des Roboters um ein 10faches. Es gilt also hier eine gesunde Mischung zwischen maximaler Geschwindigkeit und höchstmöglicher Trefferwahrscheinlichkeit zu finden.

Aus diesen Anforderungen läßt sich jetzt zum Beispiel folgendes Szenario entwickeln.

3.1 Szenario

Wenn man sich einige der Vorlesungsräume im Hochhaus am Berliner Tor ansieht, könnte man denken, die Mensa sei umgezogen. Es stehen dort teilweise ganze Tische mit Bechern, Tassen und Tellern herum. Um dieser „Geschirrflut“ ein wenig entgegen zu wirken, wäre es praktisch, man könnte einen „stummen Diener“ los schicken, der dann zum Beispiel nur die weißen Becher einsammelt, die farblosen bis hellgrauen Plastikbecher aber stehen lässt. Diese beiden Objekte lassen sich dann nur noch über die Form, aber schwierig über die Farbe unterscheiden.

Erkannte Becher werden anschließend zu einem ihm zugeteilten Ablageplatz gebracht. Dieses kann auf mehrere Möglichkeiten passieren. Zum einen kann der Roboter sich über die Sensorik wie Sonar und Shaftencoder orientieren, so dass er sich zu seinem Ziel hin tasten muß. Den direkten Weg zu seinem Ziel wird er dabei mit höchster Wahrscheinlichkeit nicht nehmen, da er nicht weiß, wo das gefundene Objekt tatsächlich liegt. Eine andere Methode ist das Ausnutzen der Odometrie, mit der sich der Roboter über eine interne Karte orientieren kann. Eine Anwendung hierfür wird in der Diplomarbeit von Jörg Grawe und Oliver Groth [7] näher beschrieben, die sich mit Garbage-Collection in partiell bekannten Räumen befasst.

3.2 Aufgabenstellung

Aus dem im vorigen Abschnitt beschriebenen Szenario ergibt sich nun folgende Aufgabenstellung:

Zum einen soll der Roboter bzw. die Kamerasoftware in der Lage sein, mehrere beliebige Objekte zu erlernen und sich diese in einer bestimmten Datenstruktur zu merken. Diese Objekte sollen dann so weit zu unterscheiden sein, dass man ohne Probleme zwei Objekte mit gleicher Farbe, aber unterschiedlicher Form trennen kann. Zum anderen soll es jetzt möglich sein, diese Objekte mit einer relativ hohen Trefferrate wiederzufinden bzw. wiederzuerkennen.

Die erkannten Objekte sollen dann nach einer sicheren Detektierung zu einem, dem Gegenstand zugeordneten Platz gebracht werden. Das Sortieren der Objekte soll dabei nur der Darstellung dienen, dass ein Objekt richtig erkannt und zugeordnet wurde, wobei das Ziel (eine farbige Pappe) über die Kamera gesucht wird.

Kapitel 4

Bildverarbeitungsverfahren

4.1 Zwei Hypothesen

Die Bilderkennung für die Pioneer-Roboter ist im Zusammenhang mit der Sonycamera und der Cognachrome-Karte schon recht gut. Diese beiden Systeme basieren dabei im Allgemeinen auf dem Prinzip der Farberkennung, d.h. es werden Farbschwerpunkte im Bild ermittelt und für die weitere Verarbeitung herangezogen. Allerdings lässt die Zuverlässigkeit in Sachen Trefferrate doch sehr zu wünschen übrig. Gegenstände mit ähnlicher Farbe lassen sich sehr schwer unterscheiden. So soll bei einer Vorführung ein Roboter, statt sich auf die Suche nach roten Bechern zu machen, auf die Garderobe des Publikums „gestürzt“ haben.

Um diese Fehlinterpretationen von gefundenen Objekten zu minimieren, scheint es sinnvoll zu sein, die Erkennung nicht auf einem Verfahren, wie die Farberkennung, beruhen zu lassen, sondern noch eine weitere Eigenschaft des Objektes hinzu zu nehmen. Aus diesen einzelnen Hypothesen lassen sich dann bessere Rückschlüsse auf ein eventuell gefundenes Objekt ziehen. In den folgenden Abschnitten werden die Farberkennung und Objektdetektion in Bildern erläutert.

4.2 Bildverarbeitung

4.2.1 Digitale Bilder

Dieser Abschnitt ist aus der Diplomarbeit von Rainer Balzerowski [3] entnommen. Digitale Bilder werden durch eine Matrix mit Spalten und Zeilen repräsentiert. Die Elemente dieser Matrix sind Bildpunkte, die üblicherweise als Pixel bezeichnet werden. Dieses Kunstwort steht für „Picture Element“ und kennzeichnet die kleinste, sichtbare Einheit eines digitalen Bildes. Das einfachste Bild einer festen Größe ist ein Schwarz–Weiß–Bild. Bei einem solchen Bild benötigt ein Pixel genau 1 Bit. Dieses Bit kann dann die Werte 0 (Schwarz) oder 1 (Weiß) annehmen. Diese Bilder werden auch als Binärbilder bezeichnet.

Die Anzahl der Bits pro Pixel bezeichnet man als Farbtiefe (auch wenn dieser Begriff bei Schwarz–Weiß–Bildern nicht angebracht scheint).

Nimmt man 8 Bit pro Pixel an, so kann ein Pixel 256 verschiedene Grauwerte annehmen. Hier repräsentiert 0 die Farbe Schwarz, 255 die Farbe Weiß und alle Werte dazwischen repräsentieren Grautöne.

Um Farbbilder darstellen zu können, benötigt man Mehrkanalbilder. Diese Bilder bestehen in der Regel aus drei einfarbigen Einzelbildern, die zu einem Farbbild gemischt werden, wie in Abbildung 4.1 zu sehen ist.

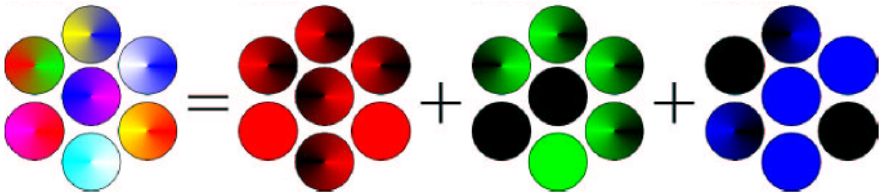

Abbildung 4.1: RGB Mischfarben

Nimmt man für jedes der Einzelbilder 8 Bit Farbtiefe an, so erhält man für das Gesamtbild eine Farbtiefe von 24 Bit. Mit 24 Bit pro Pixel lassen sich $2^{24} = 16777216$, also über 16 Millionen Farben für jedes einzelne Pixel darstellen. Allerdings benötigt ein 24 Bit Farbbild aber auch vierundzwanzig mal mehr Speicher als ein Schwarz–Weiß–Bild gleicher Größe.

Die in den digitalen Bildern zu findenden Objekte werden in dieser Arbeit jetzt durch zwei Eigenschaften beschrieben. Ein Merkmal, welches auch gleichzeitig das Markantere der Beiden ist, ist die Farbe eines Objektes. Das zweite Merkmal ist die Form, in diesem Fall das Verhältnis zwischen Höhe und Breite, eines Objektes. Wenn diese beiden Features in einem Bild gefunden werden, soll die Detektion als erfolgreich gelten. Was letztendlich damit gemeint ist, wird in der Abbildung 4.2 deutlich.

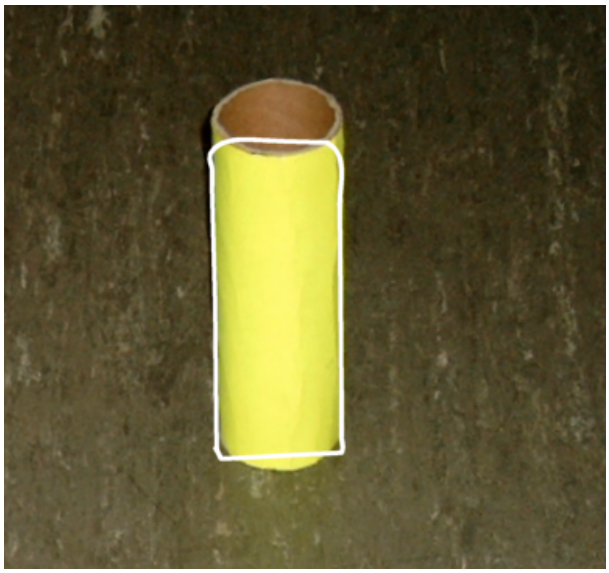

Abbildung 4.2: Beispiel einer Farb- und Formdetektion

In den folgenden Abschnitten werden mehrere Varianten von Farbräumen und ihre Vor- und Nachteile erläutert, ebenso ein Verfahren zur Beschreibung der Form eines Objektes.

4.2.2 Farbräume

Wenn man von der Farbe rot spricht, denkt man sofort an ein leuchtendes Rot. Dabei darf man nicht vergessen, dass es sehr viele Abstufungen von der Farbe Rot gibt, die zum Beispiel durch Lichteinfall sehr stark beeinflusst werden können. Hier gilt es einen Weg zu finden, diese Farbnuancen mit aufnehmen zu können, ohne irgendwelche Bildverluste zu haben.

Hinzu kommt noch, dass die Welt, in der wir leben, eine dreidimensionale Welt ist. Für uns ist dieses völlig normal. Eine Kamera kann im Normalfall diese Welt aber nur zweidimensional aufnehmen, d.h. die Tiefen-Informationen gehen dabei verloren. Man könnte also nicht mehr unterscheiden, ob ein Objekt senkrecht steht oder mit der Oberkante nach vorne geneigt ist. Dabei treten dann schnell Schatten auf, die das Objekt zwar immer noch in der eigenen Farbe erscheinen lassen, für das Erkennen kann es aber zu erheblichen Problemen führen.

Nun gibt es die Möglichkeit, Bilder in einen anderen, besser beschreibenden Farbraum umzuwandeln und hiermit die Berechnungen durchzuführen.

Es ist erstaunlich, dass beinah jede Farbe, die wir sehen können, in eine Mischung von nur drei Farben zerlegt werden kann. Die meisten Kinder lernen dies in der Schule und wissen, dass die Grundfarben blau, gelb und rot sind. Mit Fingerfarben versuchen die Kinder die sogenannten Sekundärfarben grün, orange und violett zu erzeugen. Wenn das Hören auf diese Art funktionieren würde, könnte man jede Musiknote in die Summe aus drei Tönen zerlegen. Drei Töne ergeben einen schönen Gitarrenakkord, aber es reicht nicht für Rock'n Roll.

Interessanterweise reichen die drei Farben dank der Funktionsweise des menschlichen Auges aus. Es gibt drei verschiedene Zapfenarten im Auge (rot, grün und blau); das erklärt, warum nur drei Farben benötigt werden. Dass alle Farben durch das Mischen der drei Farben erzeugt werden können, ist extrem nützlich. Es ist wie ein Alphabet. Mit einer kleinen Sammlung von Buchstaben kann jedes Wort gemacht und jede Bedeutung enthüllt werden. Für Farben gilt dasselbe. Mit nur drei Farben kann ein Monitor prinzipiell jede Farbe darstellen. Mit nur drei Tinten kann ein Drucker prinzipiell jede Farbe drucken.

Das Konzept der Erzeugung aller Farben aus drei Grundfarben hat ein paar Begrenzungen in der Praxis. Ohne ins Detail zu gehen, genügt es jedoch zu wissen, dass die Menge von Farben, die durch das Hinzufügen verschiedener Mengen der drei Grundfarben erzeugt werden kann, ein Farbraum genannt wird. Die Form eines auf diese Weise definierten Raums ist immer ein Würfel.

Prinzipiell sind die drei Grundfarben zum Erzeugen eines Farbraums nicht

einheitlich. In der Theorie kann jedes Tripel von unabhängigen Farben zum Zerlegen eines Farbraums genutzt werden. In der Praxis jedoch sind Farbräume, die aus zwei unterschiedlichen Mengen von Grundfarben erzeugt werden, nicht identisch. Typischerweise gibt es einen Teil eines Farbraums, der nicht durch den anderen dargestellt werden kann.

Die beste Wahl der Grundfarben hängt gewöhnlich von dem physischen Gerät ab, das die Farben erzeugen soll. Monitore nutzen rot, grün und blau, um einen RGB genannten Farbraum zu erzeugen. Drucker nutzen Cyan, Magenta und Gelb und arbeiten in einem CMY genannten Farbraum. Die physische Art der Zusammensetzung der Farben erklärt die zwei Farbräume. Monitore senden Licht, während Tinten Licht reflektieren. Wenn wir ein weißes Licht auf Tinte strahlen, sehen wir die Farbkomponenten des weißen Lichts, die reflektiert und nicht von der Tinte absorbiert werden.

Der RGB-Farbraum und der für die Wahrnehmung wichtige HSV-Farbraum werden in allen Einzelheiten beschrieben und werden von einer Beschreibung der Beziehung zwischen RGB und HSV gefolgt. Der für den Druck wichtige Farbraum CYMK wird hier nur ansatzweise erläutert.

4.2.2.1 Der RGB Farbraum

Der RGB-Farbraum ist für uns wichtig, weil er das ist, was wir sehen, wenn wir auf den Monitor schauen. Der Monitor mischt verschiedene Mengen der Grundfarben Rot, Grün und Blau. Deswegen können die am Computermonitor sichtbaren Farben durch ein Tripel von Zahlen oder als Vektoren in einem dreidimensionalen Würfel abgebildet werden. Der Wert jeder Grundfarbe kann nur in einem festen Bereich abgebildet werden. Pro Grundfarbe werden 8 Bits benutzt, somit ist der Bereich pro Grundfarbe von 0 bis 255. Hier bedeutet 0, dass keine Grundfarbe genutzt wird und 255 bedeutet, dass soviel Grundfarbe benutzt wird, wie der Monitor überhaupt erzeugen kann. Obwohl es Punkte wie Weissabgleich, Eichung und menschliche Wahrnehmung gibt, wird angenommen, dass die drei Grundfarben bei voller Stärke weiss erzeugen und bei Stärke 0 schwarz erzeugen. Natürlich ist das erzeugte Schwarz nicht dunkler als der ausgeschaltete Monitor.

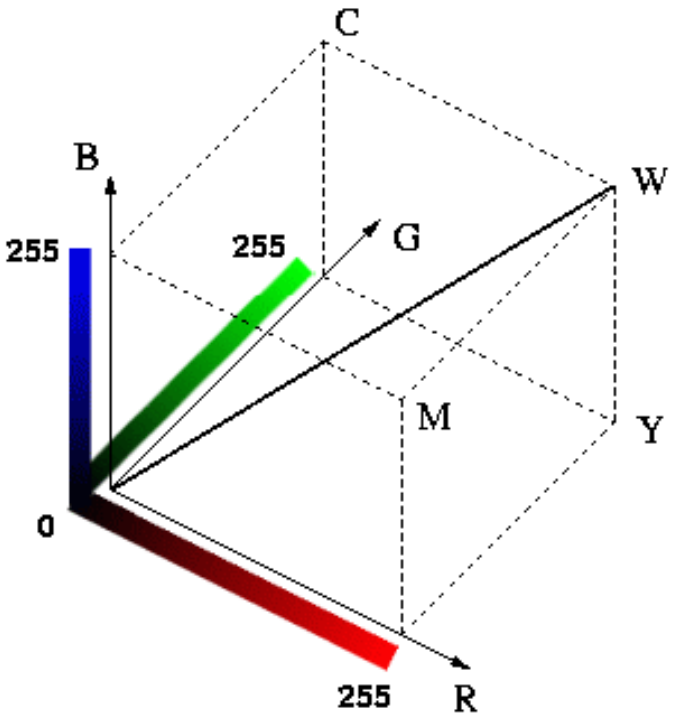

Abbildung 4.3: RGB Farbmodell

Jede Würfelachse stellt einen Wert von Rot, Grün oder Blau im Bereich [0,255] dar. Die rote Achse, mit R bezeichnet, zeigt die zugehörige Farbskala darunter. Die grüne und die blaue Achse sind ähnlich dargestellt. Wie vorher erwähnt, bedeutet 0, dass der Monitor keine Grundfarbe ausstrahlt und 255 maximale Strahlung. Werte zwischen 0 und 255 bringen Abstufungen der Farbstärke hervor.

Wie das Wort Farbraum andeutet, verhalten sich die Farben wie Vektoren. Sie können durch Addition und Subtraktion zum Erzeugen anderer Farben im Würfel verknüpft werden. Also stellt der Ursprung des Würfels $0^R\ 0^G\ 0^B$ die totale Abwesenheit von Farbe dar, d.h. schwarz. Die entfernteste Ecke vom Ursprung ist die Summe der größten roten, grünen oder blauen Farbstärken bzw. $255^R\ 255^G\ 255^B$. Das erzeugt die Farbe Weiss, weshalb diese Ecke mit W bezeichnet ist. Die anderen Ecken des Würfels stellen die verschiedenen Grund- und Sekundärfarben dar. Wir haben schon Rot, Grün und Blau erwähnt. Die Restlichen sind Cyan, Magenta und Gelb.

	R	G	B
Rot	255	0	0
Grün	0	255	0
Blau	0	0	255
Cyan	0	255	255
Magenta	255	0	255
Gelb	255	255	0
Schwarz	0	0	0
Weiss	255	255	255

Tabelle 4.1: 8 Ecken des RGB-Würfels

Die Hauptdiagonale des RGB-Würfels ist als eine Linie zwischen dem schwarzen Ursprung und Weiss gezogen. Diese Linie stellt die Farben des RGB-Würfels mit den gleichen Farbwerten für Rot, Grün und Blau dar. Alle diese Punkte auf dieser Linie sind grau. Je näher am Ursprung, desto dunkler ist das Grau, je näher am W, desto heller. Aus diesem Grund wird die Hauptdiagonale im RGB-Würfel als neutrale Achse bezeichnet. Grau ist neutral, weil es keinen Farbwert begünstigt; es beinhaltet gleiche Werte für Rot, Grün und Blau.

4.2.2.2 Der HSV Farbraum

Die Wahrnehmung von Farben und die Art, wie wir darüber im Alltag reden, wird durch den RGB-Farbraum nicht gut unterstützt. Wenn wir zum Beispiel an das Anstreichen von Wänden im Wohnzimmer denken, grübeln wir darüber, welche Farbschattierung es sein sollte, wie hell wir es wollen und ob wir es in Pastell oder lebhaft haben wollen.

Das erste, was wir normalerweise bei einer Farbe bemerken ist ihr Farbwert. Der Farbwert beschreibt die Farbschattierung und wo die Farbe im Farbspektrum zu finden ist. Rot, Gelb und Purpur sind Worte, die eine Farbe beschreiben.

Abbildung 4.4 stellt das HSV-Farbmodell als einen Kegel dar. Die Gründe hierfür werden schnell klar werden.

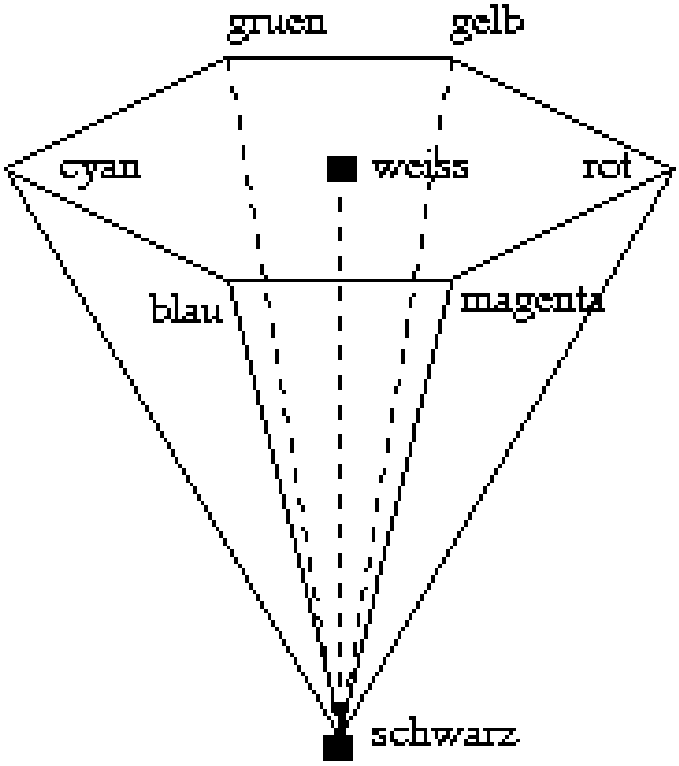

Abbildung 4.4: HSV Farbmodell

Der nächste wesentliche Aspekt einer Farbe ist normalerweise die Sättigung S. Die Sättigung beschreibt, wie rein ein Farbwert in Bezug auf weiss ist. Zum Beispiel ist eine ganz rote Farbe ohne weiss ganz gesättigt. Wenn wir etwas Weiss dem Rot hinzufügen, wird das Ergebnis mehr pastellfarben und die Farbe verschiebt sich nach rosa. Der Farbwert ist immer noch Rot, aber er ist weniger gesättigt. Dies wird mit der senkrechten Leiste in Abbildung 4.4 dargestellt. Sättigung ist ein Prozentwert zwischen 0 und 100. Ein reines Rot hat kein Weiss und ist 100 Prozent gesättigt.

Schliesslich hat Farbe auch eine Helligkeit. Dies ist eine ungefähre Beschreibung, wie viel Licht aus der Farbe kommt. Wenn die Farbe sehr viel Licht zurückwirft, würden wir sie als hell bezeichnen. Stelle man sich einen roten Sportwagen tagsüber vor. Seine Farbe sieht hell aus. Vergleicht man das mit der Wahrnehmung des Wagens, wenn die Nacht hereinbricht. Der Wagen ist immer noch rot, aber er sieht matter aus, weil die umgebende Beleuchtung weniger Licht auf das Auge

zurückwirft. Weniger Licht bedeutet, dass die Farbe dunkler aussieht. Die horizontale Leiste in Abbildung 4.4 zeigt einen Bereich von roten Werten. Helligkeit wird wie Sättigung als Prozentwert zwischen 0 und 100 angeben. Dieser Bereich kann als der Lichtbetrag gesehen werden, der die Farbe beleuchtet. Wenn zum Beispiel der Farbwert Rot und die Helligkeit hoch ist, sieht die Farbe hell aus. Wenn die Helligkeit gering ist, sieht die Farbe dunkel aus.

So gesehen sind Farbwert, Sättigung und Helligkeit ein alternativer Farbraum. Jede Farbe kann in diese drei Anteile zerlegt werden und wie bei RGB ist es möglich, diesen Raum als Würfel darzustellen.

Weil sich die vom Monitor erzeugten Farben rot, grün und blau mischen, ist es nützlich und lehrreich zu wissen, wie der HSV-Farbraum innerhalb des RGB-Raums dargestellt wird. Darauf wird in einem späteren Abschnitt genauer eingegangen.

4.2.2.3 Beziehung zwischen RGB und HSV Farbraum

Zum besseren Verständnis der HSV- und RGB-Farbräume hilft es, ihre Beziehung zueinander zu verstehen. Eine sehr sinnvolle Übung ist das Identifizieren der Teile des RGB-Würfels, dessen Oberflächen mit konstantem Farbwert, Sättigung und Helligkeit entsprechen.

Wir beginnen am besten mit der Bestimmung der Helligkeit. Dieser Begriff wurde im vorherigen Abschnitt eingeführt und ist ein Mass der sichtbaren Wahrnehmung dafür, wieviel Licht ein Bereich ausstrahlt. Der Begriff Helligkeit hängt von der Farbe des Lichts ab. Es ergibt gleichviel Sinn, von der Helligkeit einer roten Lampe wie von der einer weissen Lampe zu sprechen.

Ein Farbmonitor ist lediglich eine Ansammlung von tausenden Lampen (d.h. Pixeln), die rotes, grünes und blaues Licht ausstrahlen. Weil jede dieser Lampen unabhängig geregelt werden kann, ist die gesamte Helligkeit eines Monitorbereichs die Summe der Helligkeiten jeder Farbe. Farben mit der gleichen Helligkeit im RGB-Würfel sind dann solche, deren drei Farbanteile sich zum gleichen Wert

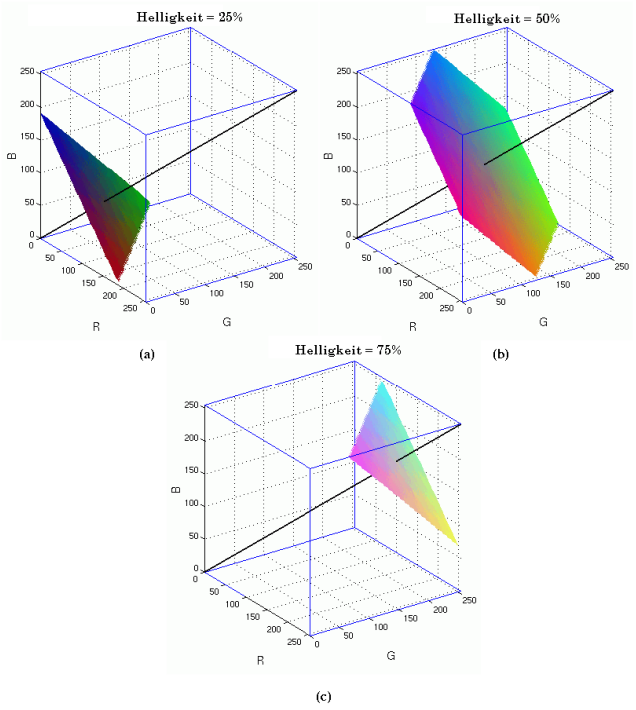

Abbildung 4.5: RGB Würfel

summieren. Für eine bestimmte Ebene der Helligkeit wird dies durch eine Ebene senkrecht zur neutralen Achse dargestellt.

Abbildung 4.5 zeigt drei Beispielebenen senkrecht zur neutralen Achse des RGB-Würfels. Der Würfel in Abbildung 4.5(a) zeigt die dunkelste Ebene, der Würfel in Abbildung 4.5(c) die hellste und der Würfel in Abbildung 4.5(b) eine Ebene mit dazwischenliegender Helligkeit. Beachten Sie, dass die Farben in jeder Ebene gleich hell erscheinen. Die drei Würfel in der Abbildung zeigen Ebenen mit einer Helligkeit von 25, 50 und 75%. Reines Weiss und reines Schwarz haben eine Helligkeit von 0% bzw. 100% und bestehen aus einem einzigen Punkt im Würfel.

Es gibt viel Konzepte, die sich auf Helligkeit beziehen und die verschiedene nützliche Eigenschaften haben. Helligkeit wird formal mit (R+G+B)/3 festgelegt; sie ist eine physische Eigenschaft der Farbe und entspricht der menschlichen

Wahrnehmung nicht sehr gut.

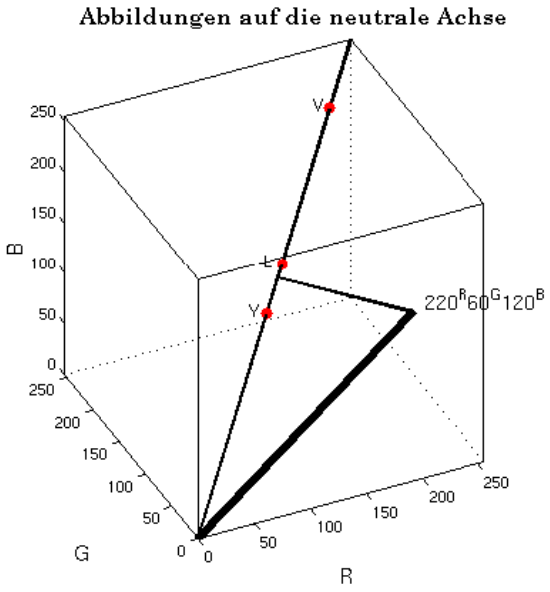

Abbildung 4.6: Neutrale Achse

Abbildung 4.6 zeigt, wie die verschiedenen Definitionen der Helligkeit auf der neutralen Achse abgebildet werden. Die Abbildung zeigt die Stelle im RGB-Würfel einer Farbe, 220^R 60^G 120^B und den jeweils entsprechenden Wert für Value (Helligkeit) V, Beleuchtung L und Leuchtkraft Y. Wie man in Abbildung 4.6 sehen kann, ist Value am hellsten. Die Leuchtkraft und die Beleuchtung sind unbeständig und können je nach Wahl des RGB-Tripel am dunkelsten sein.

Die Definition der Sättigung bezieht sich auf die der Helligkeit. Die Sättigung gibt die relative Farbfülle in Bezug auf die Helligkeit wieder. Was ist Farbfülle? Eine Antwort kann im Zusammenhang mit dem RGB-Würfel und der neutralen Achse gegeben werden. Wie bereits besprochen, ist die neutrale Achse die Diagonale im Würfel von 0^R 0^G 0^B bis 255^R 255^G 255^B d.h. schwarz und weiss und die Graustufen dazwischen. Deshalb hat die Achse im gewöhnlichen Sinn keine Farbe (bzw. Farbwert). Die Farbfülle jedes Punkts steht deshalb in einem Verhältnis zum senkrechten Abstand zur neutralen Achse. Punkte, die näher an der Ach-

se liegen, sind weniger farbig (d.h. näher an Grau) und weiter entfernte Punkte sind farbiger. Die Sättigung eines Punkts im RGB-Würfel ist dann das Verhältnis seiner Farbfülle zu seiner Helligkeit. Dies bedeutet, dass Oberflächen mit einer gleichbleibenden Sättigung im RGB-Würfel Kegel um die neutrale Achse bilden.

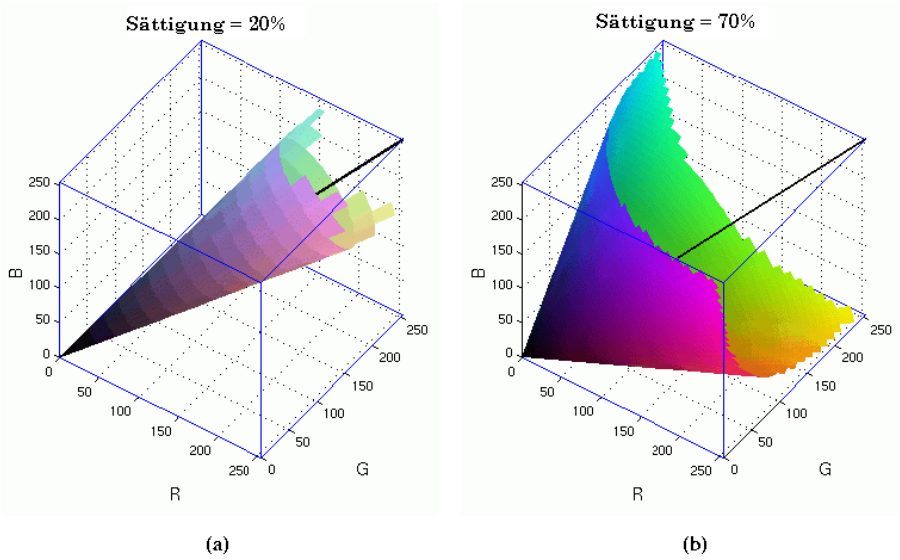

Abbildung 4.7: Kegel gleichbleibender Helligkeit

Abbildung 4.7 zeigt zwei RGB-Würfel. Der Würfel in Abbildung 4.7(a) zeigt den zu einer Sättigung von 20% gehörigen Kegel und Abbildung 4.7(b) zeigt den zu einer Sättigung von 70% gehörigen Kegel. Die Farben des rechts befindlichen Kegels machen aufgrund der stärkeren Sättigung den lebhafteren Eindruck. Die Farben im linken Kegel sind aufgrund der relativ neutralen Farben viel blasser. Eine neutralere Farbe nennt man Pastell.

Schliesslich ist es die Definition von der Farbe bezogen auf das, was wir umgangssprachlich unter Farbe verstehen. Die Farbe eines Punkts im RGB-Würfel wird durch den Winkel rund um die neutrale Achse bestimmt. Sieht man in die in Abbildung 4.2.2.1 gezeigten Ecken des RGB-Würfels, kann man sehen, dass Rot, Gelb, Grün, Cyan, Blau und Magenta gleichmässig im Winkel zur neutralen Achse verteilt sind. Deshalb ist die durch die neutrale Achse und jeden Punkt auf der Oberfläche des Würfels definierte Fläche eine Ebene mit demselben Farbwert.

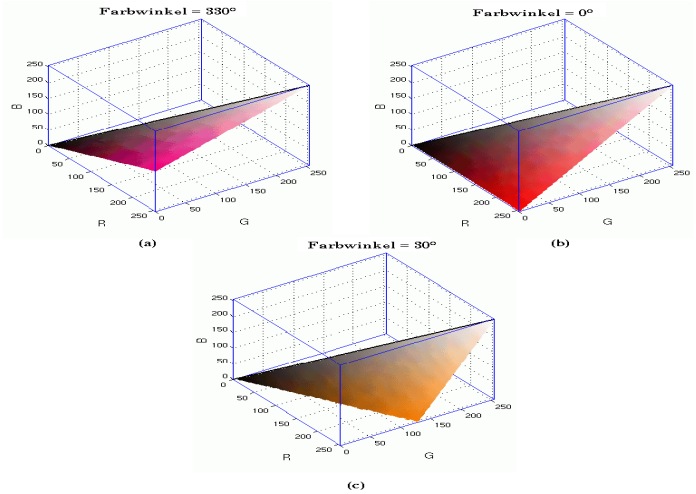

Abbildung 4.8: Flächen mit gleicher Farbe

Abbildung 4.8 stellt drei RGB-Würfel mit verschiedenen Flächen mit gleicher Farbe dar. Weil die Farbe eine Funktion des Winkels ist, ist ihr Bereich zwischen 0 und 360 Grad. Die rote Ecke des Würfels ist als Farbwert von 0 definiert, die auch den Wert 360 hat. Diese Farbe wird in Abbildung 4.8(b) gezeigt. Der Würfel in Abbildung 4.8(a) zeigt einen Farbwert von 330, Purpur, und die Abbildung 4.8(c) einen Farbwert von 30, Orange. Obwohl der Farbwert auf jeder Fläche konstant ist, ändern sich die Helligkeit und Sättigung über den ganzen Bereich der möglichen Werte.

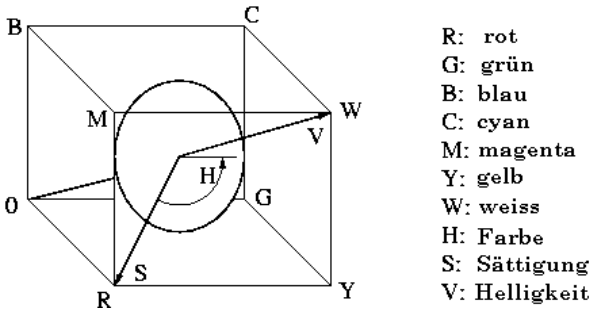

Abbildung 4.9: HSV-Koordinaten im RGB-Würfel

Abbildung 4.9 fasst die Beziehung zwischen RGB und HSV zusammen. Obwohl die neutrale Achse Leuchtkraft und nicht Helligkeit verkörpert, wird diese

Bezeichnung oft missbraucht und auf die Helligkeitsachse bezogen.

Mit dem HSV-Koordinatensystem im Hinterkopf können mehrere Beobachtungen über Farbbereiche im RGB-Würfel gemacht werden. Die erste ist, dass Cyan, Magenta und Gelb hellere Farben als Rot, Grün und Blau verkörpern, weil die letzteren weiter unten auf die neutrale Achse projezieren. Auf ähnliche Weise entsprechen die in der Pyramide von Cyan, Gelb, Magenta und Weiß definierten Farben helleren Farben und die Pyramide, die durch den Ursprung, Rot, Grün und Blau bestimmt ist, entspricht dunkleren Farben. Farben in der Nähe der neutralen Achse des Würfels sehen pastellartig oder ausgewaschen aus, weil sie weniger gesättigt sind und weiter von der Achse entferntere Farben sehen lebhafter aus.

Für die digitale Bildverarbeitung eignet sich der HSV-Farbraum, wie in den vorherigen Ausführungen beschrieben, eindeutig am Besten, weshalb er auch in dieser Arbeit zum Einsatz kommt.

4.2.3 Bilder erkennen

Die zweite Hypothese dieser Arbeit beruht darauf, Objekte anhand ihrer Umrisse und Kanten zu erkennen. Dabei soll das Verhältnis zwischen Höhe und Breite des Objekts genügend Auskunft darüber geben, um welchen Gegenstand es sich handelt, der vor der Kamera steht. Um diese Informationen aber aus einem Bild herauszubekommen, braucht es einige Schritte. So muß das Objekt sich vom Hintergrund unterscheiden lassen und wenn dieses möglich ist, sollten sich die Umrisse des Objektes, besser noch die Abmasse und die Form finden lassen. Wie dieses funktioniert, wird in den nächsten Abschnitten näher beschrieben.

4.2.3.1 Schwarz-Weiß-Filter

Der wichtigste Schritt bei der Objekterkennung ist, das Objekt von seinem Hintergrund zu trennen. Dieses kann man sehr einfach über einen Schwarz-Weiß-Filter realisieren.

Hierzu wird jedes Pixel eines Bildes einzeln betrachtet. Liegt die Farbe des Pixels in einem gewissen Toleranzbereich der zu suchenden Farbe, wird es weiß, ansonsten schwarz. Das heißt, alle Pixel, die der zu suchenden Farbe in etwa entsprechen, werden markiert, der Rest wird ausgeblendet. Dieses kann man in der Abbildung 4.10 sehr gut sehen. Hier wird zum Beispiel die Farbe des Lochers markiert, der Rest wird ausgeblendet. Der Locher hebt sich also von seinem Hintergrund ab.

Abbildung 4.10: Schwarz-Weiß-Filter

Sehr oft kommt es dazu, dass sich nach einem Schwarz-Weiß-Filter störende weiße Pixel im Ergebnis befinden. Dieses passiert dann, wenn die Toleranz zum Beispiel zu großzügig gewählt wurde, so dass Farben, die ähnliche RGB Werte haben, mit in das Zielbild rein geraten. Dieses Rauschen in den Bildern sollte so klein wie möglich gehalten werden.

Um dieses zu erreichen, gibt es diverse Filter, die je nach Bildvorlage mehr oder weniger geeignet sind. Eine gute Möglichkeit, sich ein Bild von den verschiedenen Filtern zu machen, bietet die Software AdOculos [4]. Mit diesem Programm lassen sich alle Arten von Filtern ausprobieren, so dass man ein gutes Gefühl dafür bekommt, welcher Filter für den entsprechenden Einsatzzweck am günstigsten erscheint.

4.2.3.2 Median-Filter

Nach diversen Versuchen mit dem im vorherigen Abschnitt erwähnten Tool AdOculos, habe ich mich für den Median Filter entschieden. Dieser ist in der Lage,

kleine „Fehler“ im Bild zu entfernen, die zum Beispiel bei der Binarisierung oder aber auch schon bei der Aufnahme des Bildes entstanden sind. Außerdem lässt er sich mit einem sehr einfachen Algorithmus realisieren. Dabei lässt man eine Matrix über das gesamte Bild laufen, in diesem Fall ein 5x5 Matrix. Der Median ist dann jeweils der 13. Wert der sortierten Matrix. Anschließend wird das mittlere Pixel in der 5x5 Matrix auf diesen Medianwert gesetzt. Einzelne Pixel und ausgefranste Ränder an Objektkanten lassen sich damit sehr gut entfernen. Ein kleiner Nachteil dieses Verfahrens ist die Rechenintensität, da in dem Algorithmus die Matrix sortiert werden muß.

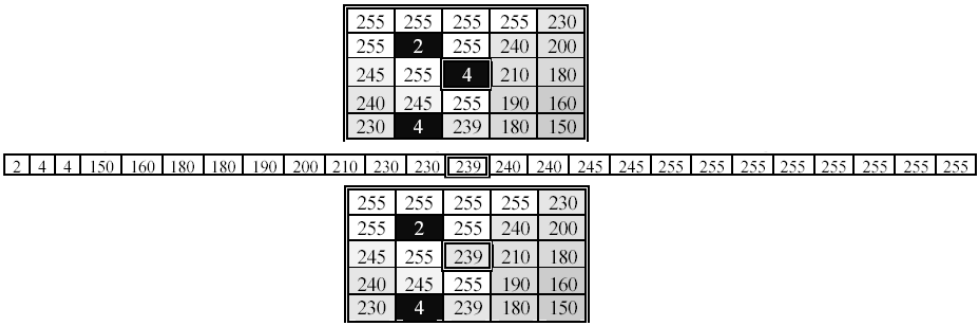

Abbildung 4.11: Anwendung des Median-Filter

4.2.3.3 Kanten finden

Um die Kanten der beliebig lernbaren Objekte zu finden, wird für diese Arbeit die im nächsten Abschnitt beschriebene Hough-Transformation verwendet. Voraussetzung hierfür ist, dass zuvor die Umrisse der eventuell gefundenen Objekte markiert werden. Das heißt im Klartext, dass bei einem Kreis der Ring, der den Umfang beschreibt, hervorgehoben wird und der Rest des Kreises die Hintergrundfarbe annimmt. In Abbildung 4.12 kann man dieses sehr gut sehen.

Hierbei wird wieder, wie beim Median-Filter, jedes Pixel mit denen in der Nachbarschaft liegenden verglichen. Gibt es hier jetzt Unterschiede zwischen den Pixeln, wird der aktuelle Bildpunkt entweder auf schwarz oder auf weiß gesetzt, je nachdem, was der Algorithmus berechnet. Details zum Algorithmus der Kan-

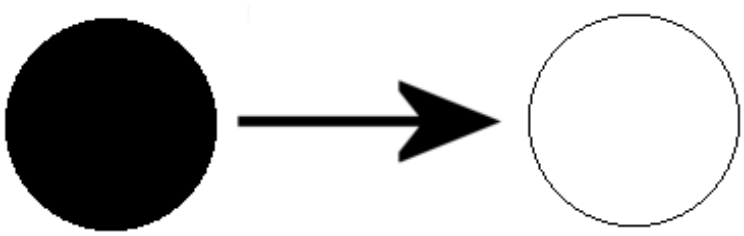

Abbildung 4.12: Kantenfilter

tenfindung kann man ebenso wie den Median-Filter in diverser Literatur finden, auf die im Literaturverzeichnis dieser Diplomarbeit hingewiesen wird [4].

4.2.3.4 Houghtransformation

Die Hough-Transformation stellt ein Verfahren zur Detektion kollinearer Punkte dar, welches 1962 von P.V.C. Hough entworfen und später von Duda und Hart weiterentwickelt wurde. Seit ihrer Erfindung wurde diese Methode intensiv untersucht, was zu mehreren Verallgemeinerungen und einer Vielzahl von Anwendungen in der Computervision und Bildverarbeitung führte. Sie stellt vielseitige Methoden zum Erkennen von Objekten bereit, die in geschlossener, parametrisierbarer Form dargestellt werden können, wie zum Beispiel Linien und Kreise.

Die Idee der Hough-Transformation besteht jetzt darin, alle zu einer Struktur gehörenden Punkte im binären Bildraum auf genau einen Punkt im Transformationsraum (Akkumulator) abzubilden. Dabei geben die Punkte im Transformationsraum dann Rückschlüsse auf das Vorhandensein von Linien im Bildraum.

Um die Punkte des binären Bildraumes in den Transformationsraum zu bekommen, benutzen wir die Hessesche Normalform. Diese beschreibt eine Gerade anders als man es normalerweise gewohnt ist. Dabei wird die Gerade durch den senkrechten Abstand zum Ursprung r_0 und den Winkel α_0 dargestellt. Daraus ergibt sich dann folgende Formel:

$r_0 = xcos(\alpha_0) + ysin(\alpha_0)$ mit r_0, α_0 konstant

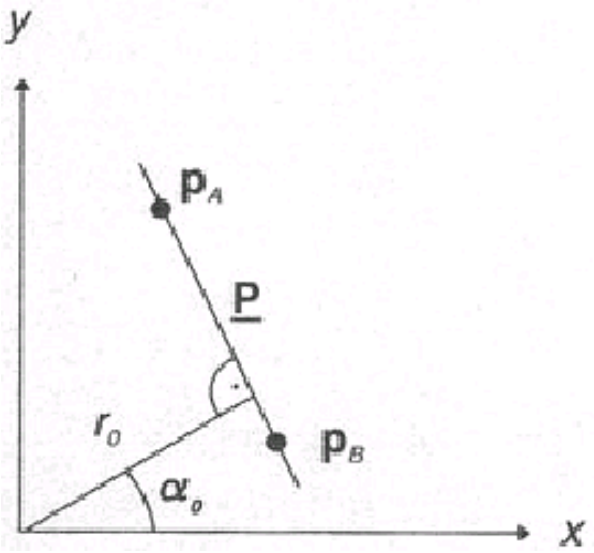

Abbildung 4.13: Hessesche Normalform

Nach erfolgter Transformation jedes Bildpunktes des binären Bildraumes in den Akkumulator erfolgt die Analyse. Dabei deutet ein Maxima im Akku auf eine Gerade im Bildbereich hin. Diese braucht man jetzt nur zurück in den Bildbereich zu transformieren und schon hat man die gesuchte Linien.

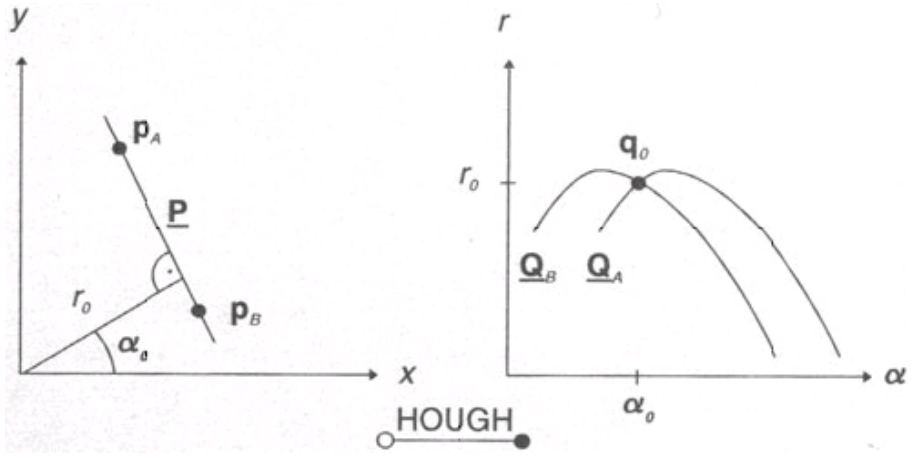

Abbildung 4.14: Bildraum Akkumulator

Die Hough-Transformation lässt sich aber nicht nur für gerade Linien einsetzen, sondern für alle beliebigen Formen, solange sich diese mathematisch beschreiben lassen. Darauf werde ich hier aber nicht näher eingehen, da sich in die-

ser Arbeit die Erkennung auf senkrechte und waagerechte Linien beschränkt und die Erklärung den mathematischen Rahmen dieser Arbeit sprengen würde. Weiterführendes Material gibt es im Internet und in der entsprechenden Literatur, auf die ich hier hinweisen möchte [4].

Da sich der Akkumulator mit bis zu 200 Linien füllen kann, muß bei der Auswertung eine Selektion der relevanten Linien vorgenommen werden. In diesem Fall werden bei den senkrechten und waagerechten Linien nur die äußersten benötigt, dass heißt, die größten Abmessungen eines Objektes werden extrahiert; der Rest wird verworfen. Die Abstände dieser Linien ergeben jetzt die Höhe und die Breite, mit deren Hilfe sich das Objekt um das Merkmal „Form“ beschreiben läßt.

Zusammenfassend gibt es für ein Objekt jetzt vier Merkmale, die in der Modelldatenbank, auf die in einem späteren Kapitel näher eingegangen wird, abgelegt werden können: Farbwert, Helligkeit, Sättigung und Form.

Kapitel 5

Design und Realisierung

Bei dieser Diplomarbeit wird das Hauptaugenmerk auf die Entwicklung der Software gelegt. Im wesentlichen besteht die hier entwickelte Software aus einem Dialog, über den der Benutzer diverse Einstellungen vornehmen kann. Hinter dem Dialog verbergen sich diverse Funktionen, welche die eigentlichen Bildverarbeitungsprozesse darstellen. Der Zusammenhang zwischen der eingesetzten Hardware und der entwickelten Software wird in den folgenden Abschnitten eingehend erläutert.

5.1 Das Design

Einen Überblick über die in dieser Arbeit entwickelte Software sowie den Zusammenhang zwischen den neuen Komponenten und den bereits bestehenden kann man sich mit Hilfe der folgenden Abbildung 5.1 verschaffen.

Dabei wird deutlich, dass der Umfang der Software sich in drei Blöcke aufteilt:

1. PioVision GUI
2. PioVision (Main)
3. PioVisionInterface

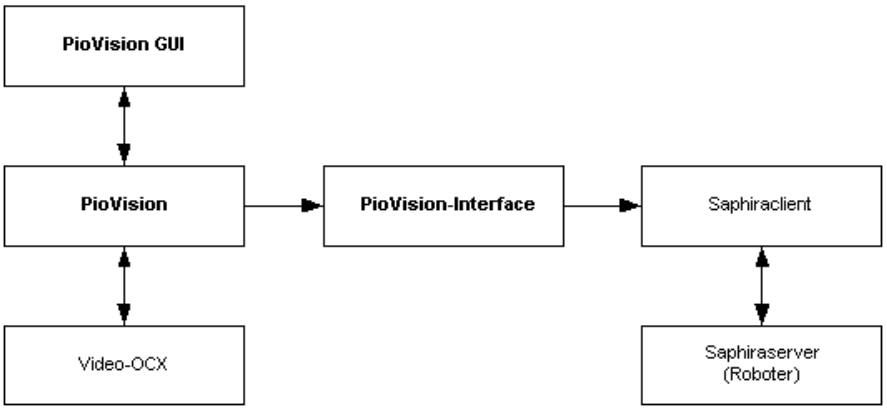

Abbildung 5.1: Überblick über die Komponenten

In den folgenden Abschnitten werden die einzelnen Blöcke der Software genauer erklärt. Da der Anwender zuerst auf die grafische Oberfläche treffen wird, werde ich diese zuerst erläutern.

5.2 PioVision GUI

Den eigentlichen Schwerpunkt dieser Arbeit stellt das Programm „PioVision“ dar. Dieses Tool übernimmt die Bildauswertung und stellt eine Schnittstelle zur Steuerungssoftware Saphira zur Verfügung. Über die grafische Benutzerschnittstelle[1], die in Abbildung 5.2 zu sehen ist, hat der Benutzer die Möglichkeit, diverse Einstellungen an der Bildauswertungssoftware vorzunehmen. Die Oberfläche ist dabei mit Hilfe der grafischen Entwicklungsumgebung Microsoft Visual C++ entworfen worden.

Das große Feld auf der linken Seite dient dazu, die aufgenommen Bilder bzw. die verarbeiteten Bilder darzustellen. Ebenso lassen sich hiermit die Objekte beim „Trainieren“ kontrollieren.

Die drei Schieberegler (H, S und V) dienen dazu, die Schwellwerte für den

[1]GUI (Graphical User Interface)

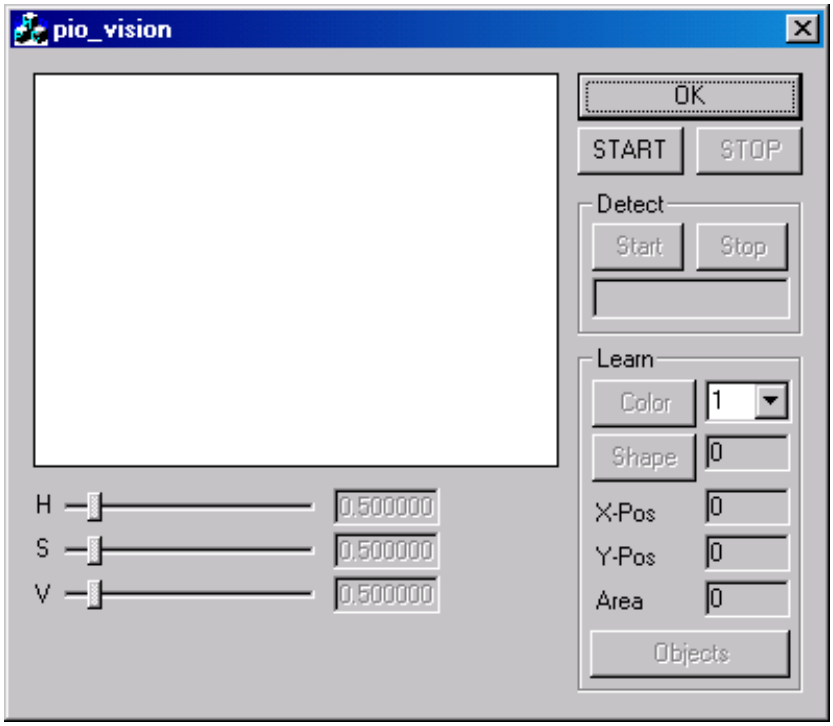

Abbildung 5.2: GUI von PioVision

Farbwert, die Sättigung und die Helligkeit aus dem HSV-Farbraum für jedes einzelne Objekt, welches erlernt wird, anzupassen.

In dem Feld „Learn“ werden die verschiedenen Objekte (zur Zeit drei) gelernt. Dabei wird zuerst die Farbe eines Gegenstandes aufgenommen, von dem anschließend die Konturen mit in die Merkmalsliste aufgenommen werden. Zur besseren Kontrolle werden der Schwerpunkt und die Fläche des gerade zu erlernende Objektes mit angezeigt.

Sobald das erste trainierte Objekt angelegt ist, kann man die Software in den „Detect-Modus“ versetzen. Nachdem dieser aktiviert wurde, wird das aufgenommene Bild auf gelernte Gegestände hin untersucht. Zuerst werden die möglichen Farben verglichen. Gibt es eine Übereinstimmung, werden die Konturen berechnet und mit denen in der Merkmalsliste auf Übereinstimmung geprüft. Kommt es zu einer Übereinstimmung, wird Saphira über eine spezielle Schnittstelle darüber in Kenntniss gesetzt. Die genauere Funktionsweise der Erkennung wird in einem

späteren Abschnitt noch genauer erklärt.

5.3 PioVision

Das eigentliche Hauptprogramm hat im Großen und Ganzen nur zwei wichtige Aufgaben: Zum einen das Erlernen von Objekten und zum anderen dieses „Wissen“ dafür zu benutzen, erlernte Gegenstände in einem beliebigen Umfeld wiederzuerkennen. Um dieses zu realisieren, bedarf es mehrerer Schritte, die in dem folgenden Sequenzdiagramm in Abbildung 5.3 ersichtlich werden.

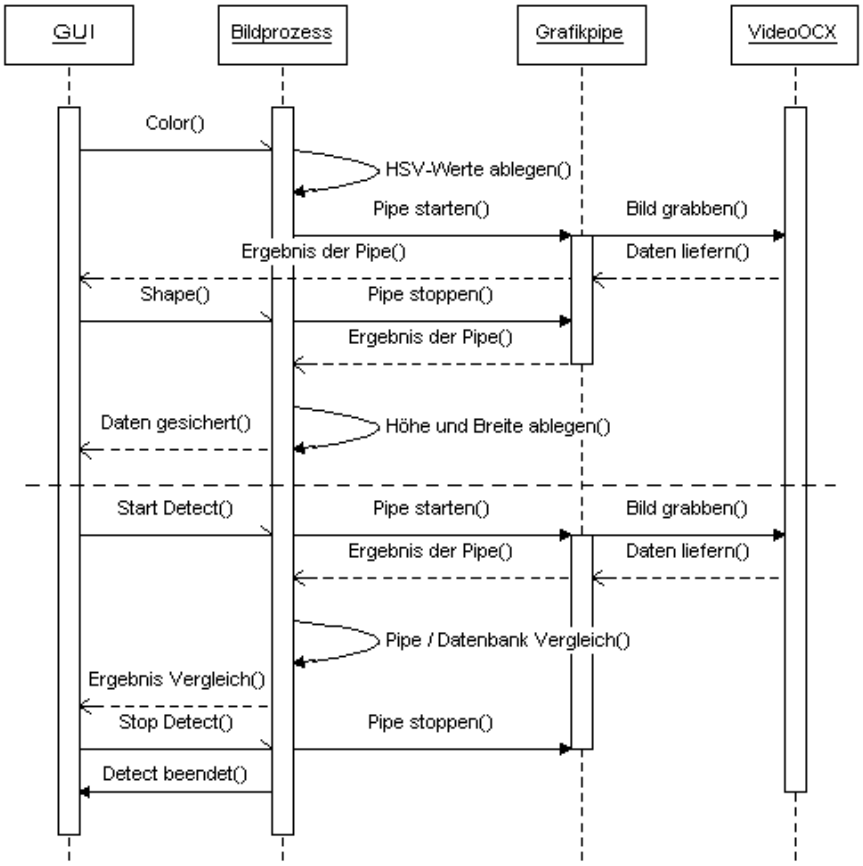

Abbildung 5.3: Erlernen und Erkennen von Objekten

5.3.1 Bilder digitalisieren

Hierbei kann man sehen, dass einer der wichtigsten Schritte die Digitalisierung der Bilddaten ist. Hierfür wird die Framegrabberkarte mit Hilfe der VideoOCX [12] ausgelesen. Der Weg des Bildes von der Kamera zum PC wird in der Abbildung 5.4 deutlich.

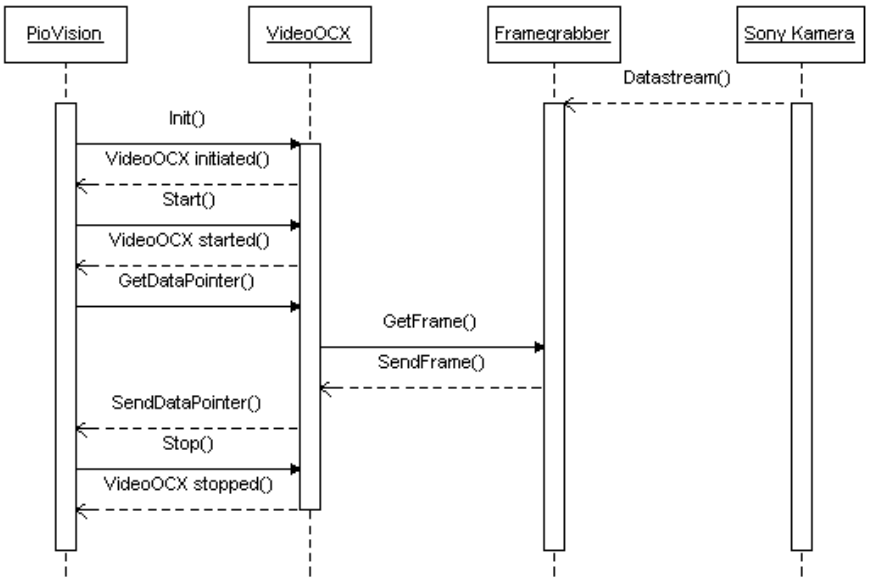

Abbildung 5.4: Vom Bild zum Bit

Die VideoOCX ist dabei eine Schnittstelle, die es ermöglicht, eine beliebige „Video for Windows“-Quelle auszulesen und diese Bilddaten dem hier verwendeten Microsoft Visual Studio zur Verfügung zu stellen. Dabei stellt diese eine Vielzahl von bildverarbeitenden Funktionen zur Verfügung, die hier aber nicht zum Tragen kommen. In dieser Arbeit kommt somit nur ein kleiner Teil des gesamten Funktiosumfanges zur Verwendung, wie in der folgenden Tabelle ersichtlich ist.

Genauere Erklärungen zu den einzelnen hier verwendeten Befehlen gibt es in den Hilfedateien zu VideoOCX, die dem gesamten SDK-Paket beiliegen.

Befehl	Aufgabe
Init()	Initialisierung der VideoOCX
Start()	Starten des Captureprozesses
Stop()	Beenden des Captureprozesses
GetDataPointer()	Datenpointer auf aktuellen Frame
GetHeight()	Höhe der Videoauflösung
GetWidth()	Breite der Videoauflösung

Tabelle 5.1: Verwendete VideoOCX Befehle

5.3.2 Grafikpipe

Die digitalisierten Frames[2] werden, sobald sie im Speicher des Arbeitsrechners angelangt sind, an eine sogenannte Grafikpipe übergeben. Diese Grafikpipe ist dabei in Anlehnung an das Architekturmodell „pipe-and-filter“ entwickelt worden, welches in der Abbildung 5.5 zu sehen ist.

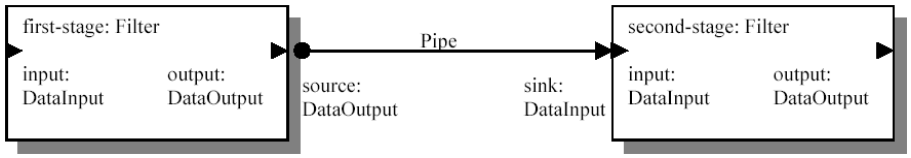

Abbildung 5.5: Architektonische Darstellung eines „pipe-and-filter“ - Systems

Diese abstrakte Darstellung [5] allein lässt schon einige Aussagen über ein „pipe-and-filter“ System zu. Es ist sowohl die Richtung des Datenflusses zu erkennen (von input nach output), als auch wohldefinierte Schnittstellen (source und sink), die die Daten passieren müssen, um von einem Filter zu einem anderen zu gelangen. Dabei muß die „source“ der „pipe“ mit einem „output“ eines Filters und die „sink“ mit einem „input“ eines Filters verbunden sein.

Die Arbeitsweise dieser Pipe kann man sich als eine Verkettung von mehreren einzelnen Filtern vorstellen, die nacheinander die Bilddaten zugereicht bekommen und diese dann mit ihren speziellen Funktionen bearbeiten und analysieren. Einen Überblick über die Grafikpipe dieser Arbeit kann man sich mit Hilfe der Abbildung 5.6 verschaffen.

[2]Frame (Ein einzelnes Bild einer gesamten Bilderfolge)

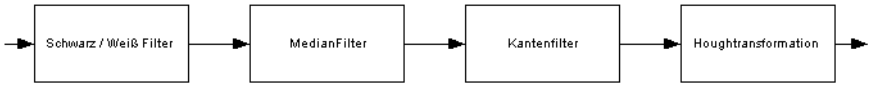

Abbildung 5.6: Grafikpipe

Die Funktionsweisen der einzelnen Filter der Grafikpipe sind im vorherigen Kapitel ausführlich beschrieben worden, auf welches ich hiermit verweisen möchte.

5.3.3 Modelldatenbank

Die Grafikverarbeitung stellte in dieser Arbeit nicht die größte Schwierigkeit dar, da es schon eine Vielzahl von Literatur über dieses Thema gibt, an der man sich orientieren kann. Viel schwieriger ist es, das Wissen über ein Objekt in einer geeigneten Form abzulegen. Dabei sollen diese Daten für spätere Zusatzfunktionen ohne große Probleme erweitert werden können, um eventuell ausgefallenere Objekte aufnehmen zu können. Ich habe mich hier für eine Art Modelldatenbank entschieden, die über ein einfaches Datenobjekt realisiert wird. Dieses Datenobjekt beinhaltet alle benötigten Werte eines Objektes, die für die Wiedererkennung von nöten sind. Welche Werte aber werden zwingend benötigt, um eine Unterscheidung vornehmen zu können? Um eine Zuordnung der einzelnen Objekte machen zu können, wird bei jedem Lernvorgang eine Objekt-ID vergeben und mit in die Datenbank übernommen.

In einem vorherigen Kapitel fiel schon der Begriff „zwei Hypothesen“. Hiermit sind zum einen die Erkennung über die Farbe und zum anderen über die Form eines Objektes gemeint. Wie die Farbe eines Objektes beschrieben werden kann, findet man ausführlich in Kapitel 4 dieser Arbeit. Es müssen folglich die drei Werte des HSV Farbraumes abgelegt werden können. Zusätzlich hierzu werden noch die eingestellten Toleranzwerte der GUI übernommen, die je nach Lichtverhältnissen unterschiedlich sein können.

Als zweite Hypothese kommt hier die Form eines Objektes zum Tragen. Da ich hier noch von idealen Körpern ausgehe, in diesem Fall Zylindern, werden

nur zwei weitere Werte des Objektes benötigt. Da die Kamera sich in einer Höhe befindet, die es nicht zuläßt, diese Röhren von zu weit oben zu betrachten, werden nur noch die Höhe und die Breite des Objektes verwertet, da die runde Öffnung an der Oberseite aus diesem Blickwinkel wie eine Gerade erscheint.

Die Objekte lassen sich jetzt aber anhand dieser Daten nicht ohne weiteres erkennen. Die Röhren sehen nämlich nicht aus jeder Richtung gleich aus. So verändert der Zylinder zum Beipiel ständig seine Farbe, abhängig davon, ob die Sonne scheint oder die Raumbeleuchtung an ist. Das menschliche Auge bekommt davon nicht so viel mit, aber die Kamera reagiert hier auf jeden kleinen Unterschied. Diesem Schwanken der Objektmerkmale wird mit einer Toleranz entgegengewirkt, die beim Erlernen der Farben mit übernommen wurde.

5.3.4 Auswerten der Modelldatenbank

Die Erkennung der Objekte erfolgt im Grunde genommen über simple Vergleichsmethoden zwischen den aktuellen Werten der Grafikpipe und Werten aus der Datenbank. Dabei wird dieses Vergleichen in zwei Stufen durchgeführt. Das Bild wird zuerst auf Farben hin untersucht, da dieses das markantere Merkmal und somit leichter zu finden ist. Sobald eine Farbe aus der Datenbank mit den aktuellen Farben übereinstimmt, wird die Grafikpipe aktiviert. Da wir hier mit recht großen Objekten arbeiten, ist die Schwelle für den Farbenvergleich recht hoch angesetzt, da nicht jeder kleine Farbpixel, der die richtige Farbe hat, erkannt werden soll.

Die von der Grafikpipe gelieferten Höhen- und Breitenangaben werden jetzt nicht direkt mit den Werten der Datenbank verglichen. Da man davon ausgehen muß, dass der Roboter sich bewegt und sich somit mal näher und mal weiter weg von dem Objekt befindet, werden die Höhen- und Breitenangaben in Relation zu einander gesetzt und dann verglichen. So kann ein und dasselbe Objekt auf verschiedenen Distanzen mit gleicher Trefferwahrscheinlichkeit erkannt werden. Anhand dieser Werte läßt sich jetzt der Objektmittelpunkt berechnen, der für die weiterführende Verarbeitung durch Saphira vonnöten ist.

5.4 PioVision-Interface

Die Ergebnisse der Objekterkennung sind zu diesem Zeitpunkt nur dem Tool PioVision bekannt. Da aber der Pioneer-Roboter über diese in Kenntnis gesetzt werden muß, damit dieser seine nächste Aktion ausführt, wird eine Schnittstelle zwischen PioVision und der Steuersoftware Saphira benötigt.

Die Saphirasoftware ist von Hause aus in der Lage, „fremden“ Code in der eigenen Umgebung auszuführen. Hierfür kann man DLLs erstellen, die sich dann zu jeder beliebigen Zeit in der Software laden und ausführen lassen. In der Abbildung 5.7 kann man sehr gut das Zusammenspiel der einzelnen Komponenten sehen.

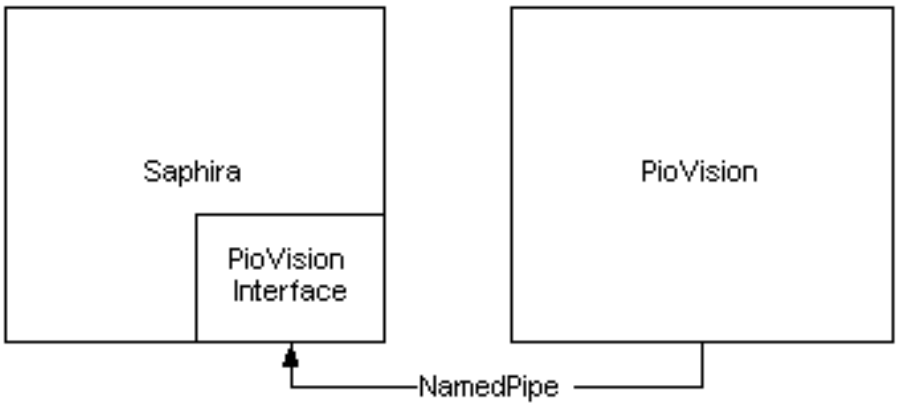

Abbildung 5.7: PioVision Interface

Die eigentliche Kommunikation zwischen der DLL und PioVision erfolgt über eine sogenannte NamedPipe. Diese bietet die Möglichkeit, zwischen zwei Prozessen beliebige Strings auszutauschen. In diesem Fall beschränkt sich die Kommunikation aber darauf, dass PioVision nur Daten an Saphira senden kann. Die NamedPipe ist aber so ausgelegt, dass auch ein Rückkanal möglich wäre. Dieser wird aber für die Aufgabenstellung nicht benötigt.

Die Saphiraseite wartet dabei in einem extra Thread ständig auf die eintreffenden Nachrichten von PioVision. Dadurch wird der Roboter nicht blockiert und andere nebenläufige Prozesse, wie zum Beispiel für die Sonars, können weiterhin

abgearbeitet werden. Sobald PioVision ein Objekt erkennt, wird über die Named-Pipe ein String übermittelt, der die Informationen über den Gegenstand enthält. Dazu gehören die Objekt-ID, sowie die Koordinaten innerhalb des Kamerabildes.

Die Übergabe der Daten erfolgt zum jetzigen Stand ohne jegliche Absicherung, das heißt, es wird nicht darauf geachtet, ob Saphira die Daten schon ausgelesen hat oder nicht. Aus meiner Sicht, zumindest zur Zeit der Entwicklung von PioVision, war dieses auch nicht vonnöten, da die Geschwindigkeit des Roboters nicht so schnell war, dass sich die Objekte vor der Kamera zu schnell ändern würden.

Kapitel 6

Resümee

6.1 Ergebnisse

Zum Abschluß der Programmierung habe ich die Software noch ausgiebig getestet, um festzustellen, ob das ganze Paket zuverlässig Objekte unterscheiden kann. Dafür wurden drei unterschiedliche Objekte trainiert. Zusätzlich zu diesen drei Objekten wurden zwei weitere Objekte in den Erfassungsbereich der Kamera positioniert. Der Roboter hatte jetzt die Aufgabe, nur die erlernten Objekte aus den fünf angebotenen herauszufinden. Nach fünf Durchgängen wurden in drei Versuchen jeweils alle Objekte richtig erkannt, in zwei Versuchen wurden von den dreien nur zwei Zylinder gefunden. Es gab aber keine falschen Treffer. Dieses ergibt immerhin eine Trefferquote von mehr als 86 Prozent.

Im direkten Vergleich der Sony-Camera mit PioVision schneidet die Software wesentlich besser ab, da das Verfahren der Kamera lediglich die größten, zusammenhängenden Farbflächen auswertet. Durch die Formerkennung der Objekte ist man jetzt in der Lage, diese verschiedenen Objekte auch nach ihrer Größe zu unterscheiden. Bei mehreren, gleichfarbigen Objekten mit unterschiedlicher Größe geht die Kamera immer vom größten zum kleinsten. PioVision kann diese Objekte eindeutig unterscheiden, da alles, was erlernt wurde, eine Objekt-ID besitzt.

Die Fotos im Anhang A zeigen den Roboter während der Lernphase und bei der Detektierung eines Objektes.

6.2 Aussichten

Die in dieser Arbeit entwickelte Formerkennung ist zur Zeit noch auf die Erkennung rechteckiger Objekte begrenzt. Es wäre aber ohne großen Aufwand möglich, durch Anpassung der entsprechenden Filter weitere Formen in die Modelldatenbank aufzunehmen. Desweiteren ist noch keine Unterscheidung von nahen und weiter weg befindlichen Objekten vorgesehen, was aber durch geschickte Auswahl zusätzlicher Merkmale ergänzt werden kann.

Zur Zeit muß der Benutzer noch mit Hilfe der GUI entscheiden, ob ein Objekt für den Lernvorgang geeignet ist oder nicht. Da es für das Zusammenspiel mit Saphira wünschenswert wäre, diese GUI auszuschalten, müßte eine Methode gefunden werden, daß die Software diese Entscheidung selber treffen kann.

Anhang A

Bilder

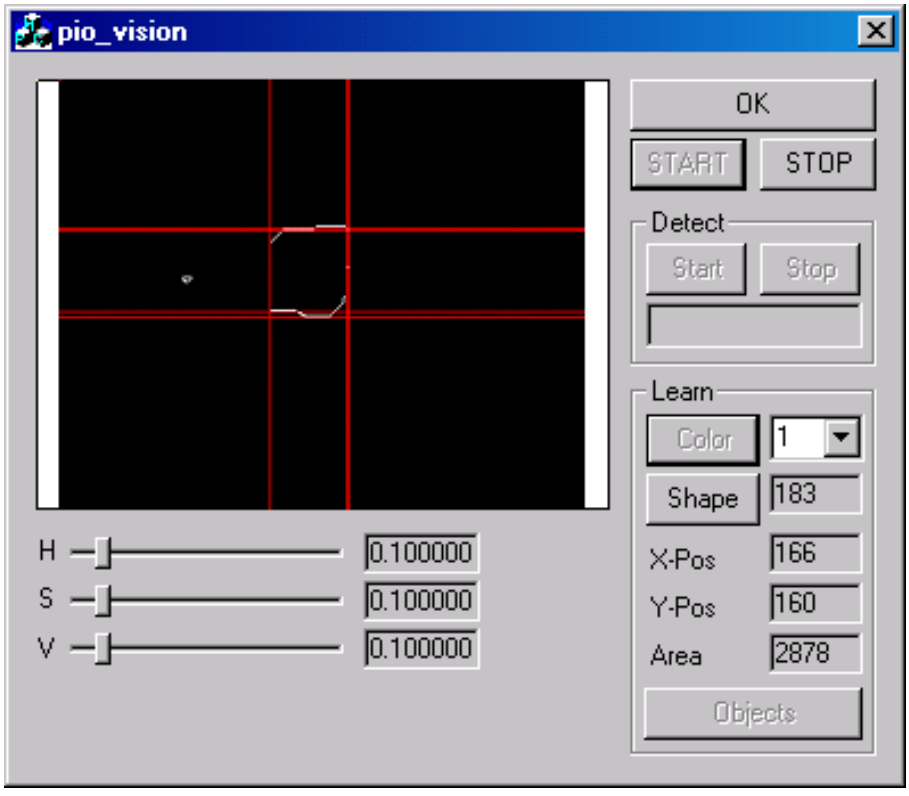

Abbildung A.1: PioVision im Lernmodus

Abbildung A.2: PioVision im Detectmodus

Abbildung A.3: PioVision im Detectmodus

Anhang B

Source Code

Es werden hier nur die wichtigsten Funktionen dieser Diplomarbeit aufgeführt, die für die Bildverarbeitung benötigt wurden. Den gesamten Sourcecode findet man auf der beigelegten CD-Rom.

```
void CPio_visionDlg::BlackAndWhite
(BYTE *data, int size)
{
  int pixel;

  for(pixel=0; pixel<size; pixel=pixel+3)
  {
    rgb_to_hsv(data[pixel+2],data[pixel+1],data[pixel]);

    if(m_hh > m_learn_hsv.hue - (float)(m_slider_hue/10.0) &&
       m_hh < m_learn_hsv.hue + (float)(m_slider_hue/10.0) &&
       m_ss > m_learn_hsv.saturation - (float)(m_slider_saturation/10.0) &&
       m_ss < m_learn_hsv.saturation + (float)(m_slider_saturation/10.0) &&
       m_vv > m_learn_hsv.value - (float)(m_slider_value/10.0) &&
       m_vv < m_learn_hsv.value + (float)(m_slider_value/10.0))
    {
      data[pixel]   = 255;
      data[pixel+1] = 255;
      data[pixel+2] = 255;
    }
    else
    {
      data[pixel]   = 0;
      data[pixel+1] = 0;
      data[pixel+2] = 0;
    }
  }
}
```

```
void CPio_visionDlg::Median
(BYTE *data, int height, int width)
{
  int xpos, ypos, median, x_window, y_window;

  for(ypos=1; ypos<height-1; ypos++)
  {
    for(xpos=1; xpos<width-1; xpos++)
    {
      median = 0;

      for(y_window=-1; y_window<=1; y_window++)
      {
        for(x_window=-1; x_window<=1; x_window++)
        {
          if(data[((ypos+y_window)*3*width)+((xpos+x_window)*3)]==255)
          median++;
        }
      }
      if(median>=4)
      {
        data[(ypos*3*width)+(xpos*3)]   = 255;
        data[(ypos*3*width)+(xpos*3)+1] = 255;
        data[(ypos*3*width)+(xpos*3)+2] = 255;
      }
      else if(median<4)
      {
        data[(ypos*3*width)+(xpos*3)]   = 0;
        data[(ypos*3*width)+(xpos*3)+1] = 0;
        data[(ypos*3*width)+(xpos*3)+2] = 0;
      }
    }
  }
}
```

```
void CPio_visionDlg::EdgeDetection
(BYTE *data_in, BYTE *data_out, int height, int width)
{
  int xpos, ypos;
  int pixel = 0;

  for(ypos = 0; ypos < height; ypos++)
  {
    for(xpos = 0; xpos < width; xpos++)
    {
      if(xpos > 0 && ypos < height-1)
      {
        if(data_in[pixel]-data_in[pixel-3] != 0)
        {
          data_out[pixel] = 255;
          data_out[pixel+1] = 255;
          data_out[pixel+2] = 255;
        }
        else if(data_in[pixel]-data_in[pixel+width*3] != 0)
        {
          data_out[pixel] = 255;
          data_out[pixel+1] = 255;
          data_out[pixel+2] = 255;
        }
        else
        {
          data_out[pixel] = 0;
          data_out[pixel+1] = 0;
          data_out[pixel+2] = 0;
        }
      }
      else
      {
        data_out[pixel]=0;
        data_out[pixel+1]=0;
        data_out[pixel+2]=0;
      }
      pixel=pixel+3;
    }
  }
}
```

```
void CPio_visionDlg::Hough
(BYTE *data, int height, int width)
{
  int i,j;
  int r,phi,phi_t;

  for(i=0; i<360; i++)
  {
    for(j=0; j<200; j++)
    {
      m_hough[i][j] = 0;
    }
  }

  for(i=0; i<height; i++)
  {
    for(j=0; j<width; j++)
    {
      if(data[(i*3*width)+(j*3)] == 255)
      {
        for(phi=0;phi<180;phi++)
        {
          r=(int)(j*cos(phi*3.14/180))+(int)(i*sin(phi*3.14/180));

          if(r>=0)
          {
            m_hough[phi][r]++;
          }
          else if(r<0)
          {
            r=-r;
            phi_t = phi + 180;
            m_hough[phi_t][r]++;
          }
        }
      }
    }
  }
}
```

```
void CPio_visionDlg::HoughAnalyse
(int height, int width)
{
  int i,j;
  int rmax[200];
  int phimax[200];
  int hough_max[200];

  m_lines = 0;

  for(i=0; i<360; i++)
  {
    for(j=0; j<200; j++)
    {
      if(m_hough[i][j] > 10 && m_lines < 200)
      {
        hough_max[m_lines] = m_hough[i][j];
        rmax[m_lines]      = j;
        phimax[m_lines]    = i;
        m_lines++;
      }
    }
  }

  for(i = 0; i < 200; i++)
  {
    m_hough_x1[i] = 0;
    m_hough_y1[i] = 0;
    m_hough_x2[i] = 0;
    m_hough_y2[i] = 0;
  }

  for(i = 0; i < m_lines; i++)
  {
    // vertikale Linien
    if(phimax[i] == 0 || phimax[i] == 180)
    {
      m_hough_x1[i] = rmax[i];
      m_hough_y1[i] = 0;
      m_hough_x2[i] = rmax[i];
      m_hough_y2[i] = height-1;
    }
    // horizontale Linien
```

```
    else if(phimax[i] == 90 || phimax[i] == 270)
    {
      m_hough_x1[i] = 0;
      m_hough_y1[i] = height - rmax[i]-1;
      m_hough_x2[i] = width-1;
      m_hough_y2[i] = height - rmax[i]-1;
    }
  }
}
```

```
void CPio_visionDlg::rgb_to_hsv
(int r, int g, int b)
{
  float max, min;
  float rr,gg,bb;

  rr = (float)(r/255.0);
  gg = (float)(g/255.0);
  bb = (float)(b/255.0);

  if(rr>=gg && rr>=bb) max = rr;
  if(gg>=rr && gg>=bb) max = gg;
  if(bb>=gg && bb>=rr) max = bb;

  if(rr<=gg && rr<=bb) min = rr;
  if(gg<=rr && gg<=bb) min = gg;
  if(bb<=gg && bb<=rr) min = bb;

  m_vv = max;

  if(max != 0.0) m_ss = (max - min)/max;

     if(rr == max && gg == min) m_hh = 5 + (rr-bb)/(rr-gg);
  else if(rr == max && bb == min) m_hh = 1 - (rr-gg)/(rr-bb);
  else if(gg == max && bb == min) m_hh = 1 + (gg-rr)/(gg-bb);
  else if(gg == max && rr == min) m_hh = 3 - (gg-bb)/(gg-rr);
  else if(bb == max && rr == min) m_hh = 3 + (bb-gg)/(bb-rr);
  else if(bb == max && gg == min) m_hh = 5 - (bb-rr)/(bb-gg);

  m_hh = (float)(m_hh * 3.14 / 3);
}
```

```
#include "saphira.h"

#define MAX 256

BOOL run;
char buffer[MAX+1];

void thread(void *threadnumber)
{
HANDLE pio_vision_handle;
char name[MAX+1];
DWORD nr;

sprintf(name,"\\\\.\\pipe\\pio_pipe");

while( run )
{
pio_vision_handle = CreateNamedPipe( name,
 PIPE_ACCESS_INBOUND,
 PIPE_TYPE_BYTE,
 1,
 MAX,
 MAX,
 500,
 NULL);

do
{
strcpy(buffer, "");

do
{
ReadFile(pio_vision_handle, buffer, MAX, &nr, NULL);
Sleep(10);
}
while (strcmp(buffer, "") == 0);

sfMessage(buffer);
}
while (strcmp(buffer, "###") != 0);

Sleep(10);
```

```
CloseHandle(pio_vision_handle);
}

_endthread();
}

EXPORT void sfPioVisionStart()
{
run = TRUE;
_beginthread(thread, 4096, (void *) 1);
}

EXPORT void sfPioVisionStop()
{
run = FALSE;
}

EXPORT void sfPioVisionGetObj()
{

}

/*---------------------------------------------------------------*/
/*---------------------------------------------------------------*/
/*---------------------------------------------------------------*/

EXPORT void sfLoadInit(void)
{
sfAddEvalFn("sfPioVisionStart",  sfPioVisionStart,  sfVOID, 0);
sfAddEvalFn("sfPioVisionStop",  sfPioVisionStop,  sfVOID, 0);
sfAddEvalFn("sfPioVisionGetObj",  sfPioVisionGetObj,  sfVOID, 0);
}

EXPORT void sfLoadExit(void)
{

}
```

Anhang C

Inhalt der CD-ROM

Auf der zu dieser Arbeit gehörenden CD-ROM befinden sich folgende Dateien:

1. Diese Arbeit als PDF
2. Komplettes Projekt PioVision
3. Komplettes Projekt PioVisionInterface
4. Diverse Fotos zur Diplomarbeit

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] ACTIVMEDIA: *Activmedia*. online. – URL http://www.activmedia.com. – Zugriffsdatum: 10.03.2003

[2] ACTIVMEDIA: *Saphira*. online. – URL http://www.activrobots.com/SOFTWARE/index.html. – Zugriffsdatum: 10.03.2003

[3] BALZEROWSKI, Rainer: *Realisierung eines Webcam basierten Kamera-Systems für mobile Roboter*. online. 2002. – URL http://www.informatik.haw-hamburg.de/~robots/papers.html. – Zugriffsdatum: 11.03.2003

[4] BÄSSMANN, Kreyss: *The Imaging Source AdOculos*. online. – URL http://www.theimagingsource.com/prod/soft/adoculos. – Zugriffsdatum: 27.01.2003

[5] GAMMA, E.: *Design Patterns*. 1995

[6] GMBH, Dazzle E.: *Dazzle*. online. – URL http://www.dazzle-europe.de. – Zugriffsdatum: 10.03.2003

[7] GROTH, Grawe: *Entwicklung und Realisierung eines Garbage-Collection Serviceroboters für partiell bekannte Räume*. online. – URL http://www.informatik.fh-hamburg.de/~pioneer/. – Zugriffsdatum: 16.01.2003

[8] KONOLIGE, Kurt: *Kurt Konolige*. online. – URL http://www.ai.sri.com/~konolige. – Zugriffsdatum: 02.11.2002

[9] LEGO: *Lego Mindstorms*. online. – URL http://mindstorms.lego.com/de/index.asp. – Zugriffsdatum: 10.03.2003

[10] LUCK, Kai von: *IKS-project (Integration Kognitiver Systeme)*. online. – URL http://www.informatik.fh-hamburg.de/~robots/. – Zugriffsdatum: 06.12.2002

[11] LUCK, Kai von: *Krabat-Project*. online. – URL http://www.informatik.fh-hamburg.de/~krabat/. – Zugriffsdatum: 06.12.2002

[12] MARVELSOFT: *Video OCX*. online. – URL www.videoocx.de. – Zugriffsdatum: 10.03.2003

[13] MICROSOFT: *Microsoft*. online. – URL www.microsoft.com. – Zugriffsdatum: 10.03.2003

[14] MICROSOFT: *Microsoft Foundation Classes*. online. – URL www.microsoft.com. – Zugriffsdatum: 10.03.2003

[15] MIT: *MIT 6270*. online. – URL http://web.mit.edu/6.270/www/. – Zugriffsdatum: 10.03.2003

[16] NEWTONLABS: *ARC C*. online. – URL http://www.newtonlabs.com/arc/. – Zugriffsdatum: 10.03.2003

[17] NEWTONLABS: *Cognachrome Vision System*. online. – URL http://www.newtonlabs.com/cognachrome/. – Zugriffsdatum: 10.03.2003

[18] NEWTONLABS: *Newtonlabs*. – URL http://www.newtonlabs.com. – Zugriffsdatum: 10.03.2003

[19] PETERS: *USB-Grundlagen, USB-1.1*. online. – URL http://www.sportmarketing-peters.de/usb.html. – Zugriffsdatum: 27.01.2003

[20] SONY: *Sony*. online. – URL www.sony.com. – Zugriffsdatum: 17.11.2002

[21] WISCHMANN, Arne: *Sonarsensorik für mobile Roboter*. online. 2002. – URL http://www.informatik.haw-hamburg.de/~robots/papers.html. – Zugriffsdatum: 11.03.2003

Stichwortverzeichnis

Versicherung über Selbstständigkeit

Hiermit versichere ich, daß ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfaßt und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum	Unterschrift des Studenten