university of applied sciences
gegr. 1970 fachhochschule hamburg
FACHBEREICH ELEKTROTECHNIK UND INFORMATIK

Elektrotechnik und Informatik

Diplomarbeit

Rainer Balzerowski

Realisierung eines Webcam basierten Kamera Systems für mobile Roboter

+++ neuer name +++
hochschule für angewandte
wissenschaften hamburg

university of applied sciences
gegr. 1970 fachhochschule hamburg
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIK

Diplomarbeit

Rainer Balzerowski

Realisierung eines Webcam basierten Kamera Systems für mobile Roboter

Studiengang Technische Informatik
Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Gunter Klemke
Abgegeben am 16. Januar 2002

BERLINER TOR 3 · D - 20099 HAMBURG

Realisierung eines Webcam basierten Kamera Systems für mobile Roboter

Stichworte

Mobiler Roboter, LEGO MINDSTORMS, Infrarotkommunikation, Bildverarbeitung, Webcam

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit der Möglichkeit günstige Webcams durch digitale Bildverarbeitung als Sensor für mobile autonome Roboter einzusetzen.

Das im Rahmen der Diplomarbeit entstandene Programm ist für das Robot–Lab der Hochschule für Angewandte Wissenschaften Hamburg gedacht, in dem LEGO MINDSTORMS Robotics Invention System eingesetzt werden, mit denen das entstandene Programm über eine Infrarotschnittstelle kommuniziert. In dieser Arbeit werden die angewandten Methoden der Bildverarbeitung beschrieben und Alternativen aufgezeigt.

Realization of a web cam based vision system for mobile robots

Keywords

mobile robot, LEGO MINDSTORMS, infrared communication, digital image processing, web cam

Abstract

This thesis (diploma) concerns with the possibility of using low-priced web cams as sensor for mobile autonomous robots by doing digital image processing.

The program developed in context of this thesis is for the use at the Robot lab of the university of applied sciences Hamburg, in which LEGO MINDSTORMS Robotics Invention Systems are used. The developed program will communicate with these Systems using an infrared interface. The applied methods of image processing are described and alternatives are pointed out in this work.

Danksagung

DANKE, DANKE, DANKE !!!!!

Inhaltsverzeichnis

Kapitel 1

Einleitung

Roboter[1] sind mittlerweile Teil unserer Umwelt. Ihre Anzahl wächst genauso wie die Anzahl ihrer Einsatzgebiete stetig. Nicht nur die stationären Industrie–Roboter, sondern auch mobile Service–Roboter sind in unserer Umgebung zu finden. So gibt es mittlerweile rasenmähende und staubsaugende Roboter sowie Roboter–Hunde als Haustierersatz.

Robotik, *"eine wissenschaftlich-technische Disziplin, die sich mit der Konstruktion, der Programmierung und dem Einsatz von Robotern befaßt"* [DUDEN 1994], ist eine interdisziplinäre Herausforderung. Nicht nur die Hardware ist mit ihrer aufwendigen Mechanik und der empfindlichen Elektronik schwer zu beherrschen, auch dem Informatiker bietet dieses Gebiet ein anspruchsvolles Betätigungsfeld. Alle Gebiete der künstlichen Intelligenz finden auf dem Gebiet der Robotik Anwendung. Neuronale Netze und genetische Algorithmen gehören ebenso zu dem Gebiet der Robotik wie Sensordaten–Fusion und natürlich die digitale Bildverarbeitung, besser das Bildverstehen [Nilsson 1998].

Das Bildverstehen versucht, durch maschinelle Sehsysteme und geeignete Berechnungsmethoden technische Probleme zu lösen [Neumann 2000].

Als Vorbilder des Bildverstehens stehen die biologischen Sehsysteme. Allerdings sind die Vorgänge in biologischen Sehsystemen, also dem Auge und dem Gehirn, von einer so hohen Komplexität, daß sie erst in den Anfängen verstanden worden sind. Für den Menschen sind Dinge wie z.B. Schattenanalyse[2] selbstverständliche Dinge, über die er nicht nachzudenken braucht. Für ein Rechensystem mit Kameras ist dies aber immer noch eine fast unüberwindliche Hürde.

[1] Roboter: eine Wortschöpfung des tschechischen Malers und Schriftstellers Josef Capek. Allerdings war es das Theaterstück "R.U.R. (Rossum's Universal Robots)" seines jüngeren Bruders Karel, ebenfalls Schriftsteller, das dieses Wort seit 1921 weltweit populär machte. Das tschechische Wort "robota" bedeutet in etwa "Zwangsarbeiter". Diese Information verdanke ich Prof. Jan Nouza, Professor an der "Technical University of Liberec", Tschechische Republik. Danke Jan !!

[2] Ein blaues Auto ist für uns Menschen immer noch Blau, wenn es im Halbschatten eines Baumes steht.

1.1 Motivation

Bereits seit 1986 forscht das renommierte Forschungszentrum Massachusetts Institute of Technologies (MIT) an einem LEGO–Stein kompatiblen Mikrocomputer, dem “MIT Programmable Brick” [Martin u. a.]. Dieser Mikrocomputer hat LEGO inspiriert, selbst einen Mikrocomputer zu entwickeln, der auf der Forschungsarbeit des MIT basiert. Dieser LEGO–Stein wird als RCX im LEGO MINDSTORMS Robotics Invention System vertrieben.

Seit Ende 2000 gibt es von LEGO unter dem Namen Vision Command auch eine Kamera, die zusammen mit dem RCX für Robotik Aufgaben benutzt werden kann.

Die Idee zu dieser Arbeit entstand, nachdem ich mit Kommilitonen an einem Wettbewerb im Rahmen einer Werbeveranstaltung teilgenommen habe, bei dem es darum ging, die LEGO Vision Command Kamera zusammen mit mehreren LEGO RCX für Robotik Aufgaben einzusetzen. Die zum Lieferumfang der Kamera gehörende Software ist in der Lage, auf Bewegung sowie auf angelernte Farben zu reagieren. Als Reaktion werden Kommandos an den zu einem Roboter gehörenden RCX übermittelt. Diese Aufgaben erfüllte die LEGO Software allerdings nicht zu unserer Zufriedenheit, da nur noch die Kommandos der Vision Command Software auf dem RCX ausgeführt werden. Eine gesonderte Programmierung des RCX ist nicht mehr möglich, selbst die an den RCX angeschlossenen Sensoren können nicht ausgelesen werden. Diese Unzufriedenheit führte zu der Idee eine eigene Software für diese Kamera und den RCX zu schreiben, die im Rahmen des Robot–Labs der Hochschule für Angewandte Wissenschaften Hamburg eingesetzt werden kann. Die Realisierung dieser Software ist Teil dieser Arbeit.

1.2 Zielsetzung

Ziel dieser Arbeit ist es, geeignete Verfahren der digitalen Bildverarbeitung zu finden und zu implementieren. Die zu entwickelnde Software soll es Benutzern der LEGO Vision Command Kamera und des LEGO RCX ermöglichen, Positionen verschiedener Farbflächen im Sichtbereich der Kamera an den RCX zu übermitteln. Um dieses Ziel erreichen zu können, müssen einige Rahmenbedingungen festgelegt werden.

Die Software ist für eine ebene, weiße Spielfläche gedacht, über der an einem Stativ hängend die Kamera angebracht ist. Die Beleuchtung sollte homogen, von oben und möglichst ohne Schattenwurf erfolgen. Tageslicht sollte nach Möglichkeit vermieden werden. Diese Bedingungen vereinfachen die digitale Bildverarbeitung und ermöglichen ein recht zuverlässiges Bildverstehen im Bezug auf die verschiedenen Farbflächen.

1.3 Beschreibung der Aufgabe

Die zu entwickelnde Software soll unter anderem im Rahmen eines an der Hochschule für Angewandte Wissenschaften Hamburg stattfindenden Robotik–Wahlpflichtkurses eingesetzt werden.

In diesem Kurs wird in jedem Semester versucht, eine neue Aufgabe zu bewältigen. Diese Aufgaben sind durch autonome mobile Roboter zu lösen, die aus LEGO–Steinen und einem RCX sowie diversen Sensoren und Aktoren bestehen dürfen.[3]

Für diese Roboter soll auch die Vision Command Kamera zugänglich gemacht werden.

Die zu entwickelnde Software soll es ermöglichen, ein von der LEGO Vision Command Kamera aufgenommenes Bild so in Informationen zu wandeln, daß sie über die Infrarotschnittstelle an den RCX geschickt werden können.

Hierbei soll die Software etwas Ähnliches leisten, wie die Hardwarelösung Cognachrome Vision System von newton labs [Newton]. Diese spezielle Karte zur Bildverarbeitung ist in der Lage, mehrere Objekte einer angelernten Farbe zu beobachten. Man kann dieser Karte drei unterschiedliche Farben "beibringen", von der jeweils mehrere Objekte beobachtet werden können. Abhängig von der Anzahl der beobachteten Objekte und der zu berechnenden Daten, können bis zu sechzig Bilder pro Sekunde bearbeitet werden.

Am Ausgang der Karte stehen auf Wunsch für jedes Objekt verschiedene Daten zur Verfügung.

Diese Daten sind:

- X– und Y–Koordinate des Schwerpunktes
- Größe
- Winkel der Hauptachse
- "Aspect Ratio"[4]

des jeweiligen Objektes. Diese enorme Leistung hat natürlich ihren Preis. Diese Karte kostet mehrere tausend US Dollar.

Die Softwarelösung, die im Rahmen dieser Diplomarbeit entwickelt wird, sollte in der Lage sein, jeweils ein Objekt einer bestimmten Farbe zu beobachten.

Die zu erkennenden Objekte können von einer der sechs Farben Rot, Grün, Blau, Gelb, Cyan oder Magenta sein.

[3] Neben dem RCX wird auch ein ebenfalls am MIT entwickeltes Robotik–Board eingesetzt, das nach dem Kurs, in dem es am MIT eingesetzt wird, 6.270 benannt wurde.

[4] Mit Aspect Ratio bezeichnet newton research labs die Quadratwurzel aus dem Verhältnis der Hauptachse zur Nebenachse.

Flächen dieser Farben werden bestimmt und ihre Größe sowie ihr Schwerpunkt berechnet. Die Größe wird benötigt, um die größte Fläche einer Farbe zu bestimmen, falls mehrere Flächen einer Farbe vorhanden sind. Da nur ein Koordinatenpaar pro Farbe verwaltet wird, werden die Informationen einer kleineren Fläche der gleichen Farbe verworfen.

Die Koordinatenpaare der einzelnen Farbflächen werden bezüglich der Auflösung der Kamera ermittelt und auf Anforderung, per Infrarotkommunikation an den RCX des Roboters geschickt.

1.4 Aufbau der Arbeit

Kapitel 2 erläutert die Rahmenbedingungen der Diplomarbeit. In Kapitel 3 werden Überlegungen angestellt, welche Schnittstellen und Software–Werkzeuge für die Entwicklung des Programmes eingesetzt werden können. Das Konzept hinter der zu entwickelnden Software wird in Kapitel 4 vorgestellt. In Kapitel 5 wird auf die Implementierung eingegangen. Abschließend wird in Kapitel 6 ein Resümee der Arbeit gezogen.

1.5 Eingetragene Warenzeichen

Kapitel 2

Rahmenbedingungen der Diplomarbeit

Dieses Kapitel beschreibt die Rahmenbedingungen der Diplomarbeit anhand des Robot–Labs der Hochschule für Angewandte Wissenschaften Hamburg. Diese Arbeit basiert auf der in dem Robot–Lab eingesetzten LEGO Ausstattung. Dieser Bereich wird deshalb ausführlich beschrieben.

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 IKS Projekt [von Luck] 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 [Krabat], für Robotikzwecke entwickelt worden, auf dem ein embedded Linux läuft. Ein Teil des Labores beschäftigt sich mit den weit verbreiteten Pioneer Robotern der Firma ActivMedia [Pioneer]. Der Teil, der sich mit LEGO Robotern beschäftigt, ist inzwischen zweigeteilt. Ein Teil setzt das bereits 1989 am MIT entwickelte 6.270 [Martin] Board ein, das nach der Kursnummer am MIT benannt worden ist. Der andere Teil benutzt den LEGO RCX. Dieser ist wesentlicher Bestandteil dieser Arbeit.

2.2 Beschreibung des LEGO Labors

Das Robot–Lab der Hochschule für Angewandte Wissenschaften Hamburg hat für die Ausbildung von Lehrern und Schülern einige LEGO MINDSTORMS Robotics Invention System (RIS) Kästen gekauft. Zusätzlich zu diesen RIS Kästen gibt es die gleiche Anzahl Vision Command Kameras.

Diese Ausstattung steht in jedem Semester den Studenten eines Wahlpflichtkurses an der Hochschule für Angewandte Wissenschaften Hamburg zur Verfügung. Dieser Kurs findet im sechsten und siebten Semester für die Studenten der Studiengänge Softwaretechnik und technische Informatik statt.

Auf der Basis dieser Ausstattung ist ein Szenario erstellt worden, das als Grundlage dieser Arbeit dient. Dieses Szenario sieht folgendermaßen aus: An einem Stativ hängend wird die Vision Command Kamera befestigt. Neben der Kamera hängt der LEGO *Infrarot Tower*. Dies ist der Infrarotsender, der über die serielle Schnittstelle mit dem Computer verbunden ist, auf dem die zu entwickelnde Software läuft. Über diesen Sender findet die Kommunikation zum RCX statt. Unterhalb der Kamera befindet sich die rechteckige Spielfläche. Sie ist Weiß und hat in etwa eine Größe von 2,0 m Breite und 1,5 m Länge. Die Farbe Weiß wurde gewählt, um die auf der Spielfläche auftretenden Farben besonders gut bestimmen zu können. Wie bereits in Abschnitt 1.2 erwähnt, sind die Grundfarben Rot, Grün und Blau sowie die daraus direkt mischbaren Farben Cyan, Magenta und Gelb für die zu erkennenden Farbflächen ausgewählt worden. Auf die additive Farbmischung und ihre Farben wird in Abschnitt 4.4.1 genauer eingegangen.

2.3 LEGO MINDSTORMS Robotics Invention System

Im Jahr 1998 hat die Firma LEGO den ersten MINDSTORMS Robotics Invention System Kasten auf den Markt gebracht. Die Aufschrift auf dem Karton wirbt mit dem Slogan:

> "Power of Robotics @ Your Command"

Im folgenden Jahr bereits erschien ein Nachfolger. Dieser Kasten wurde MINDSTORMS Robotics Invention System 1.5 genannt. Weitere zwei Jahre später war dann zum ersten Mal der MINDSTORMS Robotics Invention System 2.0 Kasten auf dem Markt[1].

[1] Die verschiedenen Robotics Invention System Kästen tragen folgende LEGO Set–Nummern: RIS 1.0 ⇔ 9719, RIS 1.5 ⇔ 9747, RIS 2.0 ⇔ 3804

Abbildung 2.1: LEGO MINDSTORMS 1.0

Der Inhalt der drei verschiedenen Kästen unterscheidet sich nicht wesentlich. Die wirklichen Unterschiede bestehen in der Software, dem RCX und dem Infrarotsender.

Während der RCX der ersten Version einen Anschluß für ein externes Netzteil hatte, müssen die Käufer der 1.5 und 2.0 Version auf diesen Anschluß verzichten. Der RCX läßt sich ausschließlich mit sechs Mignonzellen betreiben.

Im Unterschied zu den Versionen 1.0[2] und 1.5 enthält die Version 2.0 keinen seriellen Infrarotsender, sondern einen für die USB–Schnittstelle. Das hat den großen Vorteil, daß man keine Spannungsversorgung in dem Infrarotsender benötigt. Der serielle Infrarotsender muß im Gegensatz zum USB–Gerät von einem handelsüblichen 9 Volt Block gespeist werden.

Ebenfalls anders als bei den Versionen 1.0 und 1.5 ist bei der Version 2.0 die Software. Dies betrifft sowohl die LEGO Programmierumgebung als auch die auf dem RCX gespeicherte “Firmware”. Diese neue Programmierumgebung und die neue Firmware lassen sich aber auch zusammen mit den RCX der älteren Versionen benutzen.

[2]Eine Version 1.0 gibt es eigentlich gar nicht, im allgemeinen wird die erste Version aber so bezeichnet.

Grundsätzlich allerdings enthalten alle drei MINDSTORMS Robotics Invention System Kästen die folgenden Teile:

- RCX Mikrocomputer
- Software auf CD–ROM
- Konstruktionsanleitung
- mehr als 700 LEGO Teile inklusive
  - 2 Motoren
  - 2 Druck–Sensoren (Taster)
  - 1 Licht–Sensor
- Infrarotsender inklusive Anschlußkabel

2.3.1 RCX und ”seine“ Welt

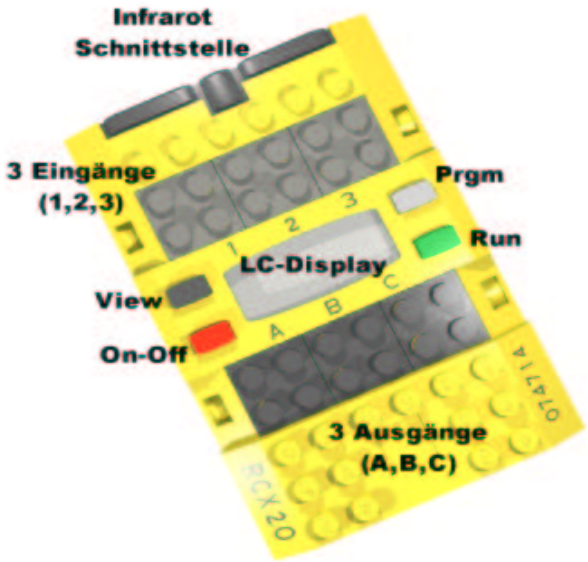

Abbildung 2.2: LEGO MINDSTORMS RCX

Den Kern des RCX macht der Ein–Chip–Mikrocomputer H8 aus dem Hause Hitachi aus.[3] Dieser Mikrocomputer verfügt bereits über fünf 8–bit Ein–/Ausgabe–Ports, einen 8–bit Eingabe–Port und einen 3–bit Ein–/Ausgabe–Port.

[3]Genauer gesagt, ist es ein Hitachi H8/3292 mit einem LEGO statt eines Hitachi Logos.

Diese Ports sind so programmiert, daß drei 8–bit Eingänge und drei 8–bit Ausgänge zur Verfügung stehen.

Weiter sind bereits eine serielle Schnittstelle, verschiedene Timer, Analog/Digital Wandler, 512 Byte RAM sowie 16 kByte ROM auf diesem Mikrocomputer integriert. Das ROM ist vorprogrammiert und enthält so etwas wie ein "Mini–Betriebssystem", um z.B. die Firmware in den RCX zu laden, Töne auszugeben oder Zahlen auf dem LC–Display auszugeben.

Dieser Hitachi Mikrocomputer ist mit 32 kByte externem RAM, einem LC–Display mit dazugehörigem Controller, drei Motortreibern, einem kleinen Lautsprecher, Infrarot Sende– und Empfangsdioden sowie ein wenig Logik erweitert worden.

Auf diese Weise erhält der RCX folgende Merkmale:

- 3 Ausgänge für Motoren und andere Aktoren
- 3 Eingänge für aktive wie auch passive Sensoren
- 4 Taster
- 1 (infrarote) serielle Schnittstelle
- 1 LC–Display
- interner Lautsprecher

Dies alles ist in einem Gehäuse untergebracht, das mit den herkömmlichen LEGO–Steinen verbunden werden kann.

Die Anschlüsse für die Ein– und Ausgänge sind ebenfalls in LEGO–Stein–Technik ausgeführt und elektrisch so beschaltet, daß man Sensoren und Aktoren nicht verpolt anschließen kann.

Das LC–Display gibt Auskunft darüber, welches Programm aktiv ist, über den Status des aktuellen Programmes, kann Sensordaten ausgeben, die Systemzeit anzeigen und einiges mehr.

Die vier Taster haben die Aufgabe den RCX ein– bzw. auszuschalten, ein Programm auszuwählen und dieses zu starten. Der vierte Knopf dient dazu, sich die Werte der Ein– bzw. der Ausgänge auf dem LC–Display anzeigen zu lassen.

Die Funktionen von LC–Display und Tastern beziehen sich allerdings auf die Verwendung der von LEGO mitgelieferten Firmware. Auf diese und die Alternativen wird in den nächsten Abschnitten eingegangen.

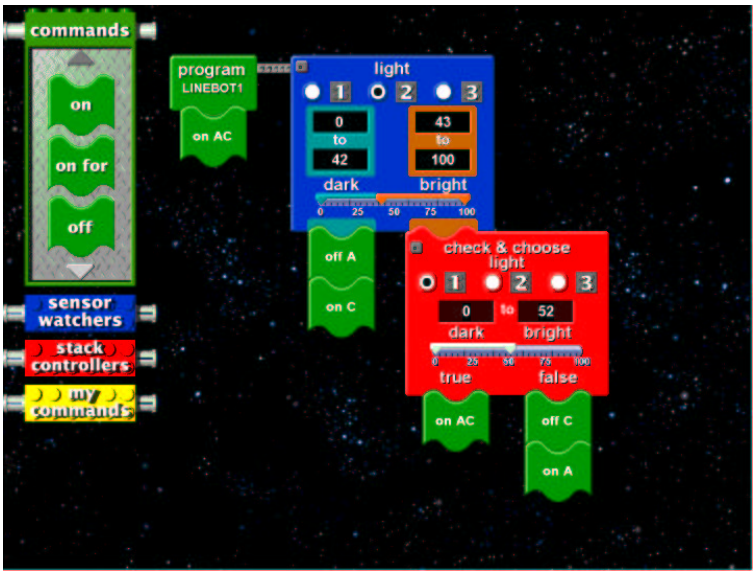

Abbildung 2.3: LEGO MINDSTORMS Programmierumgebung

LEGO Robotics Invention System Software

Die Software, die LEGO mit ihrem Robotics Invention System vertreibt, ist eine für Kinder ab zwölf Jahren gedachte, grafische Programmiersprache. Bevor man allerdings mit der Programmierung seines ersten Roboters und somit seines ersten RCX beginnen kann, muß man ein umfangreiches Repertoire an Übungsaufgaben bewältigen, die den RCX und die grafische Programmiersprache erklären und den Umgang mit beidem erleichtern soll.

In diesen Lektionen wird nicht nur das Programmieren des RCX erklärt, sondern ebenfalls die Inbetriebnahme des RCX und des Infrarotsender am PC.

Es werden Beispiele gegeben, wie man einen Roboter aus LEGO bauen kann und welche Aufgaben sich mit diesem Roboter lösen lassen.

Die LEGO Programmiersprache besteht aus einer Art LEGO–Programmier–Steinen, die intuitiv zu einem Programm zusammengesetzt werden können. Sie ist multitaskingfähig und erlaubt es, neben einem Haupt–Task weitere neun Tasks auf dem RCX laufen zu lassen. Diese LEGO–Programmier–Steine gliedern sich in drei Hauptgruppen. Dies sind die so genannten “Commands”, die “Sensor Watchers” und die “Stack Controllers”. Eine vierte Hauptgruppe heißt “My Commands”, in der öfter gebrauchte Kombinationen aus den anderen drei Hauptgruppen als eigenes Kommando abgespeichert werden können.

Jedes Element aus der Gruppe der “Sensor Watchers” bildet einen eigenen Task. Deshalb sind maximal 9 “Sensor Watchers” zulässig.

Nachdem ein Programm mit Hilfe der Programmierumgebung erstellt worden

ist, wird das erstellte Programm in einen “Bytecode” übersetzt, der anschließend über die Infrarotschnittstelle an den RCX gesendet wird. Startet man nun das Programm auf dem RCX, so wird der Bytecode im Speicher des RCX mit Hilfe der Firmware ausgeführt.

LEGO hat zusammen mit dem Robotics Invention System 2.0 eine neue Firmware herausgegeben. Diese Firmware wird Version 2.0[4] genannt. Diese Arbeit bezieht sich auf die Firmware der Version 2.0.

LEGO Firmware In diesem Abschnitt wird auf die Eigenschaften der LEGO Firmware 2.0 auf dem RCX näher eingegangen.

Die Firmware 2.0 ist ein Bytecode Interpreter. Dieser Interpreter führt Befehle auf unterster Ebene aus, egal ob diese Bytecodes aus dem gespeicherten Programm kommen oder über die Infrarotschnittstelle. Die Interpretation beruht hierbei auf einer virtuellen Maschine. Diese virtuelle Maschine ist ein wesentlicher Bestandteil der LEGO Firmware 2.0.

Die virtuelle Maschine hat folgende Eigenschaften:

- 5 Programmplätze
  - jeweils 1 Haupt–Task und 9 Neben–Tasks
  - 8 Sub–Routinen
- 32 persistente[5], globale Variablen, die von allen Tasks, Sub–Routinen und Programmplätzen gemeinsam (!) genutzt werden.
- 16 lokale Variablen für jeden Task. Diese lokalen Variablen sind nur in dem jeweiligen Task sichtbar.
- 4 System Timer, die von allen Tasks und Sub–Routinen geteilt werden. Die Timer laufen in 100 Millisekunden Schritten von 0 bis 32767 (0x7FFF). Danach fangen sie von vorne mit 0 an.
- 3 Eingänge
- 3 Ausgänge
- 1 Lautsprecher
- 1 LC–Display

[4]Die Datei firm0309.lgo bezeichnet die Firmware 1.0, während die Datei firm0328.lgo die Version 2.0 bezeichnet.

[5]Persistent bedeutet hier, daß diese Variablen nur nach Austausch der Batterien mit Null initialisiert werden.

NQC

Die Abkürzung NQC steht für "Not Quite C", also "nicht ganz C". Dies beschreibt die Fähigkeiten dieser Programmierumgebung ziemlich gut.

Diese von Dave Baum [Baum 2000; Baum u. a. 2000] entwickelte Programmierumgebung ermöglicht es, den RCX textbasiert, auf Basis der LEGO Firmware zu programmieren. Die hier benutzte Sprache ähnelt der Programmiersprache C, ist allerdings sehr beschränkt und besitzt weder den vollen Sprachumfang noch alle Datentypen der Sprache C [Baum].

Die Programmierumgebung NQC produziert einen Bytecode, entsprechend dem der LEGO Entwicklungsumgebung. Dieser wird ebenfalls per Infrarotschnittstelle an den RCX übertragen und dort von der virtuellen Maschine der LEGO Firmware ausgeführt. Das Programm der Abbildung 2.3 würde in NQC so aussehen[6]:

```
//linebot1.nqc
//turn by stopping one tread

// sensor and motors
#define EYE     SENSOR_2
#define LEFT    OUT_A
#define RIGHT   OUT_C

//thresholds
#define LEFT_THRESHOLD      42
#define RIGHT_THRESHOLD     53

task_main()
{
    SetSensor(EYE, SENSOR_LIGHT);
    On(LEFT+RIGHT);

    while(true)
    {
        if (EYE <= LEFT_THRESHOLD)
        {
            Off(LEFT);
            On(RIGHT);
        }
        else if (EYE >= RIGHT_THRESHOLD)
        {
            Off(RIGHT);
            On(LEFT);
        }
```

[6]Dieses Programmbeispiel stammt aus Dave Baums Buch *Definitive Guide to LEGO MINDSTORMS* [Baum 2000] mit freundlicher Genehmigung von Dave Baum. Danke Dave !!

```
            else
            {
                On(LEFT+RIGHT);
            }
        }
    }
```

Da NQC auf der LEGO Firmware beruht, stehen NQC die selben Ressourcen zur Verfügung wie der LEGO Entwicklungsumgebung. So gibt es bei NQC ebensowenig Fließkomma–Arithmetik wie bei der LEGO Entwicklungsumgebung. Allerdings gibt es im Gegensatz zur LEGO Entwicklungsumgebung, die nur für Windows verfügbar ist, auch NQC Versionen für Linux, MacOS oder MS-DOS.

Möchte man also die Fähigkeiten des Hitachi H8 Mikrocomputers richtig nutzen, muß die LEGO Firmware ersetzt werden. Diesen Gedanken haben einige Leute aufgegriffen und Ersatz für die LEGO Firmware geschaffen. Mit diesen "Mini–Betriebssystemen" beschäftigen sich die nächsten Abschnitte.

Alternativen zur LEGO Firmware

Die eben vorgestellte Programmierumgebung NQC läßt schon erahnen, daß sich der LEGO RCX nicht nur unter 12+ jährigen Kindern großer Beliebtheit erfreut. Mittlerweile hat sich eine sehr große Fangemeinde gebildet, die im *LUGNET* [LUGNET] (LEGO User Group NETwork) ein Forum gefunden hat. Hier sei besonders die "Robotics News Group" [LUGNews] erwähnt. Diese Gruppe beschäftigt sich mit allem, was mit LEGO–Robotern zu tun hat.

Das LUGNET beschreibt sich selbst folgendermaßen:

> "Global community of LEGO enthusiasts
> LUGNET unites LEGO fans worldwide through discussion groups, web pages, and services. As an independent site by fans, for fans, it is neither owned nor operated by the LEGO Company."

Diese Fangemeinde hat mittlerweile erstaunliche Dinge hervorgebracht, zu denen natürlich auch Dave Baums NQC zählt. Im Rahmen des LUGNET sind einige, die Firmware übertreffende, Programmierumgebungen entwickelt worden, auf die, zumindest teilweise, als nächstes eingegangen wird. Neben erweiternder Software sind bereits einige Aktoren und Sensoren im LUGNET entwickelt worden, auf die in Abschnitt 2.3.2 verwiesen werden.

LegOS LegOS wurde 1998 von dem deutschen Studenten Markus L. Noga ins Leben gerufen. Zum Zeitpunkt der Erstellung dieser Arbeit sind 12 Personen weltweit an der Weiterentwicklung dieses Firmware–Ersatzes beteiligt [LegOS].

LegOS ersetzt die LEGO Firmware vollständig. Anstatt einen Bytecode auf dem RCX ausführen zu lassen, wird bei LegOS mit Hilfe eines C–Cross–Compilers ausführbarer Code für den Hitachi H8 Mikrocomputer erzeugt. Der Code wird dabei von dem in Linux–Kreisen üblichen gcc Compiler übersetzt. Dies hat den Vorteil, daß die Restriktionen der LEGO Firmware aufgehoben werden. Der Bytecode–Interpreter der LEGO Firmware ist im Gegensatz zu LegOS sehr langsam. Es mußte allerdings ein neues Betriebssystem inklusive der Ansteuerung aller Peripherie entworfen werden.

Dies sind die Gründe, weshalb sich die Möglichkeiten der LEGO Firmware deutlich von den Möglichkeiten, die LegOS bietet, unterscheiden.

LegOS ist ein *Open Source* Projekt. Jeder der gewillt ist, seinen eigenen Betriebssystemkern zu kompilieren, kann dies mit im Internet frei verfügbaren Werkzeugen tun. Hierbei kann er den Hitachi H8 Mikrocomputer seinen Wünschen entsprechend programmieren.

Dinge wie z.B. C++ Unterstützung, Fließkomma–Arithmetik, direkter Zugriff auf die Hardware, Prioritäten gesteuertes Multitasking oder Semaphoren sind für LegOS kein Problem mehr.

Weitere Qualitäten wie UDP–Netzwerk via Infrarot, Ereignisse, Timer–Funktionen und Ähnliches stehen bereits auf der Wunschliste der Entwickler und werden sicherlich in Kürze realisiert sein.

LegOS steht für Windows 95/98/2000 sowie Linux zur Verfügung.

LeJOS Ende 1999 entschloß sich Jose H. Solorzano, ein in den USA lebender Ecuadorianer, nachdem er vergeblich nach einer Java Runtime gesucht hatte, eine eigene virtuelle Java Maschine für den RCX zu erstellen. Diese nannte er, da sie für die Verwendung im RCX nicht sehr groß sein durfte, TinyVM (Tiny Virtuell Machine).

Diese TinyVM war die Grundlage von LeJOS, an deren Entwicklung neben Jose H. Solorzano zwölf weitere Entwickler arbeiten [LeJOS 2001].

LeJOS reiht sich zwischen der LEGO Firmware und LegOS ein. LeJOS lädt statt der LEGO Firmware eine ca. 16 kByte große virtuelle Maschine für Java auf den RCX. Mit dieser virtuellen Maschine kann dann Code, der mit Hilfe des von Sun frei verfügbaren *Java Development Kits* [SunJDK] erstellt worden ist, auf dem RCX ausgeführt werden.

Dies bedeutet, daß wieder ein Interpreter auf dem RCX läuft, nur diesmal nicht für den LEGO Bytecode, sondern für Java.

Die Geschwindigkeit liegt zwischen der LEGO Firmware und LegOS. LeJOS ist allerdings ähnlich leistungsfähig, wie LegOS.

Folgende Eigenschaften zeichnen LeJOS aus:
objektorientiertes Java (J++), preemptive Tasks, Fließkomma–Arithmetik mit Win-

kelfunktionen und Ähnlichem, multidimensionale Arrays, Rekursion, Synchronisation, Ausnahmefallbehandlung und einiges mehr.

Was noch ?!? Für alle weiteren Programmierumgebungen für den RCX sowie grafische Oberflächen für NQC, LegOS und LeJOS wird auf das LUGNET [LUG-News] sowie das Buch *Extreme Mindstorms* [Baum u. a. 2000] verwiesen.

Dort lassen sich bereits jetzt unter anderem der Forth–Dialekt pbforth (Programmable Brick Forth) und eine Smalltalk Art mit Entwicklungsumgebung namens BotKit finden. Für das LEGO ActiveX "spirit.ocx", auf das in Abschnitt 3.1 genauer eingegengen wird, gibt es mittlerweile PRO-RCX, BrickCommand, MindControl, GordonsBrickProgrammer und viele mehr. WebBrick ist ein Windows Programm, mit dem man einen RCX über das Internet fernsteuern kann.

2.3.2 Sensoren und Aktoren

Die MINDSTORMS Robotics Invention System Kästen werden bereits, wie in Abschnitt 2.3 beschrieben, mit einigen Sensoren und Aktoren ausgeliefert. Zu der Serienausstattung gehören als Aktoren zwei kräftige Getriebemotoren und als Sensoren zwei Taster sowie ein "Licht–Sensor".

Die Getriebemotoren können verschiedene Zahnräder oder Riemenscheiben aufnehmen, so daß sie beliebig über– oder untersetzt werden können, um Drehzahl und Kraft zu variieren. Unbelastet drehen diese Motoren mit ca. 350 min^{-1}.

Die LEGO Sensoren lassen sich in zwei Klassen einteilen, in aktive und in passive Sensoren.

Bei passiven Sensoren wird der Sensorwert mit 5 Volt am Eingang gemessen. Bei aktiven Sensoren wird die im Sensor integrierte Schaltung mit ca. 8 Volt gespeist. Alle drei Millisekunden setzt diese Versorgungsspannung für eine zehntel Millisekunde aus. Die interne Schaltung der aktiven Sensoren hält für diese Zeit die interne Versorgungsspannung aufrecht. In dieser "Lücke" wird der Sensorwert gelesen. Diese ausgefeilte Methode ermöglicht es, Sensoren die eine Versorgungsspannung benötigen, über eine Zwei–Draht–Leitung anzuschließen.

Die Taster sind passive Sensoren, die den Stromkreis mit Hilfe eines leitenden Kunststoffes schließen. Der Widerstand des Kunststoffes ist so hoch (1kΩ bis 2kΩ), daß sich die Taster nicht zum direkten Schalten von Leistung eignen. Sie sind nur als Sensoren für den RCX sinnvoll einsetzbar.

Der Licht–Sensor ist ein aktiver Sensor, der aus einer roten Leuchtdiode und einem auf Helligkeit reagierenden Foto–Transistor sowie einer dazugehörigen Logik besteht. Der Rückgabewert dieses Sensors entspricht der Helligkeit, die der Foto–Transistor mißt. Man kann mit ihm also die Helligkeit in seiner Umgebung messen oder aber durch die eingebaute LED die Menge des reflektierten Lichtes,

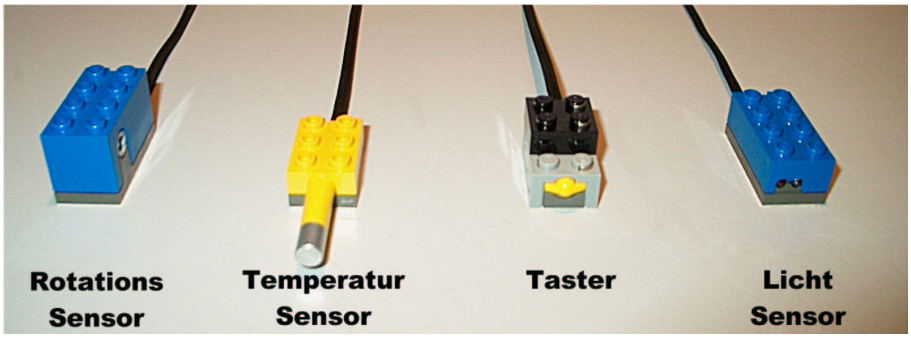

Abbildung 2.4: LEGO MINDSTORMS Sensoren

was Rückschlüsse auf den beleuchteten Untergrund zulässt.

Als weitere Sensoren im Rahmen der LEGO MINDSTORMS Serie kann man bei LEGO einen Temperatur– und einen Rotations–Sensor erhalten.

Der Temperatur–Sensor ist ein passiver Sensor, der in einem Bereich von $-20°C$ bis $+70°C$ gültige Meßwerte liefert.

Der Rotations–Sensor ist ein aktiver Sensor, der eine LEGO Achse aufnehmen kann und deren Umdrehungen mißt. Er liefert pro Umdrehung 16 Meßwerte und zwar je nach Drehrichtung aufwärts oder abwärts zählend. Dies bedeutet, auf die aufgenommene Achse bezogen, eine Genauigkeit von 22,5°.

Außerhalb des LEGO MINDSTORMS Programmes gibt es bei LEGO, besonders bei Pitsco LEGO dacta[7] [PLD], weitere Aktoren und Sensoren. Neben diversen kleineren und größeren Motoren und Lampen gibt es dort Sensoren für pH–Werte, Temperaturen ($-30°C$ bis $+130°C$), Spannungen ($-25V$ bis $+25V$), Luftfeuchtigkeit (0% bis 100%), Luftdruck (0kPa bis 200kPa) sowie Lautstärke (50dBA bis 100dBA).

Außerhalb von LEGO und Pitsco LEGO dacta sind bereits viele weitere Sensoren und Aktoren entwickelt worden. Diese Sensoren und Aktoren werden z.B. in [Gasperi; Balzerowski; Baum u. a. 2000] erklärt.

2.3.3 Infrarotkommunikation

Eine weitere Besonderheit, die den RCX auszeichnet, ist die integrierte Infrarotschnittstelle. Zusammen mit dem am PC angeschlossenen Infrarotsender wird so eine kabellose, serielle Kommunikation ermöglicht. Die folgenden Angaben beziehen sich auf die Verwendung der LEGO Firmware. Bei einem Firmware–Ersatz wie LegOS oder LeJOS liegt der Umgang mit der seriellen Infrarotschnittstelle in

[7] Pitsco LEGO dacta ist die Didaktikabteilung von LEGO und ist für den Einsatz von LEGO an Schulen und in der Forschung zuständig.

der Hand des Firmware–Ersatzes.

Bei Verwendung der LEGO Firmware werden gültige Kommandos, die von dem PC an den RCX geschickt werden, auf dem RCX genauso ausgeführt, wie Programme die vorher via Infrarot auf den RCX geladen wurden. Dies geschieht unabhängig davon, ob bereits ein Programm auf dem RCX läuft oder nicht.

Neben der Kommunikation zwischen dem PC und dem RCX und andersherum, kann man auch zwischen zwei RCX kommunizieren. Eine derartige Nachricht umfaßt ein Byte, es lassen sich also 255 verschiedene Nachrichten verschicken.[8] Diese Nachrichten lassen sich nicht adressieren. Das bedeutet, daß alle RCX die in Reichweite des sendenden RCX sind, diese Nachrichten empfangen. Möchte man mit mehr als zwei RCX untereinander Nachrichten austauschen, muß man die 255 Nachrichten sinnvoll unter den verwendeten RCX aufteilen, so daß jeder RCX nur auf die Nachrichten reagiert, die auch für ihn bestimmt sind. Ein von einem PC gesendetes gültiges Kommando wird allerdings auf allen RCX in Reichweite ausgeführt, was die Benutzung von mehreren RCX im Rahmen dieser Arbeit nicht zuläßt. Auf dieses Problem wird in Abschnitt 3.1 eingegangen.

2.4 Vision Command

Abbildung 2.5: LEGO Vision Command Kamera auf LEGO–Stativ

Ende 2000 brachte LEGO eine zur MINDSTORMS Serie gehörende Kamera auf den Markt. Dieser Kasten trägt den Namen “Vision Command”[9]. Deshalb wird diese Kamera auch Vision Command Kamera genannt.

Zu dem Vision Command Set gehört neben der Kamera ein 140 Teile umfassender Satz von LEGO–Teilen, mit dem sich beispielsweise das Stativ für die Kamera aus Abbildung 2.5 bauen läßt.

[8] Es lassen sich nur 255 statt 256 Nachrichten verschicken, da die Nachricht 0 nicht existiert.

[9] Der LEGO Vision Command Kasten trägt die Set–Nummer 9731

Die Software, die dem Vision Command Satz beiliegt, wird weiter unten in diesem Kapitel beschrieben.

2.4.1 LEGO Vision Command Kamera

Bei der Kamera, die LEGO mit ihrem Vision Command Set vertreibt, handelt es sich prinzipiell um eine Logitech QuickCam Web in einem speziellen, transparenten Gehäuse.

Dieses Gehäuse hat an den Seiten Befestigungsmöglichkeiten für "LEGO technic"–Steine sowie oben und unten Befestigungsmöglichkeiten für herkömmliche LEGO–Steine.

Die Tatsache, daß es sich bei der Vision Command Kamera im Kern um eine Logitech Kamera handelt, ermöglicht die Benutzung der Logitech Entwicklungsumgebung. Dieses "Software Development Kit" (SDK) ist auf der Logitech Homepage [LTQCSDK] für jeden interessierten Entwickler frei verfügbar. Auf das Logitech SDK wird in Abschnitt 3.2 näher eingegangen.

Die Kamera hat einige zusätzliche Ausstattungsmerkmale. Dies sind ein Mikrophon, eine schaltbare Leuchtdiode sowie einen Taster.

An der Vorderseite ist ein Ring zum manuellen Einstellen des Fokus vorgesehen. Eine Autofokusoption gibt es für diese Kamera nicht.

Das Kabel, das die Kamera mit der USB–Schnittstelle verbindet, ist ca. fünf Meter lang und für ein USB–Kabel ungewöhnlich steif. Aufgrund der massiven, fest gewebten Abschirmung hat das Kabel den Vorteil, daß es nicht so leicht beschädigt werden kann. Auf der anderen Seite hat es den Nachteil, daß dieses Kabel mobile Roboter, die die Kamera bei sich tragen wollen, behindert.

Die Vision Command Kamera unterstützt drei verschiedene Auflösungen. Dies sind 160*120, 320*240 sowie 640*480 Bildpunkte, die jeweils nur mit 24 Bit Farbtiefe zur Verfügung stehen. Auf Begriffe wie Bildpunkt und Farbtiefe wird in Abschnitt 4.4.1 noch ausführlich eingegangen.

Die Kamera hat bereits hardwareseitig einige Möglichkeiten zur Bildkontrolle. So kann man über die Kamera z.B. Einstellungen für die Verstärkung, die Helligkeit oder den Weißabgleich vornehmen. Die Justierung dieser drei Werte kann sowohl manuell, d.h. über eine geeignete Software, geschehen als auch automatisch von der Kamera selbst. Andere Dinge wie ein Anti–Flimmer–Filter, eine Farb– oder eine Helligkeitsverstärkung sind per Hand einstellbar. Weiter gibt es z.B. die Möglichkeiten das Bild zu kippen oder zu spiegeln.

Um alle möglichen Einstellungen der Kamera zu überblicken, wird die Lektüre des *Logitech QuickCam SDK 1.0 Microsoft MFC Programmer's Guide* [Logitech 2000] empfohlen.

Die Kamera liefert ein Bild, wie es von einer Webcam zu erwarten ist. Die Bildrate der Kamera hängt ganz wesentlich von der verwendeten Auflösung ab.

Den besten Kompromiß zwischen Auflösung und Bildrate stellt die Auflösung von 320*240 Pixel dar.

Das Bild der Kamera wirkt etwas unscharf und farbschwach und weist die für eine schlechte Optik typische Abschattung der Ränder auf.

LEGO Vision Command Software

Die Software, die LEGO mit ihren Vision Command Sets ausliefert, ähnelt der der Robotics Invention System Kästen. Neben dem eigentlichen Programm gibt es auch hier eine Training genannte Einführung, die die Benutzung der Software im Detail erklärt. Zusätzlich gibt es einen Challenges genannten Bereich. In diesem Bereich kann man sich den von LEGO Entwicklern erdachten Herausforderungen stellen. Diese muß man allerdings nicht auf sich allein gestellt lösen, sondern wird Schritt für Schritt zu einer Lösung geführt.

Die Vision Command Software ist wie die Robotics Invention System Software für Kinder ab 12 Jahren gedacht. Auch hier wird über möglichst intuitive LEGO–Programmier–Steine programmiert.

Die Vision Command Software funktioniert sowohl mit als auch ohne einen RCX aus einem Robotics Invention System Kasten. Die LEGO–Programmier–Steine, die sich auf den RCX beziehen, sind aber nur dann verfügbar, wenn das Programm mit Hilfe des Infrarotsenders einen RCX finden konnte.

Die Idee, die hinter der Vision Command Software steckt, beruht auf einem Schema von Regionen, denen man eine bestimmte Bedingung und einen Satz von Kommandos zuweisen kann. In Abbildung 2.6 wird ein Schema mit acht solchen Regionen benutzt. Von diesen Schemata gibt es zwanzig Stück, die sowohl eine unterschiedliche Anzahl als auch eine unterschiedliche Anordnung von Regionen haben können.

Jeder einzelnen Region kann nun eine Bedingung zugeordnet werden. Diese Bedingungen können jeweils sein:

- Farbe (angelernt)
- Licht
- Bewegung

Das bedeutet, daß eine Region, der man beispielsweise eine Farbe beigebracht hat, *auslöst*, wenn die angelernte Farbe in dieser Region zu finden ist. Genauso funktioniert es, wenn eine Region auf Bewegung reagieren soll. Hat in dieser Region eine Bewegung von bestimmtem Ausmaß stattgefunden, *löst* diese Region aus. Auf Licht wird im Prinzip so reagiert, als hätte man die Farbe Weiß angelernt und vor einem dunklen Hintergrund verwendet.

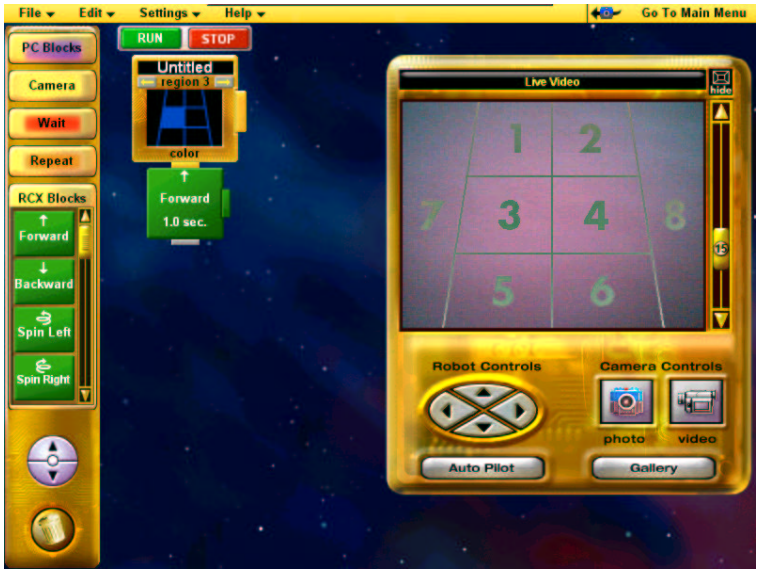

Abbildung 2.6: LEGO Vision Command Software Prinzip

Löst eine Region aus, führt die Software die Kommandos aus, die durch die Programmier–Steine für die entsprechende Region "programmiert" wurden.

In dem in Abbildung 2.6 gezeigten Beispiel ist für die Region Nummer drei ein Kommando ausgewählt worden, das den RCX dazu veranlaßt, eine Sekunde vorwärts zu fahren[10].

Es gibt fünf verschiedene Blöcke von Kommandos. Ein Block widmet sich den Kommandos, die auf dem PC ausgeführt werden. In ihm sind Kommandos zu finden, mit denen man Töne und Musik auf dem PC ausgeben kann. In einem weiteren Block befinden sich Kommandos, mit denen man Fotos oder Videos aufnehmen kann. In zwei weiteren Blöcken befinden sich Kommandos, mit denen man den Programmablauf steuern kann. Im fünften Block befinden sich Kommandos, die den RCX betreffen.

Um die Beschränktheit der Fähigkeiten eines RCX gesteuerten Roboters im Zusammenhang mit der Vision Command Software zu erkennen, muß man wissen, daß ein auf dem RCX laufendes Programm angehalten wird, sobald von der Vision Command Software über den Infrarotsender der RCX gefunden wurde. Das hat zur Folge, daß man den RCX nicht im Vorweg sinnvoll programmieren und dann die Fähigkeiten der Vision Command Software nutzen kann. Statt dessen stehen ausschließlich die Kommandos der Vision Command Software zur Verfügung.

[10]Die Software geht hier davon aus, daß der RCX nach einem bestimmten Muster in einem mobilen Roboter verbaut worden ist.

In Abbildung 2.7 ist eine Abfolge von Ausschnitten der Vision Command Software zu sehen.

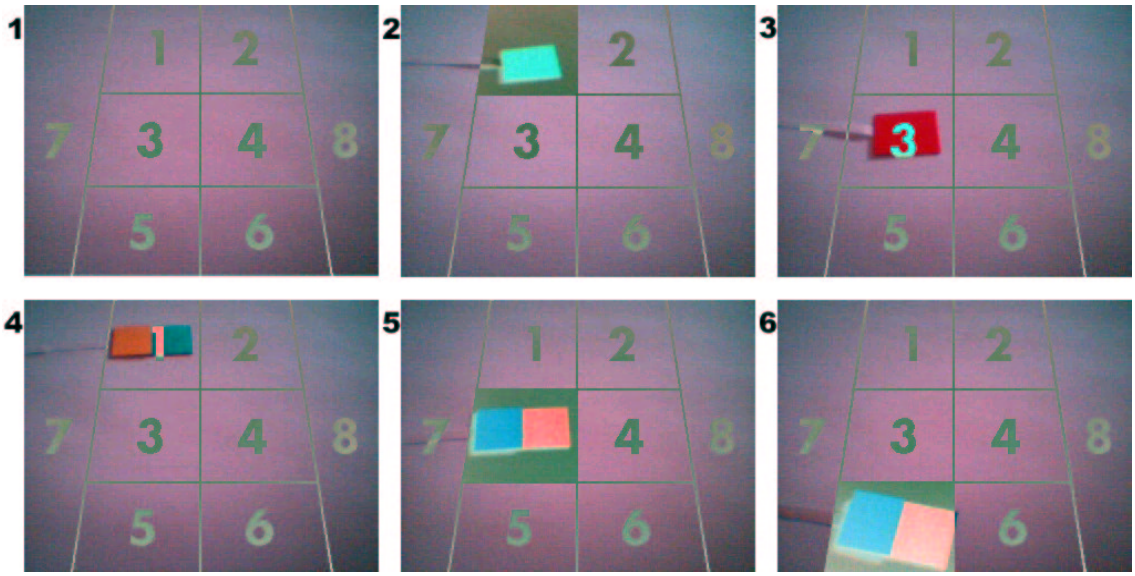

Abbildung 2.7: Funktionsweise der LEGO Vision Command Software

Diese Bilder zeigen ein Schema mit acht Regionen. Diese Regionen sind zum Teil mit Farben angelernt.

Die Region eins ist mit Rot angelernt, die Region drei mit Türkis und die Region fünf mit Gelb.

Bild 1 zeigt das Schema ohne farbige Flächen, die eine der Regionen auslösen könnte.

In Bild 2 löst eine rote Farbfläche die Region eins aus. Der Inhalt der Region wird dann invertiert dargestellt.

Die rote Farbfläche in Bild 3 kann die Region drei nicht auslösen.

In Bild 4 ist eine türkis/gelbe Farbfläche in Region eins. Auch hier wird natürlich nicht ausgelöst.

Bild 5 zeigt, daß die türkise Farbfläche die Region drei ausgelöst hat.

In Bild 6 schließlich sieht man wie die türkis/gelbe Fläche auch die Region fünf auslöst.

Löst eine Farbe eine Region aus, so wird das Kommando, daß zu der entsprechenden Region gehört, ausgeführt.

Kapitel 3

Software Plattform

Dieses Kapitel beschreibt die softwareseitigen Rahmenbedingungen dieser Diplomarbeit.

3.1 Schnittstelle PC⇔Roboter

Als Schnittstelle zum Roboter und somit zum RCX kommt nur die Infrarotkommunikation in Frage.

Zum Zeitpunkt der Entstehung dieser Arbeit gibt es nur zwei Möglichkeiten via Infrarot mit dem RCX zu kommunizieren.

Eine Möglichkeit ist, die bereits vorhandenen, rudimentären Netzwerkfähigkeiten von LegOS zu nutzen. Leider sind diese Fähigkeiten aber noch weit von einer Qualität entfernt, die einen Einsatz im Rahmen dieses Projektes gerechtfertigt hätte. Dies hat den negativen Nebeneffekt, daß ein Einsatz von LegOS im Moment noch nicht möglich ist.

Als Alternative zur Infrarotkommunikation von LegOS gibt es noch die LEGO–Firmware und die zu ihr gehörende Form der Infrarotkommunikation.

Das Protokoll, daß LEGO hierfür verwendet, wird sehr ausführlich in den "RCX Internals" [Proudfoot] von Kekoa Proudfoot erklärt. LEGO sei Dank, muß man dieses Protokoll nicht selbst implementieren, sondern kann auf eine *Spirit Control* genannte ActiveX Komponente zurückgreifen.

Diese wird von LEGO zur Verfügung gestellt und ist als Datei SPIRIT.OCX auf den CDs der LEGO Robotics Invention System Kästen vorhanden[1].

Das Spirit Control ist eine ActiveX Komponente, die für viele Programmiersprachen zur Verfügung steht. So läßt sie sich mit Microsofts Visual Basic und

[1] Das Spirit Control ist konzipiert für den seriellen Infrarotsender. Ab dem Robotics Invention System 2.0 ist allerdings ein USB–Infrarotsender enthalten. Leider ist das Spirit Control auf den CDs dieser Version nicht mehr vorhanden.

Visual C++ einsetzen, aber auch mit allen anderen Programmiersprachen, die ActiveX Komponenten unterstützen, z.B. Borland Delphi oder Borland C++ Builder. Die technische Referenz [LEGO 1998] für diese Komponente bietet allerdings nur Beispiele für Microsoft Visual Basic.

Die Kommunikation, die über dieses Spirit Control möglich ist, ist sehr einfach gehalten. Kommandos, die (infrarot) abgesetzt werden, werden von jedem RCX empfangen, bearbeitet und beantwortet. Eine Adressierung der Kommandos ist nicht möglich. Dies führt zwangsläufig zu Kollisionen. Leider ist aber keine Kollisionserkennung implementiert, weshalb die Kommunikation zwischen dem Infrarotsender mit mehr als einem RCX scheitert.

Ein empfangenes Kommando wird automatisch von der Firmware beantwortet. Als einzige Flußkontrolle dient ein Bit in dem "Opcode". Über den Opcode wird das Kommando identifiziert. Er macht ein Byte des Kommandos aus.

Die Flußkontrolle findet über das vierte Bit (8_{hex}) statt. Es ist entweder gesetzt oder nicht gesetzt.

Empfängt ein RCX ein Kommando, bei dem das vierte Bit des Opcodes gesetzt ist, reagiert er mit einem gesetzten vierten Bit des Opcodes der Antwort.

Empfängt der RCX jetzt wieder ein Kommando mit gesetztem vierten Bit des Opcodes, so wird dieses Kommando ignoriert und es wird die Antwort gesendet, die zum zuvor empfangenen Kommando gehörte.

Erst ein Kommando, bei dem das vierte Bit des Opcodes nicht gesetzt ist, wird wieder ausgeführt. Folglich ist auch bei der Antwort das vierte Bit des zur Antwort gehörenden Opcodes nicht gesetzt.

Die einzelnen Opcodes der Kommandos und der dazugehörenden Antwort sind ausführlich in den oben bereits erwähnten "RCX Internals" [Proudfoot] von Kekoa Proudfoot erklärt.

Die Kommandos des Spirit Control selbst werden in der zu ihr gehörenden technischen Referenz [LEGO 1998] anhand von VisualBasic Beispielen ausführlich beschrieben.

Sie werden in insgesamt dreizehn Gruppen unterteilt:

- Communication control commands

  Zuständig für die Kommunikation der Software mit dem Infrarotsender und dem RCX.

- Firmware control commands

  Zum Herunterladen und Aktivieren der LEGO Firmware.

- Diagnostics commands

  Prüfen die Verbindung zum Infrarotsender und/oder dem RCX.

- PBrick system commands

  Erlauben Zugriff auf die Systemkomponenten des RCX.

- PBrick output control commands

  Kontrollieren die drei Ausgänge. Größen wie Richtung, Leistung, Einschaltzeiten usw. können beeinflußt werden.

- PBrick input control commands

  Setzen die Eingänge auf einen Typen und einen Modus.

- PBrick program control commands

  Wählen das aktuellen Programm aus oder löschen Tasks.

- PBrick program execution commands

  Starten oder Stoppen von Tasks.

- PBrick flow control commands

  Für die Flußkontrolle, wie z.B. *If(), Else,* oder *While()*

- PBrick arithmetic/logical commands

  Zum direkten Schreiben in die 32 persistenten Variablen des RCX sowohl direkt als auch verknüpft mit einer arithmetischen/logischen Funktion.

- PBrick query commands

  Zuständig für das Auslesen der 32 persistenten Variablen und zur Kontrolle von Ereignissen.

- PBrick data acquisition commands (RCX only)

  Benutzen den "DATALOG" genannten Speicher, der zum Ablegen von Sensorinformationen dient, den der RCX allerdings nicht selbst auslesen kann. Diese können aber über das Spirit Control von einem PC ausgelesen werden.

- ActiveX control commands

  Dienen zum Setzen oder Auslesen der Priorität des Spirit Control.

3.2 Schnittstelle PC⇔Kamera

Für die Übergabe von Daten eines Aufnahmegerätes an eine Applikation gibt es eine Vielzahl von Möglichkeiten unter Windows.

Bereits 1990 formierte sich die "TWAIN" Arbeitsgruppe, die es sich zum Ziel setzte, einen offenen Industriestandard zur direkten Einbindung bilddigitalisierender Hardware in Anwendungen zu erarbeiten. Im Frühjahr 1992 stellte diese Gruppe, die aus den Firmen Aldus, Caere, Eastman Kodak, Hewlett–Packard und Logitech bestand, die TWAIN 1.0 Spezifikation vor [Kunze 1994].

Gedacht ist die TWAIN–Schnittstelle für die Aufnahme einzelner Bilder, wie sie Digitalkameras oder Scanner zur Verfügung stellen. Die LEGO Vision Command Kamera unterstützt die TWAIN–Schnittstelle, da aber Videodaten verarbeitet werden sollen, ist sie für die zu entwickelnde Software nicht geeignet.

Ebenfalls 1992 vertrieb Microsoft mit Windows 3.1 zum ersten Mal die Schnittstelle "Video for Windows" (VfW). Diese Schnittstelle wurde konzipiert, um Videodaten auf Festplatte zu speichern.

Mittlerweile stellen Videoanwendungen allerdings so hohe Anforderungen, daß VfW an einigen Stellen nicht mehr ausreicht. Aus diesem Grund hat Microsoft die "Windows Driver Model" Schnittstelle (WDM) entwickelt, die seit Windows 98 zum Betriebssystem gehört [WDM 1998].

Geräte mit VfW–Treibern werden allerdings weiterhin unterstützt, da eine große Anzahl dieser Geräte bereits in Benutzung sind. Auf der anderen Seite unterstützen VfW–Anwendungen aber auch Geräte mit WDM–Treibern. Diese werden dann über einen VfW–WDM–Mapper auf VfW abgebildet [VfW2WDM 1998], .

Mit der WDM–Schnittstelle läßt sich das DirectShow Software Development Kit (SDK) aus der DirectX Umgebung einsetzen.

Da es sich bei der LEGO Kamera prinzipiell um eine Logitech Kamera handelt, kann auch das Logitech SDK eingesetzt werden. Dies ist auf den Internetseiten von Logitech [LTQCSDK] frei erhältlich. Das Logitech SDK ist eigens für Logitech Kameras entwickelt worden und läßt sich deshalb leider nur mit diesen einsetzen. Dafür ist aber der Zugang zu allen Kameradaten sehr schnell und einfach möglich.

Logitech beschreibt in ihrem zum SDK gehörenden "Programmer's Guide" [Logitech 2000] die Vor– und Nachteile der Verfahren folgendermaßen:

1. Video for Windows
   + De facto standard for video capture on Windows
   - No access to camera-specific features
   - Not easy to learn
   - Minimal: Many common tasks require additional programming

2. DirectShow
   - Not easy to learn
   - Minimal: Many common tasks require additional programming
3. Logitech QuickCam SDK
   + Full support of all camera features
   + Highly accessible from many languages
   + Simple interface to learn and use
   + Advanced features such as text overlaying and motion detection
   + Allows multiple simultaneous camera connections
   - Proprietary: Will only work with Logitech QuickCam products

Da die Software für den Einsatz mit der LEGO Vision Command Kamera gedacht ist, wiegt der Nachteil, daß dieses SDK nur mit Logitech Kameras funktioniert, nicht so schwer.

Die Vorteile, die dieses SDK mit sich bringt, haben die Wahl auf das Logitech SDK fallen lassen. Dieses SDK ist für die Betriebssysteme Windows 98, Windows 2000 und Windows Millennium Edition entwickelt worden. Es steht ebenfalls als AvtiveX Komponente zur Verfügung und kann laut "Programmer's Guide" [Logitech 2000] mit Microsoft Visual C++, Microsoft Visual Basic oder anderen "high–level" Prorammiersprachen benutzt werden.

3.3 Betriebssystem

Da die LEGO Kamera einen USB–Anschluss benötigt, kommen grundsätzlich nur Betriebssysteme mit USB–Unterstützung in Frage, für die auch ein geeigneter Treiber für diese Kamera existiert.

Es soll das Spirit Control von LEGO eingesetzt werden. Über die Einsetzbarkeit des Spirit Controls stehen leider keine Informationen zur Verfügung.

Um beiden Anforderungen gerecht zu werden, liegt es nahe, die Bedingungen die LEGO für ihr MINDSTORMS Vision Command und MINDSTORMS Robotics Invention System Set stellt, auch an die zu entwickelnde Software zu stellen.

LEGO setzt als Mindestanforderung für beide Sets Microsoft Windows 98 voraus.

Aus diesem Grund soll die Software auf der Grundlage des Betriebssystems Windows 98 entstehen.

3.4 Entwicklungsumgebung

Im Logitech SDK "Programmer's Guide" [Logitech 2000] wird nur Microsoft Visual C++ und Microsoft Visual Basic erwähnt. Deshalb kommen nur diese beiden Entwicklungsumgebungen in Frage.

Da die Programmiersprache C++ als schneller und für technische Anwendungen besser geeignet gilt, ist für diese Arbeit eine Lizenz von Microsoft Visual C++ 6.0 der Hochschule für Angewandte Wissenschaften Hamburg eingesetzt worden.

3.5 Zusammenfassend

Die softwareseitigen Rahmenbedingungen können folgendermaßen zusammengefaßt werden. Als Betriebssystem wird Windows 98 eingesetzt. Die Software wird mit Microsoft Visual C++ entwickelt. Auf dem RCX muß die LEGO Firmware verwendet werden. Für die Infrarotkommunikation wird das LEGO Spirit Control benutzt. Um auf die Videodaten zugreifen zu können, wird das Logitech Software Development Kit (SDK) eingesetzt.

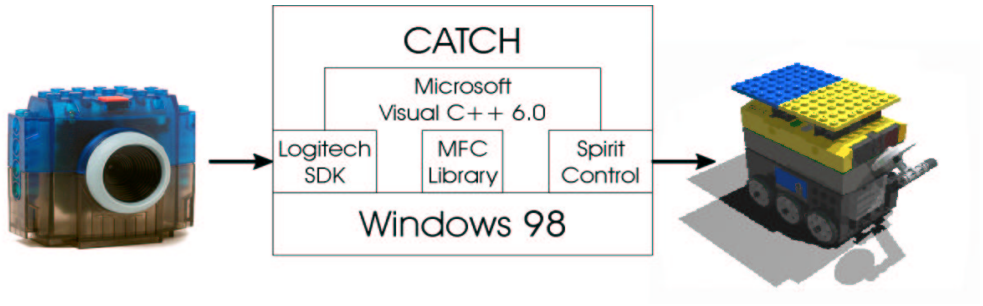

Abbildung 3.1: Rahmenbedingungen der Software (CATCH)

Kapitel 4

Konzept

Wie bereits in der Einleitung erwähnt, orientiert sich diese Arbeit an der Cognachrome Karte von newton labs. Diese Karte wird von den newton labs selbst im Rahmen der “Small Size Robot Soccer League” [RoboCup] eingesetzt. Bei diesem Roboter–Fußball hängt eine Kamera an einem Stativ über der Spielfläche. Aus dieser Vogelperspektive werden sowohl die teilnehmenden Roboter, als auch der Ball selbst beobachtet. Die Cognachrome Karte setzt diese Bilder in die Informationen um, die zur eigenen Mannschaft gehörende Roboter, benötigen. Diese Informationen werden dann über eine Funkverbindung an die Roboter geschickt.

Dieses Verfahren führte zu der Idee, etwas Vergleichbares mit LEGO–Mitteln zu bewerkstelligen. Sowohl die Kamera, als auch der Roboter und die Verbindung zwischen PC und Roboter sollen auf LEGO Technik basieren.

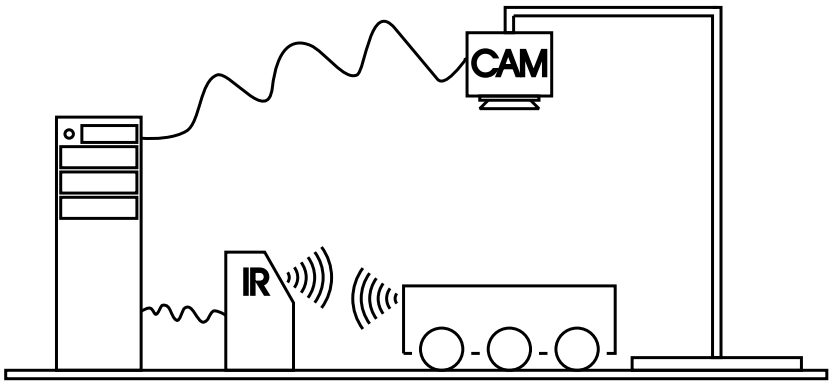

Abbildung 4.1: Abstraktes Konzept

Eine Funkverbindung gibt es bei LEGO leider nicht. Deshalb werden die Informationen mit Hilfe der Infrarotschnittstelle übertragen. Das führt zwar zwingend zu einer Sichtverbindung, stellt aber deshalb kein Problem dar, da auch die

Kamera eine Sichtverbindung zu dem Roboter haben muß. Aus diesem Grund liegt es nahe, den Infrarotsender nahe der Kamera anzubringen.

Die von der Kamera aufgenommenen Bilder werden dann von der Software analysiert und die resultierenden Informationen per Infrarot an den im Roboter verbauten RCX gesendet.

4.1 Roboter mit Schnittstelle

Roboter, die mit der im Rahmen dieser Diplomarbeit entstehenden Software zusammenarbeiten wollen, müssen einen RCX als Logik verwenden. Dieser Baustein hat, wie in Abschnitt 2.3.1 beschrieben, eine Infrarotschnittstelle mit der sich Informationen von einem PC zu einem RCX übertragen lassen.

In Abschnitt 3.1 ist bereits beschrieben worden, daß für diese Kommunikation nur das Spirit Control sinnvoll eingesetzt werden kann. Die ebenfalls in Abschnitt 3.1 erklärten Kommandos werden zur Übertragung der X–Y–Koordinaten der sechs Farbflächen sowie eines Synchronisierungsflags verwendet. Über dieses Flag läßt sich die Anforderung der Daten von der Kamerasoftware steuern. Auf diese Weise ist gewährleistet, daß die Daten, die von der Kamerasoftware kommen, auch gültig und nicht veraltet sind. Weiter werden über diese Kommandos Tasks auf dem RCX gestartet bzw. angehalten.

4.2 LEGO Kamera und Logitech SDK

Mit Hilfe des Logitech SDK kann man auf die Videodaten der LEGO Kamera zugreifen. Das Logitech SDK beinhaltet eine ActiveX Komponente, die von Logitech VideoPortal genannt wird. Man kann über dieses VideoPortal nicht nur auf die Bilddaten zugreifen, sondern auch auf alle Größen, die zur Einstellung der Kamera dienen. Dazu gehören Verstärkung, Belichtung, Weißabgleich und einiges mehr.

Die Bilddaten, die dieses VideoPortal zur Verfügung stellt, liegen im DIB Format vor. DIB steht für “Device–Independent–Bitmap”. Dieses Format ist erstmals als Erweiterung des *IBM® OS/2® Presentation Manager Bitmap–Formates* aufgetaucht und ist dafür gedacht, Bitmap–Grafiken geräteunabhängig zu machen. Mittlerweile sind eigentlich alle Bitmap–Grafiken als DIB ausgeführt, ohne dies explizit anzugeben.

Neben diesen Bilddaten besitzt das VideoPortal die Möglichkeit, Nachrichten über Kameraereignisse abzusetzen. Auf diese Weise kann das VideoPortal melden, daß ein neues Bild vorliegt, daß der Taster an der Kamera gedrückt wurde oder daß die Kamera vom USB–Port entfernt wurde.

Diese besondere Eigenschaft wird für die Kameraschnittstelle genutzt. Sobald die Nachricht über ein neues Bild vorliegt, wird dieses Bild an die Bildverarbeitung weitergeleitet.

4.3 Konzept der Bildverarbeitung

Das aus der Kamera stammende Bild wird in mehreren Schritten in die Koordinaten der einzelnen Farbflächen überführt.

Als erstes wird mit Hilfe eines Referenzbildes eine Korrektur von inhomogener Beleuchtung und Kameraschwächen durchgeführt. Die drei Kanäle des optimierten Bildes werden anschließend in zwei Schritten in Binärbilder gewandelt. Diese Binärbilder werden dann auf zusammenhängende Flächen untersucht. Aus ihnen werden die Größe und die Koordinaten der Schwerpunkte extrahiert.

4.4 Methoden der Bildverarbeitung

In den folgenden Abschnitten werden die bildverarbeitenden Verfahren vorgestellt. Als erstes wird das Bildformat, auf dem diese Verfahren beruhen, näher erklärt. Anschließend werden die einzelnen Methoden anhand eines Beispiels erläutert. Als Beispiel und Urbild für diese Methoden dient das Bild aus Abbildung 4.2. Dieses Urbild stammt von der LEGO Vision Command Kamera und wurde unter den in Abschnitt 2.2 beschriebenen Rahmenbedingungen aufgenommen. Es zeigt je eine rote und eine grüne Zielfläche und den gelb–blauen Roboter. Über die zwei Farben des Roboters kann nicht nur dessen Position, sondern auch dessen Orientierung festgestellt werden.

4.4.1 Einsatz des Bitmap Formates für die Bildverarbeitung

Wie bereits in Abschnitt 4.2 erwähnt, liefert das Logitech SDK eine Bitmap–Grafik an die Applikation. An dieser Stelle werden die Eigenschaften dieses Formates erklärt.

Digitale Bilder

Digitale Bilder werden durch eine Matrix mit N Spalten und M 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

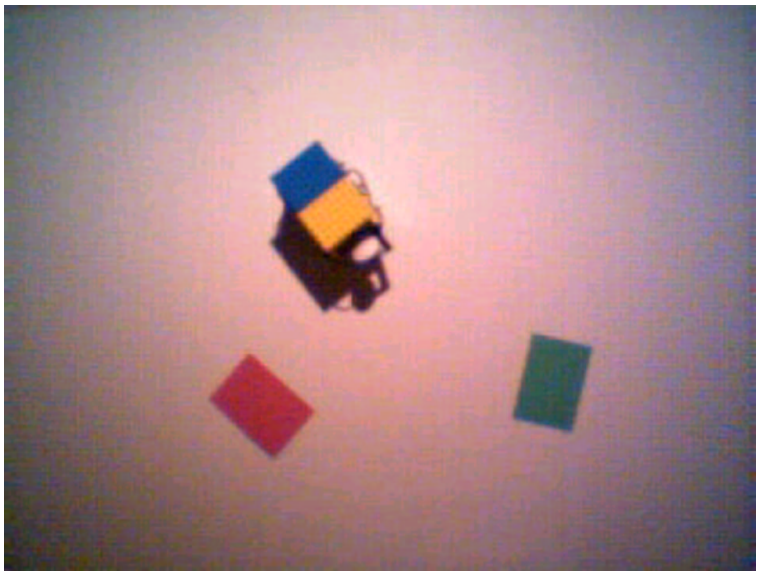

Abbildung 4.2: Urbild für die Erklärung der Bildverarbeitung

(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. Abbildung 4.3 zeigt ein einfaches Beispiel hierfür. Es zeigt die einzelnen Kanäle eines Bildes, das nach dem Prinzip der additiven Farbmischung entstanden ist.

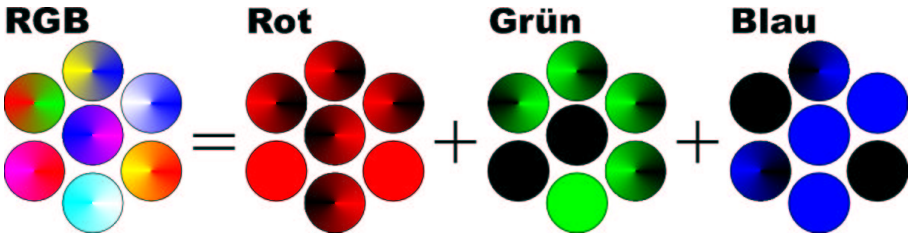

Abbildung 4.3: RGB Beispiel und dessen drei Farbkanäle

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.

Additive und Subtraktive Farbmischung Die additive Farbmischung basiert auf den Erkenntnissen des englischen Physikers Isaac Newton. Er erkannte, daß sich scheinbar farbloses Licht mit Hilfe eines Prismas in eine Vielzahl von Farben zerlegen läßt. Die Summe dieses farbigen Lichtes ergibt wieder farbloses also weißes Licht.

Um weißes Licht zu erzeugen, benötigt man die drei Farben Rot, Grün und Blau. Von diesen Farben stammt auch die Bezeichnung RGB, die ein solches System als additiv mischendes kennzeichnet. Aus diesen drei Farben lassen sich direkt die drei Farben Cyan (Blau+Grün), Magenta (Rot+Blau) und Gelb (Grün+Rot) mischen. Dies kann man der Abbildung 4.4 entnehmen.

Additive Farbmischung findet immer dann Anwendung, wenn Lichtquellen diese Farben erzeugen. So mischt z.B. jeder Monitor die Farben additiv. Auch eine Kamera nimmt Farben nach dem Prinzip der additiven Farbmischung auf.

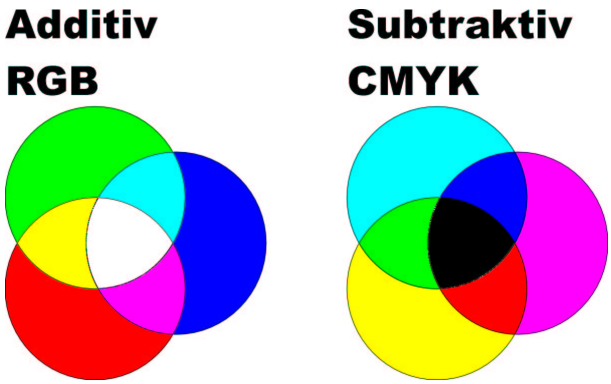

Abbildung 4.4: Additive und Subtraktive Farbmischung

Die subtraktive Farbmischung basiert auf der Tatsache, daß beleuchtete Gegenstände bestimmte Teile des Lichtes absorbieren können. Beleuchtet man beispielsweise einen Cyan farbigen Gegenstand[1], so wird der rote Anteil absorbiert und die blauen und grünen Anteile werden reflektiert.

Diese Prinzip wird als CMYK–Farbsystem bezeichnet. Hierbei stehen die vier Buchstaben für cyan, magenta, yellow und black.

Dieses Farbsystem bildet die Grundlage für den Farbdruck. Druckt man diese vier Farben geschickt neben- oder übereinander, kann man aufgrund der selek-

[1] Es wird davon ausgegangen, daß mit weißem Licht beleuchtet wird.

tiven Reflexion des Lichtes die meisten Farben darstellen. Schwarz ist eine zusätzliche Farbe in diesem Farbsystem, die deshalb aufgenommen worden ist, weil das Schwarz aus den drei Mischfarben nicht so gut deckt wie ein reines Schwarz. Eigentlich könnte man auf dieses Extra Schwarz verzichten und es durch die drei Mischfarben erzeugen. In Abbildung 4.4 ist das Prinzip der subtraktiven Farbmischung zu sehen.

Neben den hier erklärten Methoden der additiven und subtraktiven Farbmischung gibt es noch eine Vielzahl anderer Ansätze Farben zu mischen. Diese werden in Fachliteratur wie z.B. [Schlicht 1995] ausführlich erläutert.

Device–Independent–Bitmap (DIB)

Wie bereits in Abschnitt 4.2 erwähnt, liefert das Logitech SDK ein geräteunabhängiges Bitmap (Device–Independent–Bitmap oder DIB).

Üblicherweise besteht ein DIB aus mehreren Teilen. Ist ein DIB als Datei gelesen worden, beginnt ein DIB mit einem Dateikopf, der BITMAPFILEHEADER heißt. Danach folgt ein zweiter Kopf mit der Bezeichnung BITMAPINFOHEADER. Darauf folgt eine Farbtabelle und dann erst die eigentlichen Bilddaten.

Der BITMAPFILEHEADER Der BITMAPFILEHEADER ist ein 14 Byte Datensatz, der Informationen über den Typ und die Größe des Bitmaps enthält.

```
typedef struct tagBITMAPFILEHEADER { // bmfh
    WORD    bfType;
    DWORD   bfSize;
    WORD    bfReserved1;
    WORD    bfReserved2;
    DWORD   bfOffBits;
} BITMAPFILEHEADER;
```

- `bfType`

  muß zur Kennzeichnung der Datei als Bitmap immer den Wert BM oder eben $42_{hex}4D_{hex}$ enthalten. Andere Werte sind nicht zulässig.

- `bfSize`

  enthält die Dateigröße insgesamt in Bytes.

- `bfReserved1` und `bfReserved2`

  sind reserviert und müssen auf Null gesetzt werden.

- `bfOffBits`

  kennzeichnet die Entfernung vom Dateianfang bis zum Anfang der Bilddaten in Bytes.

Der BITMAPINFOHEADER Der auf den BITMAPFILEHEADER folgende BITMAPINFOHEADER trägt eine Menge an Informationen. Diese werden allerdings nur teilweise genutzt. Dieser Datensatz besteht aus 40 Byte, die folgendermaßen strukturiert sind:

```
typedef struct tagBITMAPINFOHEADER{ // bmih
    DWORD  biSize;
    LONG   biWidth;
    LONG   biHeight;
    WORD   biPlanes;
    WORD   biBitCount;
    DWORD  biCompression;
    DWORD  biSizeImage;
    LONG   biXPelsPerMeter;
    LONG   biYPelsPerMeter;
    DWORD  biClrUsed;
    DWORD  biClrImportant;
} BITMAPINFOHEADER;
```

- `biSize`

  enthält die Größe des BITMAPINFOHEADERS. Da dieser Header immer gleichgroß ist, ist dieser Wert immer 28_{hex}.

- `biWidth` und `biHeight`

  geben die Breite und die Höhe der Bitmap–Grafik in Anzahl Pixel an. Der Wert `biHeight` gibt neben der Höhe auch die Entstehungsrichtung des Bitmaps an. Ist dieser Wert negativ, handelt es sich um ein “top–down” –Bitmap. Diese Bitmaps haben ihren Ursprung oben links. Bei einem positiven Wert ist es ein “bottom–up” –Bitmap und der Ursprung ist unten links.

- `biPlanes`

  enthält die Anzahl der verwendeten Ebenen. Da Bitmaps nur eine Ebene haben dürfen, muß dieser Wert auf 1 gesetzt werden.

- `biBitCount`

  gibt die Farbtiefe des Bitmaps an. Zulässige Werte sind 1, 4, 8, 16, 24 und 32. Diese Werte lassen, zwischen Schwarz–Weiß und über vier Milliarden Farben, einige Varianten zu.

- `biCompression`

wird benutzt, um anzugeben, ob das vorliegende Bitmap komprimiert ist oder nicht. In dieser Arbeit wird von unkomprimierten Bitmaps ausgegangen. Deshalb wird dieser Punkt nicht näher erläutert. Für unkomprimierte Bitmaps ist dieser Wert Null.

- `biSizeImage`

  gibt die Größe des dekomprimierten Bitamps an und wird nur sinnvoll eingesetzt, wenn ein komprimiertes Bitmap vorliegt. Bei unkomprimierten Bitmaps ist dieser Wert üblicherweise Null.

- `biXPelsPerMeter` und `biYPelsPerMeter`

  geben einen horizontalen und einen vertikalen Skalierungsfaktor für das Endgerät des Bitmaps an. Dieser Wert wird in Pixel pro Meter angegeben, wird aber von den meisten Anwendungen nicht verwendet und kann deshalb auch Null sein.

- `biClrUsed`

  gibt die Anzahl von benutzten Farben in der Farbtabelle an. Ist dieser Wert Null, werden alle Farben benutzt, die mit der in `biBitCount` angegebenen Farbtiefe erzeugt werden können.

  Allerdings wird ab einer Farbtiefe von 24 Bit keine explizite Farbtabelle mehr erstellt, sondern die Information über die Farbe steckt in den einzelnen Kanälen des Bildes selbst (siehe Seite 36). Die Farbtabelle eines Bitmaps mit 24 Bit Farbtiefe wäre bereits $2^{24} * 4 = 67108864$ also 64 MB groß.

- `biClrImportant`

  wird verwendet, um die Anzahl der notwendigen Farben anzugeben. Ist dieser Wert Null, so werden alle Farben benötigt.

Die Farbtabelle Wird eine Farbtabelle benötigt, enthält diese eine über den Wert `biClrUsed` bestimmte Anzahl von RGBQUAD Einträgen. Diese RGBQUAD Einträge enthalten 4 Byte, über die die Farbe dieses Eintrags bestimmt wird.

Das erste Byte enthält den Wert für den blauen Anteil. Byte Zwei repräsentiert den grünen Anteil, das dritte Byte den roten. Das vierte Byte ist reserviert und wird zur Zeit noch nicht benutzt. Es ist immer Null.

Die Bilddaten Nach einer eventuellen Farbtabelle kommen die eigentlichen Bildinformationen. Die Anordnung der Bilddaten hängt direkt von dem in `biBitCount` angegebenen Wert ab. Sie werden ohne Trennzeichen hintereinander geschrieben. Dabei ist zu beachten, daß ein Pixel in einem Schwarz–Weiß–Bild

nur ein Bit benötigt und somit acht Pixel in ein Byte passen, bei einer Farbtiefe von 24 Bit hingegen benötigt ein Pixel bereits drei Byte.

Bei einem heute üblichen bottom–up–Bitmap beschreibt der erste Eintrag das Pixel der unteren linken Ecke. Der nächste Eintrag beschreibt das Pixel rechts daneben. Sind die Pixel der ersten Zeile beschrieben, folgen die Pixel der darüber liegenden Zeile.

Eine Bedingung, die an die Bilddaten gestellt wird, ist, daß jede Bildzeile eine Anzahl von Byte enthalten muß, die durch vier teilbar ist. Enthält eine Zeile nicht genug Informationen, so werden die fehlenden Bits oder Bytes mit Nullen aufgefüllt.

Ist zum Beispiel ein Schwarz–Weiß–Bild 150 Pixel breit (`biWidth` = 150, `biBitCount` = 1), dann besitzt eine Zeile $150/8 = 18,75$ also 19 Byte. Das neunzehnte Byte wird dann mit Bits mit dem Wert Null aufgefüllt, das zwanzigste Byte mit dem Wert Null wird erzeugt.

4.4.2 Filtern des Ursprungsbildes

Um die Erkennung der Farben aus dem Bildinhalt zu vereinfachen, ist es sinnvoll, das Eingangsbild zu bearbeiten.

So sollten Inhomogenitäten in der Beleuchtung ebenso eliminiert werden wie auch Schwächen der Kamera.

Da als Zielbild ein Bild gewünscht ist, das nur noch die Farben Rot, Grün, Blau, Gelb, Cyan, Magenta sowie Schwarz und Weiß enthält, diese Farben aber nur entstehen, wenn die einzelnen Kanäle Minimum– oder Maximumwerte annehmen, ist es sinnvoll, die einzelnen Kanäle zu binarisieren. Binarisierung ist ein einfaches Segmentierungsverfahren.

Referenzbild–Division

Selbst bei einem sehr hohen Aufwand ist es schwierig, eine perfekt gleichmäßige Beleuchtung zu erreichen.

Heutige Webcams arbeiten zur Bildaufnahme mit günstigen CCD–Sensoren[2]. Diese zeigen eine ungleichmäßige Empfindlichkeit der einzelnen Rezeptoren. Außerdem enthalten diese Kameras eine Optik, die neben den typischen Schwächen günstiger Optiken auch durch Staub verschmutzt werden.

Diese Störungen beeinträchtigen die Bildqualität und erschweren die Separierung der Objekte von dem Hintergrund. Diese Störungen würden zu systematischen Fehlern in der nachfolgenden Bildauswertung führen.

[2] CCD⇔charge coupled devices (Ladungsgekoppelte Bauelemente)

Eine einfache aber sehr effektive Methode, um inhomogene Beleuchtung und Kameraschwächen zu beseitigen, ist die Referenzbild–Division.

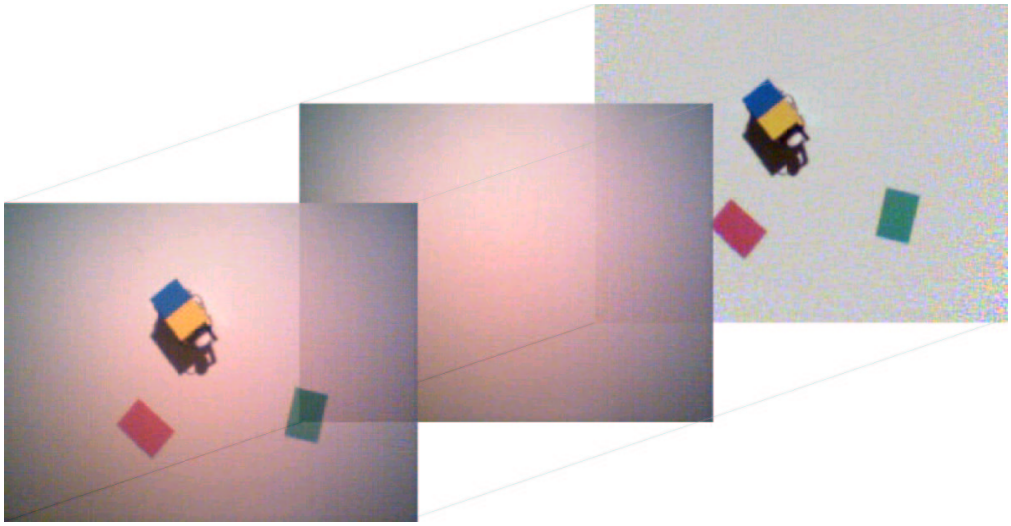

Abbildung 4.5: Prinzip der Referenzbild–Division

Hierbei wird als erstes ein Bild des Hintergrundes ohne Objekte aufgenommen.

Das zu analysierende Bild wird dann durch dieses Bild dividiert und anschließend mit einem Faktor multipliziert. Diese Division wird für alle N Spalten und M Zeilen, also für jedes einzelne Pixel durchgeführt.

$$Z_{n,m} = \frac{E_{n,m}}{R_{n,m}} * f \tag{4.1}$$

Der Faktor f ist notwendig, um das korrigierte Bild wieder in Integerzahlen darzustellen. Betrachtet man, wie im Fall dieser Arbeit, eine Szene die Licht absorbiert, liegt dieser Wert nahe dem Höchstwert für das Pixel.

In Abbildung 4.6 ist das Ergebnis dieser Division im Gesamtbild und den einzelnen Farbkanälen abgebildet.

Alternative Methoden zur Referenzbild–Division

Eine noch einfachere und schnellere Methode ist es, das aufgenommene Referenzbild von dem zu bearbeitenden Bild zu subtrahieren. Um es anzeigen zu können, wird von dem Ergebnis der Betrag gebildet. Das entstehende Bild ist invertiert. Um es in die richtige Form zurück zu transformieren, zieht man das Ergebnis von dem Höchstwert für das Pixel ab. Das Ergebnis ist ein von den Störungen befreites, farbrichtiges Bild. Allerdings hat dieses Verfahren den Nachteil, daß das

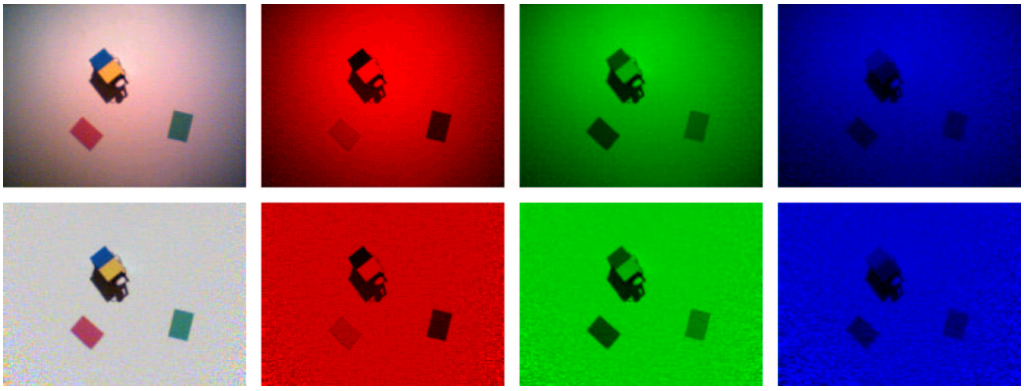

Abbildung 4.6: Erfolg der Referenzbild–Division

Ergebnisbild in den Bereichen in denen die Ausleuchtung besonders schlecht ist, sehr blaß wird. Die Referenzbild–Division führt hier zu einem deutlich besseren Ergebnis.

Eine weitere Methode ist die Zweipunkt–Kalibrierung. Diese Methode eignet sich besonders dann, wenn das Bild auch ohne Beleuchtung noch ein Restmuster (fixed–pattern–noise) enthält. In diesem Fall sind zwei Referenzbilder aufzunehmen. Eines wie bisher, ohne Objekte und mit Beleuchtung (R^m), das andere ohne Objekte und ohne Beleuchtung (R^o). Durch Anwendung dieser beiden Bilder in folgender Gleichung

$$Z_{n,m} = \frac{E_{n,m} - R^o_{n,m}}{R^m_{n,m} - R^o_{n,m}} * f \tag{4.2}$$

erhält man ein Bild, das frei von Inhomogenitäten ist. Der Faktor f ist hier aus den gleichen Gründen notwendig, wie auch bei der einfachen Referenzbild–Division.

Weitere Verfahren wie z.B. lineare und nicht lineare Grauwertskalierung, Äquidensiten, Hoch– und Tiefpaßfilter, radiometrische Kalibrierung und andere sind in der Literatur ausgiebig beschrieben, aber im Rahmen dieser Arbeit nicht maßgeblich.[3]

Segmentierung von Grauwertbildern

Der Begriff Grauwertbild mag in dem Zusammenhang mit Farbbildern als nicht richtig erscheinen. Da es sich bei Farbbildern aber um drei Farbkanäle handelt und jeder dieser drei Farbkanäle nur bestimmte Helligkeiten einer Farbe beinhaltet, ist

[3] Dem interessierten Leser sei die Lektüre von [Bässmann und Kreyss 1998; Haberäcker 1995; Jähne 1997; Kopp 1997] empfohlen.

ein solcher Farbkanal im Prinzip ein Grauwertbild. Der einzige Unterschied ist, daß ein Kanal nicht die verschiedenen Helligkeiten von Schwarz bis Weiß darstellt, sondern die zwischen Schwarz und Rot, Schwarz und Grün sowie Schwarz und Blau.

Alle RGB Farben lassen sich in einem Farbwürfel darstellen. Auf der X–Achse wird die rote, auf der Y–Achse die grüne und auf der Z–Achse die blaue Farbe aufgetragen. Die in dem Würfel eingezeichnete Diagonale besitzt für alle drei Farben den gleichen Wert. Sie repräsentiert Schwarz und Weiß, so wie auch alle möglichen Grauwerte. Diese Diagonale wird deshalb auch als Unbunt–Gerade bezeichnet.

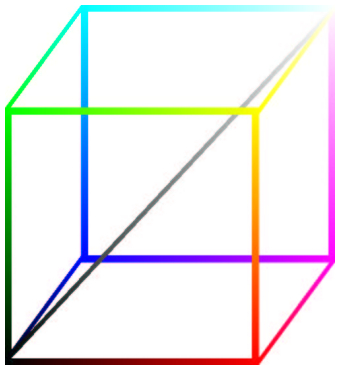

Abbildung 4.7: RGB Farbwürfel

Betrachtet man die Lage der acht Farben (siehe Anfang Abschnitt 4.4.2) in diesem Farbwürfel, so fällt auf, daß diese immer an den Ecken des Farbwürfels liegen.

Die acht Farben an den Endpunkten lassen sich somit folgendermaßen definieren:

Farbe	Rot	Grün	Blau
Schwarz	0	0	0
Weiß	*max*	*max*	*max*
Rot	*max*	0	0
Grün	0	*max*	0
Blau	0	0	*max*
Gelb	*max*	*max*	0
Cyan	0	*max*	*max*
Magenta	*max*	0	*max*

Dies legt nahe, die drei Farbkanäle zu binarisieren. Bei diesem Verfahren werden die einzelnen Pixel klassifiziert. Entweder ist ein Pixel Null oder es besitzt den Höchstwert, den dieses Pixel annehmen darf.

In einem ersten einfachen Ansatz wird diese Entscheidung über einen Schwellwert getroffen. Liegt der Wert des Pixel unter dem Schwellwert, so wird er auf Null gesetzt. Ist der Wert des Pixel größer oder gleich dem Schwellwert, wird dieses Pixel auf den Höchstwert gesetzt. Als Ergebnis kommt ein Bild zustande, das nur noch die oben genannten acht Farben enthält.

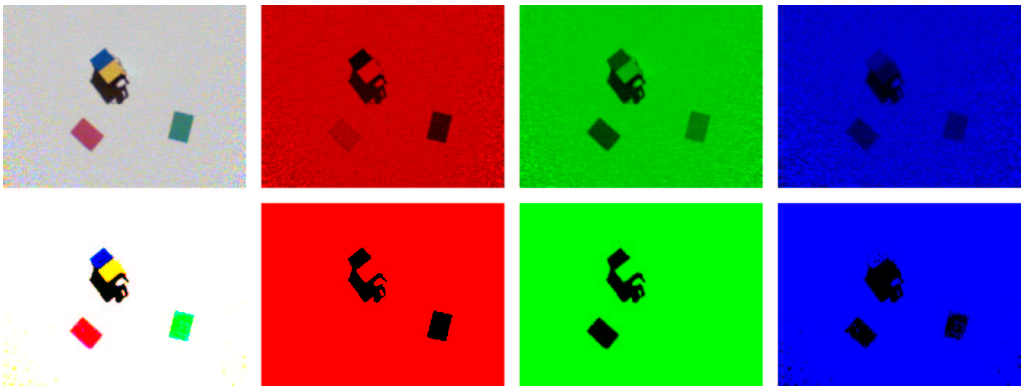

Abbildung 4.8: Erfolg der einfachen Binarisierung

Ein weiterer Ansatz bedient sich nicht eines einfachen Schwellwertes, sondern versieht diesen Schwellwert mit einer Hysterese. Hierbei wird nicht aufgrund des einzelnen Pixel entschieden, sondern es werden zwei bereits binarisierte, benachbarte Pixel für diese Entscheidung herangezogen. Dies könnten beispielsweise das binarisierte Nachbarpixel in der Zeile und das entsprechende der Spalte sein.

Da zu Beginn der Binarisierung natürlich noch keine binarisierten Pixel vorhanden sind, muß für dieses Verfahren die erste Spalte und die erste Zeile mit Werten für den Hintergrund beschrieben werden. Die erste Spalte und die erste Zeile brauchen deshalb bei der Binarisierung nicht bearbeitet werden.

Der Schwellwert wird nun unter Berücksichtigung der Nachbarpixel für jedes zu binarisierende Pixel neu berechnet.

Die Berechnung erfolgt nach folgender Vorschrift:

Schwelle

$c = c_0$,	falls	$Z_{-1} + S_{-1} = max$
$c = c_0 + x$,	falls	$Z_{-1} = S_{-1} = 0$
$c = c_0 - x$,	falls	$Z_{-1} = S_{-1} = max$

Hierbei ist c der für dieses Pixel zu benutzende Schwellwert, c_0 der generelle Schwellwert und x der Wert der Hysterese. Z_{-1} und S_{-1} sind die Werte des Zeilen– bzw. des Spaltennachbars. Diese können, da sie bereits binarisiert wurden, nur die Werte Null oder *max* besitzen.

Ist nun eines der beiden Nachbarpixel *max* und das andere Null, so befindet man sich höchstwahrscheinlich in dem Randbereich eines Objektes. In diesem Fall wird die unveränderte generelle Schwelle benutzt.

Sind beide Pixel Null, so ist das zu verarbeitende Pixel mit großer Wahrscheinlichkeit Teil eines Objektes.[4] Trifft dies zu, wird der Schwellwert um x erhöht, was zur Folge hat, daß dieses Pixel eher auf Null als auf *max* gesetzt wird, also eher zum Objekt gezählt wird als zum Hintergrund.

Im Falle das beide Pixel den Wert *max* besitzen, gehört das aktuelle Pixel mit größerer Wahrscheinlichkeit zum Hintergrund. Die Schwelle wird deshalb um x verringert, das Pixel kann den Schwellwert leichter überschreiten und wird eher auf *max* gesetzt, also zum Hintergrund gezählt.

Alternative Methoden zur Segmentierung

Die Binarisierung von Grauwertbildern ist eines der elementaren Segmentierungsverfahren. Allerdings lassen sich nicht alle Objekte einfach durch ihre Grauwerte (ggf. in den drei Farbkanälen) identifizieren. In diesen Fällen müssen zusätzliche Informationen herangezogen werden, um sinnvoll segmentieren zu können. In diesen Fällen spricht man von Segmentierung durch Klassifikation.

Ein weiteres elementares Segmentierungsverfahren, daß binarisierte Grauwertbilder liefert, verwendet eine lokale Glättung. Auch in diesem Verfahren wird mit einer Schwelle gearbeitet. Diese Schwelle wird ebenfalls für jedes Pixel neu berechnet. Allerdings werden hier nicht nur zwei Nachbarpixel herangezogen, sondern ein “lokales Fenster” um dieses Pixel herum, betrachtet.

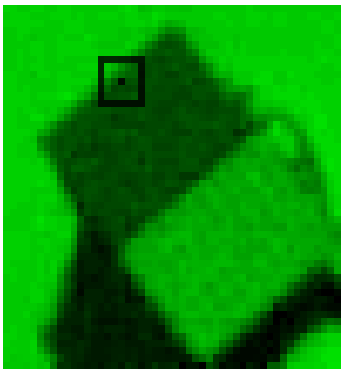

Abbildung 4.9: Binarisierung mit lokalem Fenster

Das Pixel, für das die Binarisierung durchgeführt werden soll, befindet sich in der Mitte des Fensters. Aus den Werten der umliegenden Pixel wird der Mittelwert

[4] Der Hintergrund ist weiß. Deshalb sind beim Hintergrund die RGB Anteile jeweils *max*.

bestimmt. Dieser Mittelwert (m) wird in die Berechnung der Schwelle miteinbezogen.

$$c = c_0 - w * m \tag{4.3}$$

c_0 ist hierbei die generelle Schwelle. w ist ein Gewichtsfaktor, über den die Schwelle eingestellt werden kann.

In Hintergrundbereichen existiert so eine niedrigere Schwelle, so daß dunkle Störstellen als Hintergrund erkannt werden. Auf der anderen Seite werden in Objektbereichen helle Störstellen als Objektpixel gehandhabt.

4.4.3 Merkmalsextraktion

Aus einem digitalen Bild lassen sich eine Vielzahl von Merkmalen der Objekte extrahieren. So ist es beispielsweise möglich, die "Rundheit", die Kompaktheit, den Umfang oder die Orientierung eines Objektes zu bestimmen. Komplexere Merkmale, die sich allerdings nicht so einfach extrahieren lassen, stellen die Klasse der Momente dar. Mit Hilfe dieser Momente können Objekte translations–, rotations– und größeninvariant beschrieben werden.

Im Rahmen dieser Arbeit werden nur zwei Merkmale benötigt. Dies sind die Größe des Objektes und die Lage des Schwerpunktes des Objektes. Auf diese beiden Größen wird im Folgenden eingegangen. Methoden für die zuvor genannten Merkmale werden in der Fachliteratur umfassend beschrieben.

Bevor man die Merkmale Größe und Schwerpunkt eines bereits segmentierten Bildes extrahieren kann, muß man feststellen, ob die einzelnen Pixel zum Hintergrund gehören oder zu einem Objekt. Gehören sie zu einem Objekt, muß festgestellt werden, zu welchem Objekt sie gehören. Dies wird Zusammenhangsanalyse genannt.

Zusammenhangsanalyse

Voraussetzung für eine Zusammenhangsanalyse ist ein bereits segmentiertes Bild. Um in einem RGB–Farbbild festzustellen, ob ein Pixel zu einem Objekt oder zum Hintergrund gehört, müssen alle drei Farbkanäle des Bildes bearbeitet werden. Die Ergebnisse der drei Bilder werden anschließend zu einem Gesamtergebnis kombiniert.

Alternativ zu der Betrachtung der drei einzelnen Farbkanäle kann man die Informationen dieser Kanäle vor der Betrachtung kombinieren. So ein Bild ist üblicherweise ein Grauwertbild, bei dem sowohl die einzelnen Objekte, als auch der Hintergrund durch einen Grauwert repräsentiert werden. Ein solches Bild wird

im allgemeinen Label–Bild genannt. Das in Abbildung 4.10 gezeigte Beispiel benutzt zur besseren Darstellung statt Grauwerten die vier Farben Rot, Grün, Blau und Weiß.

Ein verbreitetes Verfahren für die Zusammenhangsanalyse ist das sogenannte *Blob Coloring*. Bei diesem Verfahren wird neben dem Label–Bild ein so genanntes Marken–Bild benötigt. Dieses Marken–Bild hat die gleiche Größe wie das Label–Bild und wird zu Beginn der Analyse mit Null initialisiert. In ihm werden die Informationen für den Zusammenhang der Objekte gespeichert. Weiter wird ein Markenzähler benötigt, der zu Beginn der Analyse auf Null gesetzt wird.

Um das Label–Bild auf einen Zusammenhang untersuchen zu können, benötigt man zwei “Operatorfenster”. Diese Fenster werden durch einen Winkel gebildet. In Abbildung 4.10 sind diese Operatorfenster zu erkennen.

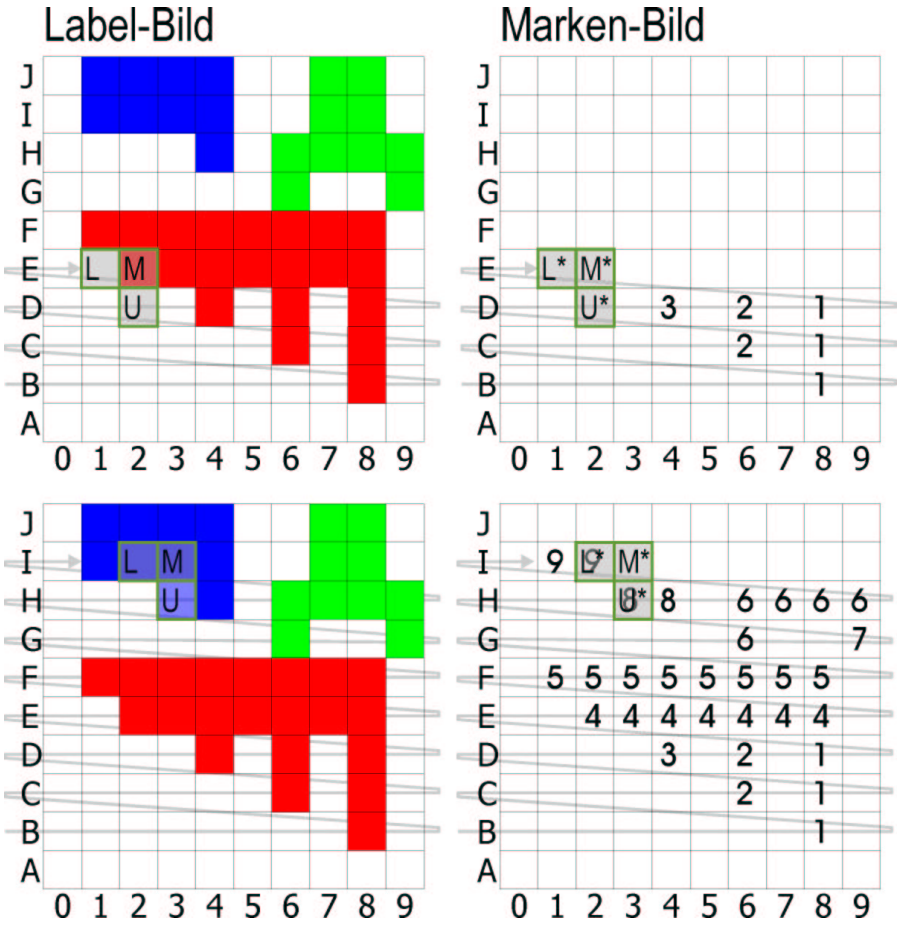

Abbildung 4.10: Zusammenhangsanalyse

Diese Fenster sind mit drei Buchstaben versehen. Der Buchstabe M bezeichnet

das aktuell zu bearbeitende Pixel (Mitte). L steht für den direkten linken Nachbarn (Links) und U für den direkten unteren Nachbarn (Unten).

Das hier beschriebene Verfahren ist ausgelegt für ein bottom–up–Bitmap. Wie bereits in 4.4.1 beschrieben, werden bottom–up–Bitmaps von unten links zeilenweise nach oben aufgebaut. Möchte man dieses Verfahren auf ein top–down–Bitmap anwenden, muß das Operatorfenster statt des direkten unteren Nachbarn den direkten oberen Nachbarn enthalten.

Das erste Operatorfenster arbeitet nur auf dem Label–Bild und wird durch einfache Buchstaben gekennzeichnet. Das zweite Operatorfenster enthält die gleichen Buchstaben, allerdings mit einem Stern versehen und arbeitet ausschließlich auf dem Marken–Bild.

Das Verfahren bearbeitet die Pixel einzeln und betrachtet für eine Zuordnung dieses Pixel, die im ersten Operatorfenster befindlichen Pixel des Label–Bildes sowie die im zweiten Operatorfenster befindlichen Marken des Marken–Bildes.

Dies hat zur Folge, daß die erste Zeile (wegen bottom–up–Bitmap also die unterste) und die erste Spalte (also die ganz links) der beiden Bilder nicht bearbeitet werden können. Diese Zeile bzw. Spalte des Label–Bildes wird mit den Werten für den Hintergrund initialisiert.

Das Verfahren beginnt dann mit dem zweiten Pixel der zweiten Zeile. Als erstes wird für das aktuelle Pixel (M) festgestellt, ob es zum Hintergrund gehört oder Teil eines Objektes ist. Gehört das Pixel zum Hintergrund, so wird mit dem nächsten Pixel fortgefahren. Ist dieses Pixel Teil eines Objektes, wird im Marken–Bild nach folgenden Regeln eine Marke gesetzt.

- $M \neq L$ und $M \neq U$

  Weder der linke (L) noch der untere (U) Nachbar entsprechen dem Wert des aktuellen Pixel (M). Diese Situation tritt beispielsweise dann ein, wenn der erste Teil des ersten Objektes erkannt wird. B8 in Abbildung 4.10 ist ein solcher Fall. Aber beispielsweise auch E2 entspricht dieser Forderung.

  In diesem Fall wird als erstes der Markenzähler inkrementiert. Der Wert des aktuellen Pixel des Marken–Bildes erhält den Wert des Markenzählers. Auf diese Weise wird eine neue Marke eingeführt.

- $M \neq L$ und $M = U$

  Der Wert des aktuellen Pixel (M) des Label–Bildes ist ungleich dem seines linken (L) Nachbarn, aber gleich dem unteren (U) Nachbarn. Eine entsprechende Situation liegt beispielsweise bei C8 in Abbildung 4.10 vor.

  Ist dies der Fall, so wird der aktuelle Wert des Marken–Bildes (M*) mit dem Wert der Marke des unteren Nachbarn gesetzt (U*). Der Markenzähler bleibt unbeeinflußt.

- M = L und M $\neq$ U

  Der Wert des linken Pixel (L) entspricht dem des aktuellen (M). Der untere (U) Nachbar hingegen entspricht nicht dem aktuellen Pixel (M). In der Abbildung 4.10 entspricht E3 dieser Situation.

  Hier wird dem aktuellen Wert (M*) des Marken–Bildes der Wert des linken (L*) Nachbarn gegeben. Auch hier wird der Markenzähler nicht verändert.

- M = L und M = U

  Sowohl der linke (L) als auch der untere (U) Nachbar entsprechen dem aktuellen Pixel (M) des Label–Bildes. In diesem Fall erhält der aktuelle Wert (M*) des Marken–Bildes den Wert des linken (L*) Nachbarn. Auch hier bleibt der Markenzähler unverändert.

  Allerdings bedarf dieser Fall einer etwas genaueren Betrachtung und es muß nicht nur das Label–Bild betrachtet werden, sondern auch das Marken–Bild.

  Zwei verschiedene Situationen sind möglich.

  Ein Beispiel für den ersten Fall ist J4 in Abbildung 4.10. In diesem Fall entspricht der linke (L*) Nachbar im Marken–Bild dem unteren (U*) Nachbarn. In diesem Fall wird die Bearbeitung einfach mit dem nächsten Pixel fortgesetzt.

  Den anderen Fall stellt I4 in Abbildung 4.10 dar. Hier unterscheidet sich der linke (L*) Nachbar des Marken–Bildes von dem unteren (U*) Nachbarn. Da aber im Label–Bild alle drei Felder des Operationsfensters gleich sind, gehört das aktuelle Pixel (M) ebenso wie der linke (L) und der untere (U) Nachbar zu dem selben Objekt.

  Daraus kann man entnehmen, daß der Wert des linken (L*) und des unteren (U*) Nachbarn des Marken–Bildes zu dem selben Objekt gehören, obwohl sie unterschiedliche Werte besitzen.

  Dieses Problem erfordert eine Äquivalenzliste. Sie protokolliert, welche Werte zu einem Objekt gehören. Die Situation E4 in Abbildung 4.10 sorgt zum Beispiel für einen Eintrag in dieser Äquivalenzliste. Dieser Eintrag zeigt, daß alle Einträge des Marken–Bildes mit dem Wert 4 zu dem selben Objekt gehören, wie die Einträge mit dem Wert 3.

Die Äquivalenzliste enthält nicht nur Einträge von Äquivalenzen von zwei unterschiedlichen Werten, sondern auch Einträge von Werten, die nur einfach vorkommen. Sobald der Markenzähler erhöht wird und somit eine neue Marke in dem Marken–Bild entsteht, wird diese Marke in die Äquivalenzliste eingetragen. Dieser Eintrag verweist also auf sich selbst. Diese Einträge sind nötig, um die

“echten” Äquivalenzen der Liste auflösen zu können. In Abbildung 4.11 sind die Schritte der Auflösung der Äquivalenzen dargestellt. Sie werden im Folgenden erklärt.

Abbildung 4.11: Auflösen der Äquivalenzliste

Die erste Äquivalenzliste enthält die Einträge, die bei der Zuammenhangsanalyse des Label–Bildes aus 4.10 entstehen würde. Als erstes wird die Äquivalenzliste sortiert. Existiert ein Eintrag in der Liste, der nicht auf sich selbst verweist, also eine “echte” Äquivalenz enthält, dann wird der Eintrag, der auf sich selbst verweist, durch den der “echten” Äquivalenz ersetzt. Das Ergebnis dieser Vorgehensweise ist in der zweiten Äquivalenzliste in Abbildung 4.11 gezeigt.

Anschließend werden die Äquivalenzen aufgelöst. Hierbei werden die Listeneinträge daraufhin untersucht, ob sie auf sich selbst verweisen oder nicht. Verweist ein Eintrag auf sich selbst, bleibt er unverändert. Verweist er allerdings auf einen anderen Eintrag, wird die Äquivalenz dieses Eintrags geprüft. Und zwar solange, bis der Eintrag auf sich selbst verweist. Dieser Wert wird dann in den zuerst untersuchten Listeneintrag eingesetzt.

Zur Verdeutlichung ein Beispiel, bezogen auf die Äquivalenzliste 2 der Abbildung 4.11.

Der erste Eintrag der Liste verweist auf den vierten Eintrag.
Der vierte Eintrag der Liste verweist auf den fünften Eintrag.
Der fünfte Eintrag der Liste verweist auf sich selbst.
Der erste Eintrag wird so geändert, daß er auf den fünften Eintrag verweist.

Das Ergebnis dieser Auflösung kann man in Äquivalenzliste 3 der Abbildung 4.11 sehen. Man kann dieser Liste bereits entnehmen, daß dieses Label–Bild

drei Objekte enthält. Die drei Gruppen verweisen entweder auf fünf, auf sieben oder auf neun.

Ersetzt man nun die einzelnen Einträge mit ihren jeweiligen Äquivalenzen, so erhält man ein bereinigtes Marken–Bild. Dieses Bild ist in Abbildung 4.12 dargestellt.

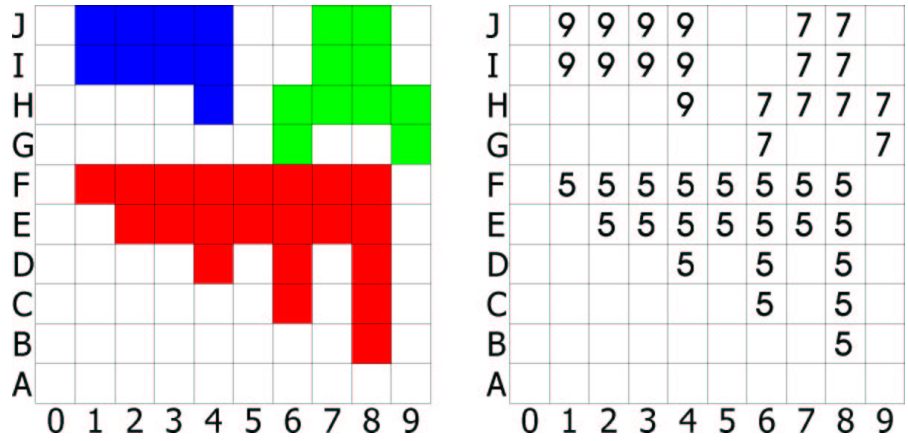

Abbildung 4.12: Ergebnis der Zusammenhangsanalyse

Diese Informationen reichen aus, um einige Merkmale aus dem Bild zu extrahieren. Verfahren für den Schwerpunkt und die Größe werden im nächsten Abschnitt beschrieben.

Bestimmen von Größe und Schwerpunkt

Sowohl die Größe als auch der Schwerpunkt lassen sich leicht aus dem Marken–Bild entnehmen. Zu diesem Zweck werden einige Variablen benötigt. Für jedes Objekt wird ein eigener Satz Variablen benötigt.

Dies läßt sich am einfachsten realisieren, indem man für diese Variablen ein eindimensionales Array der Größe des vorher verwendeten Markenzählers (*MZ*) schafft.

- $S_{[i]}$ $\Leftrightarrow$ Anzahl der Pixel des Objekts
- $SX_{[i]}$ $\Leftrightarrow$ Summe aller X–Koordinaten des Objekts
- $SY_{[i]}$ $\Leftrightarrow$ Summe aller Y–Koordinaten des Objekts

Da das Marken–Bild für jedes Objekt eine eigene Marke verwendet, kann diese Marke als Index des Arrays benutzt werden, so daß die Informationen automatisch in der entsprechenden Variable gespeichert werden. So werden umständliche Abfragen vermieden.

Für alle Einträge des Arrays die Null sind, gibt es offensichtlich keine Objekte im Marken–Bild.

Jedes einzelne Pixel des Marken–Bildes wird jetzt geprüft, ob es zum Hintergrund oder zu einem Objekt gehört. Dies geschieht durch eine einfache Prüfung auf Null. Gilt $Pixel_{n,m} = Null$, so gehört dieses Pixel zum Hintergrund.

Gilt dies nicht, wird $S_{[Pixel_{n,m}]}$ inkrementiert. Es wird also die Anzahl der Pixel, die zum Objekt $Pixel_{n,m}$ gehören, um eins erhöht. Zu $SX_{[Pixel_{n,m}]}$ wird der Wert der X–Achse addiert (also n), zu $SY_{[Pixel_{n,m}]}$ der Wert der Y–Achse (also m).

Ist dies für alle Pixel des Marken–Bildes geschehen, so enthält die Variable $S_{[i]}$ für alle i mit $0 < i \leq MZ$ die Anzahl der Pixel des Objektes mit der Marke i.

S[i]

i	
1	0
2	0
3	0
4	0
5	21
6	0
7	10
8	0
9	10

SX[i]

i	
1	0
2	0
3	0
4	0
5	111
6	0
7	75
8	0
9	28

SY[i]

i	
1	0
2	0
3	0
4	0
5	82
6	0
7	74
8	0
9	82

Abbildung 4.13: Merkmale des Beispiels

Dividiert man für alle i mit $0 < i \leq MZ$ die Werte $SX_{[i]}$ durch $S_{[i]}$, erhält man die X–Komponente des Schwerpunktes des Objektes mit der Marke i. Auf die gleiche Weise erhält man durch Division von $SY_{[i]}$ durch $S_{[i]}$ die Y–Komponente des Schwerpunktes.

Betrachtet man das Beispiel aus Abbildung 4.10, so erhält man das Ergebnis, das in Abbildung 4.13 dargestellt ist. Die Zeilen kennzeichnenden Buchstaben sind hier durch Zahlen von A = 0 bis J = 9 ersetzt worden.

Berechnet man nun die Schwerpunkte nach dem vorgestellten Verfahren, so erhält man für das im Label–Bild rote Objekt die Koordinaten $(5.3, 3.9)$, für das grüne Objekt $(7.5, 7.4)$ und schließlich für das blaue Objekt $(2.8, 8.2)$. Dieses Ergebnis ist in Abbildung 4.14 dargestellt.

Um diese Darstellung richtig interpretieren zu können, muß man beachten, das die Gitterlinien zwischen den Werten liegen. Sie kennzeichnen also nicht die Werte $\{0, 1, 2, \ldots\}$ sondern $\{0.5, 1.5, 2.5, \ldots\}$.

Alternative Methoden

Die in den letzten Abschnitten gezeigten Methoden sind sehr einfach. Sie sind für die Bearbeitung einfacher Bitmaps gedacht.

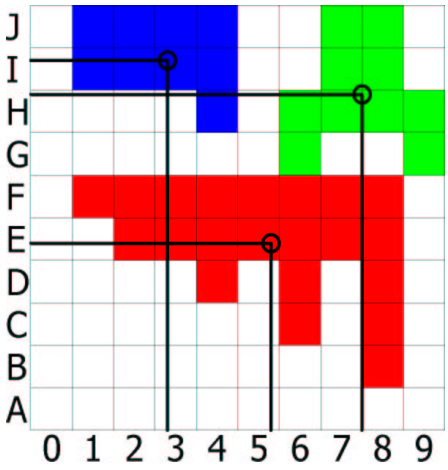

Abbildung 4.14: Lage der Schwerpunkte des Beispiels

Es gibt eine Vielzahl an Alternativen, die allerdings nicht für die Bearbeitung von Bitmaps in der Matrix–Form gedacht sind, sondern die Daten in einer spezielleren Form wie zum Beispiel des Lauflängen–Codes erwarten.

Diese Verfahren hier zu erläutern, würde voraussetzen, daß zunächst der Lauflängen–Code und dessen Entstehung aus einem Bitmap erklärt werden würde. Da das Ergebnis aber auf ähnlich einfache Weise entsteht, wird auf diese Erklärung an dieser Stelle verzichtet und auf die Fachliteratur wie zum Beispiel [Bässmann und Kreyss 1998; Haberäcker 1995; Jähne 1997] verwiesen.

4.5 Konklusion

In Abschnitt 4.4 wurde das jeweils ausgewählte Verfahren für die benötigte Bildverarbeitung vorgestellt und Alternativen zu diesem Verfahren aufgezeigt.

Diese Verfahren sind in sich geschlossen und werden nacheinander abgearbeitet. Dies führt zu einer Pipe. Die einzelnen Verarbeitungsschritte lassen sich beliebig durch ihre alternativen Methoden ersetzen.

Das Logitech SDK bietet die Möglichkeit eine Nachricht abzusetzen, sobald ein neues Bild der Kamera eingegangen ist. Diese Möglichkeit wird genutzt, um die Bildverarbeitung auf dem DIB des Logitech SDK zu starten.

Zuvor muß allerdings eine Bedingung der Bildverarbeitung erfüllt sein. Es muß ein Referenzbild erfaßt worden sein, mit dem Inhomogenitäten und Kameraschwächen nach dem vorgestellten Prinzip der Referenzbild–Division (Abschnitt 4.4.2) beseitigt werden können.

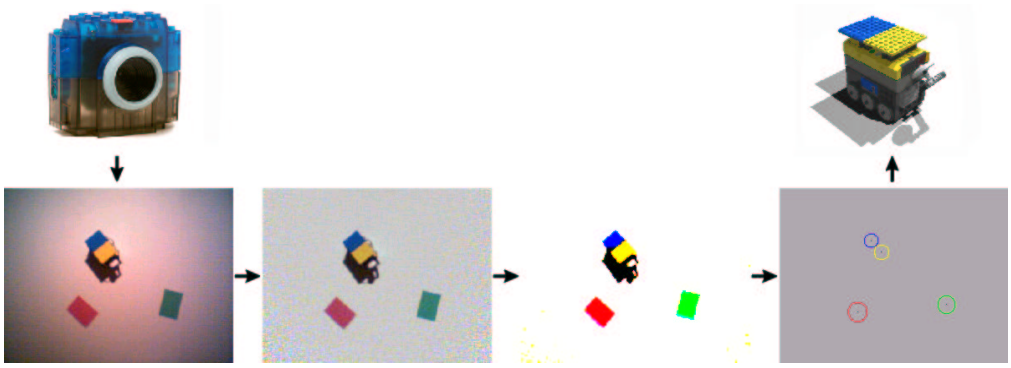

Abbildung 4.15: Reihenfolge der Verarbeitung

Das bereinigte Bild kann dann durch Binarisierung segmentiert werden, um anschließend die gewünschten Merkmale zu extrahieren (Abschnitt 4.4.2 und 4.4.3).

Nachdem die Merkmale der größten Flächen der jeweils sechs verschiedenen Farben aussortiert wurden, können diese durch die Verwendung des LEGO Spirit Control an den RCX gesendet werden.

Diese Gegebenheiten ermöglichen die Entwicklung einer Software, die der LEGO Vision Command Kamera und dem LEGO RCX Fähigkeiten verleihen, die denen der eingangs erwähnten Cognachrome Karte von newton labs ähnlich sind.

Kapitel 5

Design und Realisierung

Bei der Erstellung von Windows–Programmen unterscheidet man grundsätzlich zwischen dialogbasierten und dokumentenbasierten Anwendungen. Eine Textverarbeitung beispielsweise arbeitet auf Text–Dokumenten, während ein Programm zum Brennen von CDs durch einen Dialog realisiert ist.

Die im Rahmen dieser Diplomarbeit entwickelte Software besteht im wesentlichen aus einem Dialog, über den man die Schritte der Bildverarbeitung beobachten und deren Parameter anpassen kann. Dieser Dialog dient als grafische Benutzer–Schnittstelle des Programmes. "Hinter" diesem Dialog besteht das Programm aus einem Klassen–Modell. Dieses Modell und der Dialog werden in diesem Kapitel erklärt. Die Erklärung des Klassen–Modelles beschränkt sich allerdings auf die für das Lösen der Aufgabe relevanten Methoden.

5.1 Grafische Benutzer–Schnittstelle

Die grafische Benutzer–Schnittstelle[1] dient als einzige Möglichkeit des Benutzers zur Interaktion. Microsoft Visual C++ stellt eine Vielzahl an grafischen Elementen zur Erstellung solcher Schnittstellen zur Verfügung. Mit Hilfe dieser Elemente ist der in Abbildung 5.1 abgebildete Dialog erstellt worden. Im Hinblick auf eine eventuelle Benutzung der Software durch Mitglieder der internationalen LUGNET Gemeinde, ist dieser Dialog in englischer Sprache verfaßt worden.

Die wichtigsten Elemente des Dialoges sind die sechs Bilder. Durch sie kann der Fortschritt der Bildverarbeitung nachvollzogen werden.

In der Ecke oben links wurde das Benutzer–Interface des VideoPortals plaziert. In ihm wird das Echtzeitbild dargestellt. Außerdem kann man über dieses Interface die Kamera betreffende Einstellungen vornehmen.

[1] üblicherweise als GUI (Graphical User Interface) bezeichnet

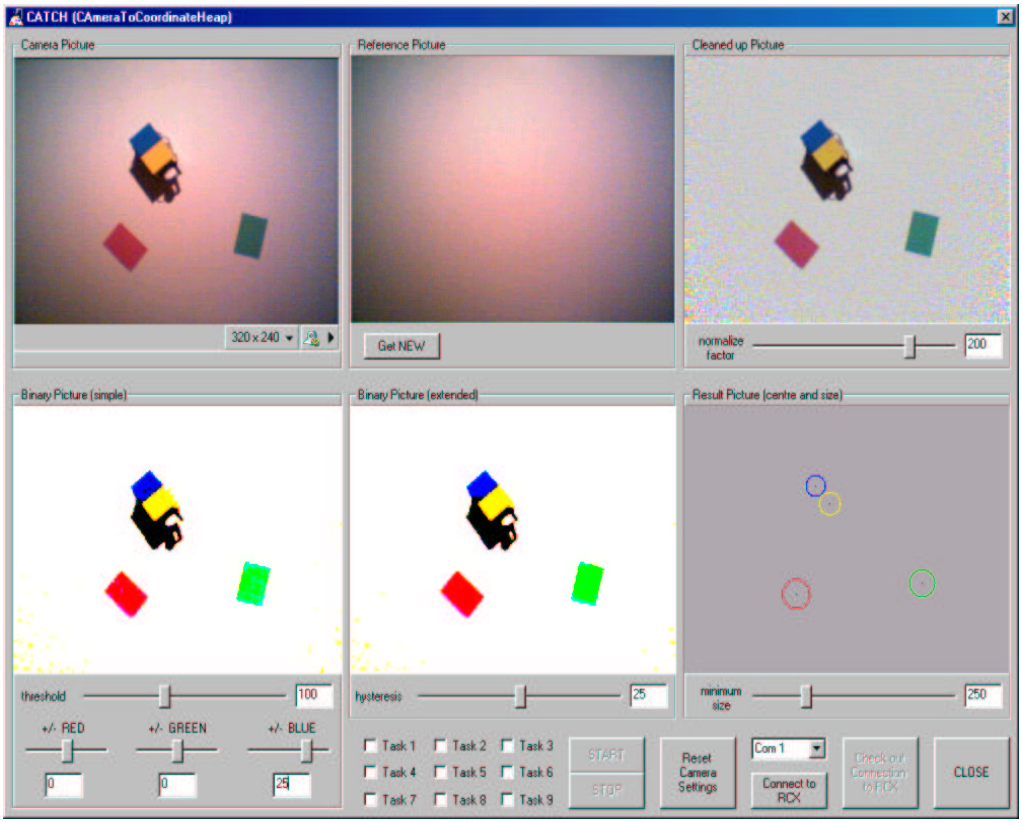

Abbildung 5.1: Das Programm CATCH (CAmera To Coordinate Heap)

Rechts neben dem Interface des VideoPortals wurde das Referenzbild angeordnet, das für die Referezbild–Division benötigt wird. Der Knopf unterhalb des Referenzbildes ermöglicht es, ein neues Referenzbild aufzunehmen.

Das Bild rechts oben stellt das mittels Referenzbild–Division optimierte Bild dar. Der Schieberegler mit der Bezeichnung “normalize factor” entspricht dem Faktor f in Gleichung 4.1 auf Seite 42.

Unten links wurde das Ergebnisbild der einfachen Binarisierung angebracht. Der zu diesem Bild gehörende Schieberegler “threshold” entspricht dem Schwellwert des in Abschnitt 4.4.2 vorgestellten einfachen Verfahrens zur Binarisierung von Grauwertbildern. Die drei unter ihm dargestellten Schieberegler ermöglichen es, die eingestellte Schwelle individuell, für jeden einzelnen der drei Farbkanäle, anzupassen.

In dem Bild in der Mitte unten wird ein erweitert binarisiertes Bild dargestellt. Über den Schieberegler “hysteresis” kann die Größe der Hysterese des erweiterten Verfahrens der Binarisierung eingestellt werden. Die notwendigen Schwellen entsprechen denen der einfachen Binarisierung. Der einzige neue Parameter ist

die Hysterese. Es mag überflüssig erscheinen beide Verfahren der Binarisierung einzusetzen, allerdings hat sich gezeigt, daß sich der Schwellwert am besten mit dem Verfahren der einfachen Binarisierung einstellen läßt.

Das Ergebnis der Merkmalsextraktion ist schließlich in dem Bild unten rechts dargestellt. Hierbei entspricht der Mittelpunkt des Kreises den Koordinaten des Schwerpunktes und die Größe des Kreises ist ein Maß für die Größe der Fläche. Mit Hilfe des Schiebereglers "minimum size" kann eine Mindestgröße der Farbfläche eingestellt werden, so daß Farbflächen die erkannt werden sollen, eine Mindestanzahl Pixel haben müssen.

Die Schaltflächen unterhalb der genannten Bilder sind in erster Linie für die Kommunikation mit dem RCX zuständig. Über sie läßt sich, falls beim Starten des Programmes keine Verbindung zum Infrarotsender und/oder zum RCX hergestellt worden ist, diese Verbindung herstellen. Die Verbindung zum RCX läßt sich prüfen oder abbauen. Es lassen sich einzelne oder mehrerer Tasks auf dem RCX starten oder stoppen. Eine Schaltfläche dient zum Rekalibrieren der Kameraeinstellungen, eine weitere zum Beenden des Programmes.

5.2 Infrarotkommunikation

Für die Kommunikation zwischen dem RCX und der Software auf dem PC wird das Spirit Control von LEGO eingesetzt.

Die Kommandos des Spirit Controls werden benutzt, um die von der Software bestimmten Positionen an den RCX zu senden.

Um die Möglichkeit offen zu halten, die Kommunikation über eine andere Schnittstelle[2] als Spirit Control zu verwirklichen, wurden die benutzten Methoden des Spirit Controls "gewrapped".

Dies bedeutet, daß eine neue Klasse (LegoComm) geschaffen wurde, in der die Funktionalität des Spirit Controls abgebildet wird. Nur in dieser Klasse sind die Möglichkeiten des Spirit Controls bekannt. Beim Senden an den RCX wird ausschließlich auf diese neue Klasse zugegriffen.

So ist es möglich, die Kommunikation zu einem späteren Zeitpunkt auf ein anderes Verfahren umzustellen. Es muß lediglich die Klasse LegoComm angepaßt werden.

5.2.1 Methoden der Klasse LegoComm

Die Klasse LegoComm besitzt folgende öffentliche Funktionen:

[2]Bei LegOS gibt es bereits jetzt rudimentäre Netzwerkfähigkeiten. Diese werden LNP (LEGO Network Protocol) genannt.

- `void SetComm(CSpirit)`

  Initialisiert das Spirit Control für die Benutzung mit dem RCX und bindet das Spirit Control mit Hilfe des Funktionsparameters an diese Klasse.

- `int InitComm(int)`

  Initialisiert den als Funktionsparameter übergebenen seriellen Port für die Benutzung durch das Spirit Control. Es wird versucht, eine Verbindung zum RCX herzustellen. Kann diese Verbindung hergestellt werden, wird ein Klang auf dem RCX ausgegeben und eine Eins zurückgegeben. Kommt keine Verbindung zustande, wird der Grund hierfür ausgegeben und eine Null zurückgegeben.

- `int CheckPort(int)`

  Überprüft die Verfügbarkeit des als Funktionsparameter übergebenen seriellen Port und das Vorhandensein eines Infrarotsenders an dieser Schnittstelle. Konnte dieser Infrarotsender gefunden werden, wird eine Eins zurückgegeben, andernfalls eine Null. In jedem Fall wird die Verbindung abgebaut und der serielle Port wieder geschlossen.

- `int CloseComm()`

  Stoppt alle Tasks auf dem RCX, baut die Verbindung zum RCX ab und gibt die serielle Schnittstelle frei.

- `void CheckConnection()`

  Setzt ein Kommando zum Spielen eines Klangs ab. Ist der RCX in Reichweite und die Infrarotverbindung steht, spielt der RCX einen Systemklang.

- `int StartTasks(USHORT)`

  Startet einen oder mehrere Tasks auf dem RCX. Der Funktionsparameter enthält bitweise die Nummern der bis zu neun zu startenden Tasks.

- `int StopTasks(USHORT)`

  Beendet einen oder mehrere Tasks auf dem RCX. Der Funktionsparameter enthält bitweise die Nummern der bis zu neun zu beendenden Tasks.

- `void SendData(short[13])`

  Fragt die Synchronisations–Variable des RCX ab. Ist ein Senden erlaubt, werden die ermittelten Daten aus der Bildverarbeitung an den RCX gesendet und die Synchronisations–Variable wird zurückgesetzt.

5.2.2 Synchronisation

Wie in Abschnitt 2.3.1 beschrieben, gibt es auf dem RCX zweiunddreißig persistente Variablen. Dies sind fest adressierte, vorzeichenbehaftete 16 bit Integer und werden von dem auf dem RCX laufenden Programm der Reihe nach vergeben. Diese Besonderheit macht sich die Kamerasoftware beim Speichern der relevanten Größen zunutze.

Eine Notwendigkeit des Programmes, das auf dem RCX läuft ist, daß die ersten dreizehn Variablen eine bestimmte Datenstruktur aufweisen.

In ihnen werden nacheinander die X– und Y–Koordinaten der sechs Farben in der Reihenfolge Blau, Grün, Cyan, Rot, Magenta und Gelb gespeichert. Die dreizehnte Variable dient zur Synchronisation der Kamerasoftware mit der RCX Software.

Diese Variable ermöglicht es, das Senden der Daten zu steuern. So sendet die Kamerasoftware nicht kontinuierlich Daten, sondern nur auf Anforderung. Diese Synchronisation ist notwendig, da das Senden der Daten via Spirit Control bis zu eineinhalb Sekunden dauern kann. Fährt der mobile Roboter in dieser Zeit, entsprechen die empfangenen Daten nicht mehr der aktuellen Position. Mit Hilfe der Synchronisations–Variablen hat man die Möglichkeit, den Roboter zu stoppen, die Daten anzufordern und erst nach Ende der Übertragung von der Bildverarbeitungssoftware den Weg (neu berechnet) fortzusetzen. Diese Anforderung obliegt dem RCX.

Die oben vorgestellte Funktion SendData(short[13]) liest diese Variable vom RCX. Ist diese Variable Null, findet kein Senden an den RCX statt. Ist sie ungleich Null, sendet die PC–Software die ermittelten Daten an den RCX, um anschließend die Synchronisations–Variable auf dem RCX auf Null zu setzen. Dies erfordert ein Setzen der Variablen vom RCX aus, was der oben erwähnten Anforderung entspricht.

Die folgende Funktion stammt aus dem Programm, das auf dem LEGO Roboter des Szenarios verwendet wird. Dieses Programm kann in Anhang B nachgelesen werden.

```
void cameracontrol() {
   Wait (wait_time); // Kamerabild aktualisieren lassen

   blau_x = 0;       // Alle Werte auf Null, damit Werte
   ...               // auf alle Fälle neu sind !
   ...
   gelb_y = 0 ;
   sync = 1;         // Kontrolle an Kamerasoftware übergeben

   while (sync);     // nichts tun solange Kamerasoftware die Kontrolle hat !
}
```

Diese Funktion wird auf dem RCX jedesmal aufgerufen, wenn neue Daten von der Kamerasoftware angefordert werden.

Zuerst werden die zwölf Variablen, die die Koordinaten aufnehmen, auf Null gesetzt. Dies garantiert, daß die anschließend in diesen Variablen gespeicherten Werte aktuelle Werte sind. Die hier verwendete Variable `sync` entspricht der Synchronisations–Variablen. Der RCX setzt sie auf Eins und wartet darauf, daß die Kamerasoftware diese Variable zurücksetzt.

5.3 Kamerakommunikation

Es war vorgesehen auch die Methoden des Logitech VideoPortales so zu kapseln, wie es mit den Methoden des Spirit Control geschehen ist. Leider muß die VideoPortal Komponente direkt an den Dialog der Anwendung gebunden werden und läßt sich nicht über einen Zeiger auf die Instanz ansprechen, was ein wrappen der Methoden verhindert. Die in der Software verwendete Instanz des VideoPortales heißt `m_c_Video` . Auf diese Instanz wird sich in diesem Abschnitt bezogen.

Bevor man die Funktionalität des VideoPortales nutzen kann, muß es beim System angemeldet und initialisiert werden. Dies geschieht über die Methode `m_c_Video.PrepareControl ("CATCH", sReg,0)` . Diese Methode bereitet das VideoPortal für die Anwendung vor und trägt den unter `sReg` angegebenen Schlüssel in der Registrierung von Windows ein.

Nach der Initialisierung des VideoPortales wird eine Verbindung zu einer Kamera hergestellt. Hierfür wird die Methode `m_c_Video.ConnectCamera2()` verwendet. Diese Methode verwendet die zuletzt benutzte Kamera oder falls noch keine Verbindung zu einer Kamera bestanden hat, die erste verfügbare.

Ist eine Verbindung hergestellt, so kann mit der Methode `m_c_Video.StartVideoHook(0)` der Benachrichtigungsmechanismus für Video–Streaming des VideoPortales gestartet werden.

Das Logitech SDK besitzt die Möglichkeit, die Software über eintretende Ereignisse des VideoPortales zu benachrichtigten. Tritt ein bestimmtes Ereignis ein, so wird die zur Klasse des Dialoges gehörende Methode `CCATCHDlg::OnPortalNotificationVideoportal1(long lMsg, long lParam1, long lParam2, long lParam3)` aufgerufen. In dieser Methode werden eine Vielzahl von Ereignissen verarbeitet. Diese werden anhand des ersten Parameters `lMsg` identifiziert. Die Verwendung der folgenden Parameter (`lParam1, lParam2, lParam3`) hängt davon ab, welches Ereignis eingetreten ist.

Die eintretenden Ereignisse, die auf diese Weise erkannt werden, werden durch folgende Nachrichten bekanntgegeben:

- `NOTIFICATIONMSG_MOTION`

Wird empfangen, wenn die in der Logitech SDK enthaltene Bewegungserkennung benutzt wird. Der Parameter `LParam1` enthält in diesem Fall die Menge der erkannten Bewegung in Prozent.

- `NOTIFICATIONMSG_MOVIERECORDERROR`

  Informiert die Software über das Auftreten eines Fehlers während der Aufzeichnung eines Videos vom Logitech SDK. Der Parameter `LParam1` enthält hier die Information, um welchen Fehler es sich handelt.

- `NOTIFICATIONMSG_CAMERADETACHED`

  Zeigt das Entfernen der Kamera vom USB–Port an.

- `NOTIFICATIONMSG_CAMERAREATTACHED`

  Meldet das Anschließen einer Kamera an den USB–Port.

- `NOTIFICATIONMSG_IMAGESIZECHANGE`

  Gibt bekannt, daß über das Benutzerinterface des VideoPortales die Auflösung der Kamera geändert wurde. Die Parameter `LParam1` und `LParam2` enthalten in diesem Fall die Breite und die Höhe der neuen Auflösung.

- `NOTIFICATIONMSG_CAMERAPRECHANGE`

  Diese Nachricht kann nicht benutzt werden.

- `NOTIFICATIONMSG_CAMERACHANGEFAILED`

  Meldet den Fehlversuch zu einer anderen Kamera umzuschalten. Der Index der Kamera wird in `LParam1` übergeben.

- `NOTIFICATIONMSG_POSTCAMERACHANGED`

  Zeigt das Umschalten zu einer anderen Kamera an. Der Parameter `LParam1` enthält den Index der neuen Kamera.

- `NOTIFICATIONMSG_CAMERBUTTONCLICKED`

  Ist die angeschlossene Kamera mit einem Taster versehen und dieser wird gedrückt, wird dies durch diese Nachricht angezeigt.

- `NOTIFICATIONMSG_VIDEOHOOK`

  Wird empfangen, wenn das Video–Streaming des VideoPortals benutzt wird. Jedes Bild, das die Kamera vollständig aufgenommen hat, wird so an die Software gemeldet. In Parameter `lParam1` wird auf die Struktur des BITMAPINFOHEADERS verwiesen. Parameter `lParam2` verweist auf die Bilddaten des zuletzt aufgenommenen Kamerabildes. Der Parameter `lParam3`

enthält einen Systemzeit–Stempel des Windows Systemes. In ihm ist die Entstehungszeit des aktuellen Bildes, bezogen auf das Starten von Windows, in Millisekunden enthalten.

- `NOTIFICATIONMSG_SETTINGDLGCLOSED`

  Diese Nachricht kann nicht benutzt werden.

- `NOTIFICATIONMSG_QUERYPRECAMERAMODIFICATION`

  Diese Nachricht kann nicht benutzt werden.

- `NOTIFICATIONMSG_MOVIESIZE`

  Zeichnet das Logitech SDK ein Video auf, wird über den Parameter `LParam3` die Größe des bereits aufgezeichneten Videos bekanntgegeben.

Die Benachrichtigung `NOTIFICATIONMSG_VIDEOHOOK` macht sich die Software für die Bildverarbeitung zunutze.

Tritt dieses Ereignis ein, so wird aufgrund der in den Parametern `LParam1` und `LParam2` übergebenen Bilddaten die Bildverarbeitung in einem neuen Task gestartet. Zuvor wird jedoch geprüft, ob noch ein Task für die Bildverarbeitung läuft. Ist dies der Fall, wird auf das Eintreffen der nächsten Bilddaten und somit auf das nächste Ereignis `NOTIFICATIONMSG_VIDEOHOOK` gewartet. Erst wenn der laufende Task abgeschlossen ist, wird ein neuer Task für die Bildverarbeitung gestartet.

Der Ablauf des Tasks wird am Ende des folgenden Absatzes beschrieben.

5.4 Bildverarbeitung

Bis auf das VideoPortal entsprechen die dargestellten Bilder des Dialoges Device–Independent–Bitmaps. Für die Software ist deshalb eine Klasse LegoDIB entwickelt worden, durch die diese Bilder repräsentiert werden. Diese Klasse nimmt sowohl den BITMAPINFOHEADER als auch die Bilddaten auf. Außerdem sind für diese Klasse diverse Methoden implementiert worden. Die wichtigsten dieser Methoden werden im Folgenden erläutert.

- `void Binary(int, int, int, int, CLegoDIB*)`

  Binarisiert das im fünften Parameter übergebene Bild nach dem auf Seite 45 vorgestellten Verfahren mit Schwelle und Hysterese. Die ersten drei Parameter enthalten die Schwellen für Rot, Grün und Blau. Der vierte Parameter enthält den Wert für die Hysterese. Das Ergebnis wird in der aufrufenden Instanz gespeichert.

- `void BinarySimple(int, int, int, CLegoDIB*)`

  Binarisiert das im vierten Parameter übergebene Bild nach dem auf Seite 45 vorgestellten Verfahren mit einfacher Schwelle. In den ersten drei Parametern werden die Schwellen für Rot, Grün und Blau übergeben. Das Ergebnis wird in der aufrufenden Instanz gespeichert.

- `void DividePic(const int,const CLegoDIB*, const CLegoDIB*)`

  Führt das auf Seite 41 vorgestellte Verfahren der Referenzbild–Division durch. Der erste Parameter enthält den notwendigen Multiplikator. Im zweiten Parameter wird das Dividendbild, im dritten Parameter das Divisorbild übergeben. Das Ergebnis der Division wird in der aufrufenden Instanz gespeichert.

- `void ClearPic(BYTE)`

  Initialisiert die aufrufende Instanz mit einem Grauwert, der im ersten Parameter übergeben wird. Die Werte können zwischen 0 (für Schwarz) und 255 (für Weiß) liegen.

- `long NoOfPixel()`

  Berechnet aus der Breite und der Höhe, die im BITMAPINFOHEADER gespeichert sind, die Anzahl der Pixel der aufrufenden Instanz und gibt diesen Wert zurück.

- `long GetWidth()`

  Gibt den Wert für die Breite der aufrufenden Instanz zurück.

- `long GetHeight()`

  Gibt den Wert für die Höhe der aufrufenden Instanz zurück.

- `BYTE GetColorOfPixel(long)`

  Gibt die Farbe eines einzelnen Pixel in codierter Form zurück. Der Parameter enthält hierbei den Index des Pixel, bezogen auf den Anfang der Bilddaten.

  Die Farbe ergibt sich als UND Verknüpfung aus 0x01 für Blau, 0x02 für Grün und 0x04 für Rot. Daraus ergeben sich die folgenden acht Möglichkeiten:

  - 0x00 $\Rightarrow$ Schwarz
  - 0x01 $\Rightarrow$ Blau
  - 0x02 $\Rightarrow$ Grün

  - 0x03 ⇒ Cyan
  - 0x04 ⇒ Rot
  - 0x05 ⇒ Magenta
  - 0x06 ⇒ Gelb
  - 0x07 ⇒ Weiß

- `void SetColorOfPixel(long, BYTE)`

  Setzt die Farbe eines einzelnen Pixel der aufrufenden Instanz. Hierbei wird die gleiche Codierung benutzt, wie sie auch die eben vorgestellte Methode `BYTE GetColorOfPixel(long)` verwendet. Der erste Parameter enthält hier ebenfalls den Index bezogen auf den Anfang der Bilddaten. Im zweiten Parameter wird der Farbwert übergeben.

Für die Bildverarbeitung selbst wurde eine eigene Klasse LegoIP (Image-Processing) geschaffen. Diese Klasse übernimmt die eigentliche Bildverarbeitung. Im Folgenden werden die bildverarbeitenden Methoden vorgestellt, die für jedes Bild in einem separaten Task ausgeführt werden. Die Bildverarbeitung arbeitet hierbei auf Objekten der Klasse LegoDIB. Die Software besitzt fünf öffentliche Objekte dieser Klasse. Diese entsprechen den Bildern in der grafischen Benutzerschnittstelle. Sie tragen die entsprechende Namen `m_c_LegoDIB_ref_pic`, `m_c_LegoDIB_cleaned_pic`, `m_c_LegoDIB_binary_simple_pic`, `m_c_LegoDIB_binary_ext_pic` und `m_c_LegoDIB_result_pic`. Zwei weitere dieser Objekte sind privat und dienen als Arbeits–Bitmap (`m_LegoDIB_work_pic`) und zur Aufbereitung der ausgemessenen Farbflächen (`m_LegoDIB_blobs_pic`). Auf diese Objekte wird in der Erklärung der Methoden der Klasse LegoIP Bezug genommen. Die wichtigsten öffentlichen Methoden der Klasse LegoIP sind:

- `void CleanUp()`

  Ruft von der Instanz `m_c_LegoDIB_cleaned_pic` aus die Methode `DividePic` der Klasse LegoDIB auf. Als Parameter werden der im Programm eingestellte Wert für “normalize factor” sowie das Arbeits–Bitmap `m_c_LegoDIB_work_pic` als Dividend und `m_c_LegoDIB_ref_pic` als Divisor übergeben. In `m_c_LegoDIB_cleaned_pic` liegt also anschließend das Ergebnis der Referenzbild–Division vor.

- `void BinarySimple()`

  Benutzt ebenfalls eine Methode der Klasse LegoDIB. Die Methode `BinarySimple` wird von der Instanz `m_c_LegoDIB_binary_simple_pic` aufgerufen. Die benötigten Parameter werden aus der grafischen Benutzerschnittstelle entnommen.

- `void BinaryExtended()`

  Entspricht weitestgehend der eben vorgestellten Methode. Allerdings wird hier von der Instanz `m_c_LegoDIB_binary_ext_pic` aus die LegoDIB–Methode `Binary` aufgerufen. Der fünfte Parameter für die Hysterese wird ebenfalls aus der grafischen Benutzerschnittstelle entnommen.

- `void CalculateMarks()`

  Berechnet die Marken und die Äquivalenzliste nach dem auf Seite 47 vorgestellten Verfahren der Zusammenhangsanalyse. Als Label–Bild dient hierbei das LegoDIB Objekt `m_c_LegoDIB_binary_ext_pic`. Nachdem alle Marken berechnet worden sind, wird die eigentliche Zusammenhangsanalyse durchgeführt und die Marken mit Hilfe der Äquivalenzliste aufgelöst.

- `void CalculateBlobs()`

  Berechnet die Merkmale Größe und Lage des Schwerpunktes jeder einzelnen Farbfläche aus dem Marken–Bild, das durch die LegoIP–Methode `CalculateMarks()` erzeugt wurde. Anschließend werden die einzelnen Farbflächen daraufhin untersucht, welche die größte dieser Farbe ist. So wird sichergestellt, daß nur die Koordinaten der größten Fläche einer Farbe weitergegeben werden.

- `void DrawBlobs()`

  Zeichnet in dem Ergebnisbild `m_c_LegoDIB_result_pic` die Koordinaten der Schwerpunkte als schwarzen Punkt und einen Kreis mit einem Radius proportional zur Größe der Farbfläche. Der Kreis wird in der Farbe der erkannten Farbfläche gezeichnet.

Die genannten, zur Klasse LegoIP gehörenden Methoden, werden in einem separaten Task der Reihe nach ausgeführt. Der Task hat folgendes Aussehen:

```
UINT ImageProcessThread(LPVOID pParam) {
    CLegoIP* LegoIP = (CLegoIP*)pParam;
    LegoIP->CalculateThresholds();
    LegoIP->CleanUp();
    LegoIP->BinarySimple();
    LegoIP->BinaryExtended();
    LegoIP->CalculateMarks();
    LegoIP->CorrectMarks();
    LegoIP->CalculateBlobs();
    LegoIP->DrawBlobs();
    LegoIP->SendCoordinates();
    return 0;               // return 0 represents success
}
```

In Abbildung 5.2 ist das vereinfachte Klassen–Modell abgebildet.

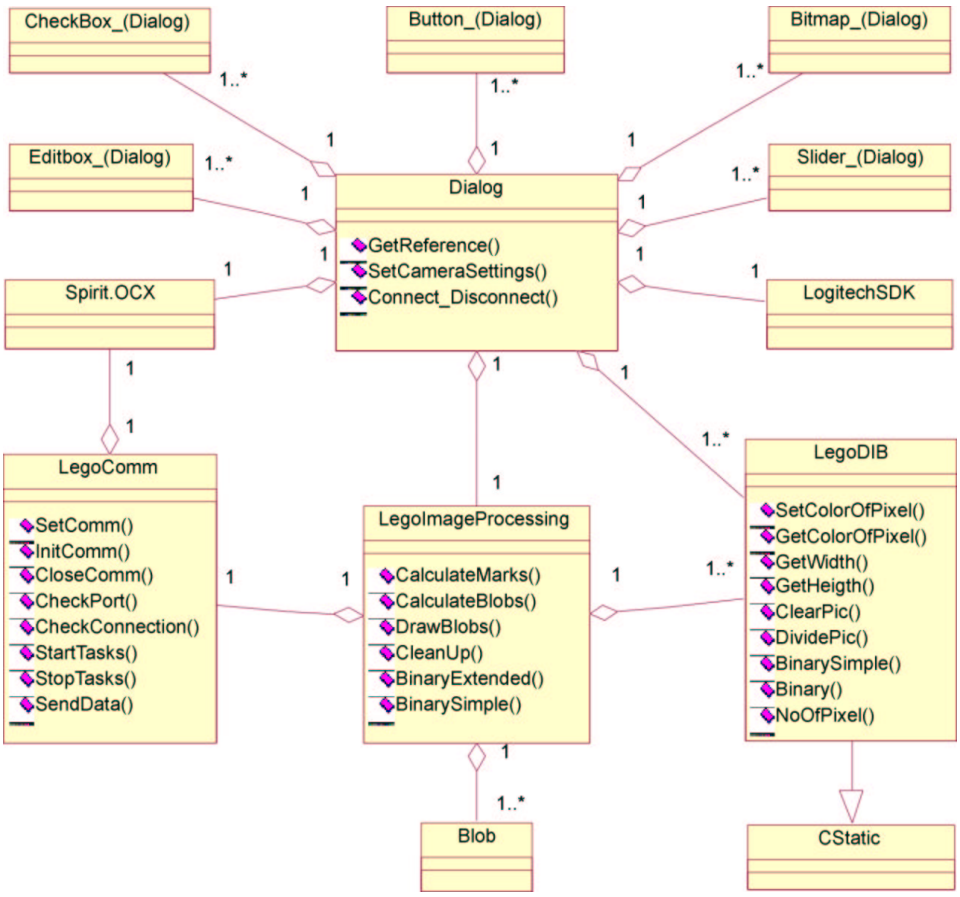

Abbildung 5.2: Vereinfachtes Klassen–Modell

5.5 Test–Szenario

Das entstandene Programm wurde nach der Fertigstellung an einem Szenario überprüft. Dieses Szenario wurde unter Berücksichtigung der Ausgangssituation aus Abschnitt 1.2 entwickelt. Hierbei galt es, einen Roboter nach Aufforderung abwechselnd auf eine grüne oder eine rote Farbfläche fahren zu lassen. Der Roboter selbst war mit einer gelben und einer blauen Farbfläche versehen, damit nicht nur seine Position sondern auch seine Orientierung ermittelt werden konnte.

Eine Bauanleitung für den LEGO Roboter im “LEGO Stil” findet man in Anhang A. Die auf dem RCX verwendete Software wurde mit NQC erstellt und kann in Anhang B nachgelesen werden. Die einzigen Sensoren, die der RCX des Robo-

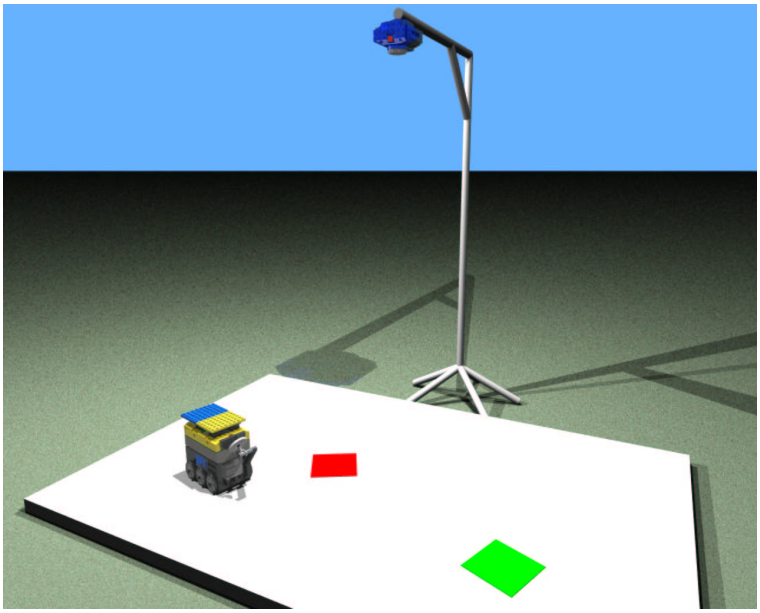

Abbildung 5.3: Prinzipieller Aufbau des Szenarios

ters verwendet, sind Rotations–Sensoren, die ein definiertes Fahren und Drehen erlauben.

Die Software wurde aufgrund dieses Szenarios unter verschiedenen Bedingungen und an verschiedenen Orten getestet. Sie wurde auf unterschiedlichen Systemen eingesetzt. Hierbei wurden Systeme mit *Intel*® *Pentium*® und *AMD*® *Athlon*™ Prozessoren eingesetzt, auf denen Windows 98 oder ME installiert war.

Selbst auf dem langsamsten, für diese Tests zur Verfügung stehenden System, einem *Intel*® *Pentium*® II mit 350 MHz lief die Software akzeptabel. Einzige Bedingungen an das System sind ein freier USB–Anschluß, ein freier serieller Anschluß sowie ein Auflösung von wenigstens 1024*768.[3]

Die Software wurde erfolgreich im Robot–Lab der Hochschule für Angewandte Wissenschaften Hamburg eingesetzt, ist im Rahmen des *Robot Contest* der Hochschule für Angewandte Wissenschaften Hamburg erfolgreich vorgeführt worden und wurde bereits mehrfach privat eingesetzt. Einige Bilder dieser Tests sind in Anhang C zu finden.

[3]Die grafische Benutzer–Schnittstelle des Programmes ist 1000*736 Pixel groß.

Kapitel 6

Resümee

Dieses Kapitel beschäftigt sich mit der entstandenen Software und deren Grenzen. Es werden Stärken und Schwächen aufgezeigt sowie Verbesserungs– und Erweiterungsvorschläge gemacht.

6.1 Ergebnis der Arbeit

Die entwickelte Software erfüllt die ursprüngliche Aufgabenstellung weitestgehend, wie Tests bewiesen haben.

Das Ergebnis der Bildverarbeitung hängt wesentlich von den Bedingungen ab, unter denen die Software eingesetzt wird. So ist die Auswahl der Farbe der zu erkennenden Flächen wichtig. Werden diese Farben nicht sorgsam ausgewählt, kann es sein, daß sie nicht richtig erkannt werden. So kommt es leicht vor, das eine grüne Fläche nicht als Grün, sondern als Cyan erkannt wird.

Ebenso wichtig ist eine möglichst gleichmäßige Ausleuchtung der Spielfläche. Tageslicht ist nach Möglichkeit zu vermeiden. Es hat den großen Nachteil, daß seine Helligkeit variiert. Dadurch funktioniert das Verfahren der Referenzbild–Division nicht mehr wie vorgesehen. Außerdem wird der Weißabgleich der Kamera durch Tageslicht beeinflußt. Der Blau–Anteil in Tageslicht ist wesentlich höher als der in Kunstlicht. Steigt die Helligkeit des Tageslichtes, wird das Bild "blauer". Damit stimmen die ausgewählten Farben nicht mehr.[1]

Das Spirit Control scheint nicht ganz ausgereift zu sein. Wie in Abschnitt 3.1 erwähnt, gibt es für die Infrarotkommunikation keine Kollisionserkennung. Findet während einer Übertragung zwischen RCX und PC–Software eine andere Infrarotkommunikation statt, endet das Programm mit einer Fehlermeldung.

Dieser Fehler ist z.B. mit der LEGO Fernbedienung[2] jederzeit reproduzierbar

[1] Grün, daß durch einen fehlerhaften Weißabgleich "blauer" wird, wird als Cyan erkannt.

[2] Diese Fernbedienung ist Teil des LEGOMINDSTORMS ULTIMATE ACCESSORY SET.

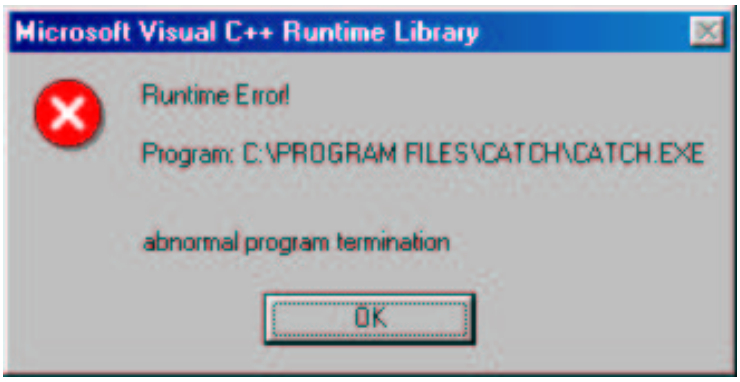

Abbildung 6.1: Fehlermeldung nach Programmabsturz

und läßt deshalb auf das Spirit Control schließen. Selbst wenn man die Fernbedienung benutzt, während die Software im Debug–Modus von Visual C++ läuft, wird lediglich die Fehlermeldung aus Abbildung 6.1 ausgegeben. Deshalb ist es leider nicht möglich, diesen Fehler genauer zu untersuchen oder sogar zu beheben.

Erst im Rahmen der ersten Tests hat sich herausgestellt, daß eine Synchronisation notwendig ist und eine kontinuierliche Übertragung der Daten zum RCX nicht sinnvoll ist. Die Gründe hierfür sind in Abschnitt 5.2.2 genannt.

6.2 Ausblick

Die entwickelte Software kann an einigen Stellen noch erweitert oder verbessert werden. Auf einige dieser Punkte wird im Folgenden eingegangen.

Das durch eine Referenzbild–Division gefilterte Bild hat an Kontrast verloren. Diesen Nachteil könnte man eliminieren, indem man das zu dem Bild gehörende Histogramm spreizt.

Ein Histogramm gibt die Häufigkeit der Intensitäten der einzelnen Farben an. Das Verfahren der Histogramm–Spreizung ist in [Jähne 1997] ausführlich erklärt.

Von Vorteil wäre es auch, wenn man die drei Schwellen für die Binarisierung nicht manuell einstellen müßte. Diese Schwellen lassen sich aus dem Histogramm entnehmen. Hierbei kann als Schwelle ein Minima zwischen zwei Maxima des Histogramms angenommen werden. Hierzu müßte jedes Histogramm der drei Farben ermittelt, geglättet und mathematisch beschrieben werden. Aus der mathematischen Beschreibung ließen sich das Minima und die Maxima des Histogramms und damit die Schwelle bestimmen.

Die entstandene Software bestimmt die Lage des Schwerpunktes und die Größe einer Fläche einer bestimmten Farbe. Es sind Aufgabenstellungen für einen Roboter denkbar, in dem die Kamerasoftware einen größeren Beitrag leisten sollte

als diese Größen. Unter Umständen wäre es wünschenswert, Informationen über die Form zu erhalten. So könnte es von Nutzen sein, die Lagen der Haupt– und Nebenachse und deren Längen zu kennen. Weiter wäre es denkbar, das Bild nicht nur auf Farben zu untersuchen sondern auch auf Formen. So könnte man das Bild auf Momente untersuchen und so Objekte translations–, rotations– und größeninvariant bestimmen. Allerdings ist die Menge der Informationen durch die Anzahl der von der LEGO Firmware zur Verfügung gestellten globalen Variablen beschränkt.

In der vorliegenden Software sind die verwalteten sechs Farbflächen von verschiedener Farbe. Es könnte aber erforderlich sein, mehrere Flächen einer Farbe zu beobachten. Deshalb wäre es nützlich, daß man die Farbe der sechs zu erkennenden Flächen in der laufenden Software festlegen kann.

Während diese Arbeit entstanden ist, hat LEGO einen Ersatz für das Spirit Control herausgegeben.[3] Diese GhostAPI[4] genannte Schnittstelle erlaubt nicht nur die Verwendung des seriellen Infrarotsenders sondern ebenfalls die des Senders für die USB–Schnittstelle. Die Verwendung des GhostAPI empfiehlt sich, um die entstandene Software auch den Benutzern des Robotics Invention System 2.0 zugänglich zu machen.

Als Alternative zu der Schnittstelle LEGO GhostAPI könnte man das LNP[5] genannte Netzwerk Protokoll von LegOS einsetzen. Dies ist allerdings wie in Abschnitt 3.1 erwähnt noch nicht sinnvoll einsetzbar. Wenn dieses Protokoll allerdings in Zukunft zuverlässig läuft, ist es eine attraktive Alternative, da mit LNP mehrere RCX gleichzeitig mit der PC–Software kommunizieren könnten. Außerdem wäre es mit LegOS möglich, wesentlich mehr Informationen an den RCX zu senden, da LegOS keine Beschränkung auf zweiunddreißig globale Variablen kennt.

6.3 Alternative Einsatzgebiete

Die Aufgabenstellung setzt voraus, daß die Kamera an einem Stativ hängend von oben auf das Spielfeld blickt. Dies liefert ein globales Weltmodell für den Roboter. Seine Welt besteht nur aus der von der Kamera sichtbaren Spielfläche.

Es wäre auch denkbar, die Kamera *auf* einem mobilen Roboter zu installieren. Die erstellte Software würde dann keine Koordinaten für ein globales Weltmodell liefern, sondern lediglich Koordinaten des Sichtbereichs, der von der Position und Richtung der Kamera (und somit des Roboters) abhängt. Das würde dazu führen, daß der Roboter ein eigenes lokales Weltbild *mit sich trägt* und keine Kenntnis

[3] Am 21. Dezember 2001 hat Michael Andersen das Erscheinen des neuen LEGO MINDSTORMS SDK im LUGNET bekannt gegeben.[Andersen]

[4] API ⇔ application program interface

[5] LNP ⇔ Lego Network Protokoll

der globalen Welt hat. Ein möglicher Anwendungsfall wäre die Erkennung von Landmarken für die Selbstlokalisation des mobilen Roboters.

Ein solcher Einsatz sollte möglich sein. Allerdings kann die Referenzbild–Division in einem solchen Fall keine Beleuchtungsschwächen korrigieren. Denn im Gegensatz zu dem ursprünglichen Einsatzgebiet, bei dem durch die Referenzbild–Division eine Trennung von Hintergrund und Objekt erfolgt, ändert sich hier der Hintergrund. Die Referenzbild–Division kann lediglich der Korrektur von Kameraschwächen dienen. Dazu müßte ein Referenzbild von einer weißen, gleichmäßig beleuchteten Fläche aufgenommen werden. In einem solchen Fall wäre es dann sehr nützlich, wenn man dieses Referenzbild speichern und nach einem Neustart der Software wieder laden könnte.

Da die Beleuchtungssituation sich ständig ändern könnte, treten die Probleme, die in Abschnitt 6.1 beschrieben wurden, bei einer solchen Anwendung vermehrt auf. Eine erhebliche Hilfe für einen solchen Einsatz wäre die automatische Ermittlung der Schwellen zur Binarisierung.

Abbildungsverzeichnis

Literaturverzeichnis

1 Pioneer : *ActiveMedia Robotics*. online. – URL http://www.activrobots.com/. – Zugriffsdatum: 7.1.2002

2 Newton : *The Cognachrome Vision System*. online. – URL http://www.newtonlabs.com/cognachrome/index.html. – Zugriffsdatum: 7.1.2002

3 LUGNET : *Global community of LEGO enthusiasts*. online. – URL http://www.lugnet.com. – Zugriffsdatum: 7.1.2002

4 LUGNews : *Global community of LEGO enthusiasts - Robotics news*. online. – URL http://news.lugnet.com/robotics. – Zugriffsdatum: 7.1.2002

5 SunJDK : *JAVA^{TM} DEVELOPMENT KIT Version 1.1.x*. online. – URL http://java.sun.com/products/jdk/1.1/. – Zugriffsdatum: 7.1.2002

6 Krabat : *Das Krabat-Projekt*. online. – URL http://www.informatik.fh-hamburg.de/~krabat/. – Zugriffsdatum: 7.1.2002

7 LegOS : *The legOS Homepage*. online. – URL http://legos.sourceforge.net/. – Zugriffsdatum: 7.1.2002

8 LTQCSDK : *Logitech's QuickCam Developer Program*. online. – URL http://developer.logitech.com/sdk/. – Zugriffsdatum: 7.1.2002

9 PLD PITSCO LEGO dacta (Veranst.): *PITSCO LEGO dacta online store*. online. – URL http://www.pldstore.com/. – Zugriffsdatum: 7.1.2002. – Didaktik Abteilung von LEGO

10 RoboCup : *RoboCup 2002*. online. – URL http://www.robocup.org. – Zugriffsdatum: 7.1.2002

11 RSNewton : *Robot Soccer at Newton Research Labs*. online. – URL http://www.newtonlabs.com/soccer/. – Zugriffsdatum: 7.1.2002

12 VfW2WDM 1998 : *VfW-to-WDM Video Capture Mapper on Windows 98 and Windows 2000*. online. Dezember 1998. – URL http://msdn.microsoft.com/library/en-us/dnwbgen/html/vfwmap.asp. – Zugriffsdatum: 7.1.2002

13 WDM 1998 : *WDM Video Capture Overview*. online. Dezember 1998. – URL http://msdn.microsoft.com/library/en-us/dnwbgen/html/vidcap.asp. – Zugriffsdatum: 7.1.2002

14 LeJOS 2001 : *LEJOS Java for the RCX*. online. Oktober 2001. – URL http://lejos.sourceforge.net/. – Zugriffsdatum: 7.1.2002

15 VCTech 2001 : *Video Capture Technologies*. online. Dezember 2001. – URL http://www.microsoft.com/hwdev/tech/stream/vidcap/default.asp. – Zugriffsdatum: 7.1.2002

16 Andersen ANDERSEN, Michael: *New MindStorms SDK*. online. – URL http://news.lugnet.com/robotics/?n=16802. – Zugriffsdatum: 7.1.2002. – Nachricht von Michael Andersen im LUGNET

17 Balzerowski BALZEROWSKI, Rainer: *3 Sharps an einem Eingang*. online. – URL http://www.informatik.fh-hamburg.de/~lego/Projekte/multi_sharp/multi_sharp_deutsch.html. – Zugriffsdatum: 7.1.2002

18 Bantz BANTZ, E. J.: *Getting Started with Video for Windows and Visual Basic*. online. – URL http://www.videoforwindows.com/articles/docs/art1004.asp. – Zugriffsdatum: 7.1.2002

19 Baum BAUM, Dave: *Not Quite C*. online. – URL http://www.enteract.com/~dbaum/nqc. – Zugriffsdatum: 7.1.2002

20 Baum 2000 BAUM, Dave: *Definitive Guide to LEGO MINDSTORMS*. Emeryville (CA), USA : APress, 2000. – ISBN 1–893115–09–7

21 Baum u. a. 2000 BAUM, Dave ; GASPERI, Michael ; HEMPEL, Ralph ; VILLA, Luis: *Extreme MINDSTORMS (An advanced Guide to LEGO MINDSTORMS)*. Berkeley (CA), USA : APress, 2000. – ISBN 1–893115–84–4

22 Bässmann und Kreyss 1998 BÄSSMANN, Henning ; KREYSS, Jutta: *Bildverarbeitung Ad Oculos*. dritte vollständig überarbeitete und erweiterte Auflage. Berlin : Springer–Verlag, 1998. – ISBN 3–540–63649–8

23 DUDEN 1994 DUDEN ; DROSDOWSKI, Günther (Hrsg.): *DUDEN, Das Große Fremdwörterbuch*. Mannheim : Dudenverlag, 1994. – ISBN 3–411–04161–7

24 Ezzel und Blaney 1997 EZZEL, Ben ; BLANEY, Jim: *Das Programmierbuch Windows 95/NT 4*. erste deutsch Ausgabe. Düsseldorf : SYBEX–Verlag GmbH, 1997. – Titel der amerikanischen Originalausgabe: Windows 95/NT4 Developer's Handbook. – ISBN 3–8155–7181–2

25 Gasperi GASPERI, Michael: *MindStorms RCX Sensor Input Page*. online. – URL `http://www.plazaearth.com/usr/gasperi/lego.htm`. – Zugriffsdatum: 7.1.2002

26 Haberäcker 1995 HABERÄCKER, Peter: *Praxis der Digitalen Bildverarbeitung und Mustererkennung*. München : Carl Hanser Verlag, 1995. – ISBN 3–446–15517–1

27 Hempel HEMPEL, Ralph: *pbForth Home Page*. online. – URL `http://www.hempeldesigngroup.com/lego/pbFORTH/`. – Zugriffsdatum: 7.1.2002

28 Jähne 1997 JÄHNE, Bernd: *Digitale Bildverarbeitung*. vierte völlig neubearbeitete Auflage. Berlin : Springer–Verlag, 1997. – ISBN 3–540–61379–X

29 Kopp 1997 KOPP, Herbert: *Bildverarbeitung interactiv*. Stuttgart : B.G. Teubner, 1997. – ISBN 3–519–02995–2

30 Kunze 1994 KUNZE, Michael: Gezähmter Strom; TWAIN – ein universelles Interface für die Bilddigitalisierung. In: *c´t magazin für computer technik* 2 (1994), Februar, S. 156–158. – ISSN 0724-8679

31 Lamport 1995 LAMPORT, Leslie: *Das LaTeX Handbuch*. deutsche Ausgabe. Bonn : Addison–Wesley, 1995. – ISBN 3–89319–826–1

32 LEGO LEGO: *RCX 2.0 Beta Software Developers' Kit*. online. – URL `http://mindstorms.lego.com/sdk2/default.asp`. – Zugriffsdatum: 7.1.2002

33 LEGO 1998 LEGO: *Controlling LEGO Programmable Bricks Technical Reference*. PBRICK.PDF. November 1998. – Im Unfang des RCX 2.0 Beta Software Developers' Kit enthalten

34 Logitech 2000 LOGITECH: *Logitech QuickCam SDK 1.0 Microsoft MFC Programmer's Guide*. QCSDKMFC.PDF. 2000. – Im Umfang des Logitech Quickcam SDK enthalten

35 von Luck LUCK, Kai von: *IKS-project (Integration Kognitiver Systeme)*. online. – URL `http://www.informatik.fh-hamburg.de/~robots/`. – Zugriffsdatum: 7.1.2002

36 Mader MADER, Lan: *DirectDraw Programming Tutorial*. online. – URL http://www.geocities.com/SiliconValley/Bay/2535/ddrawtut.html. – Zugriffsdatum: 7.1.2002

37 Martin MARTIN, Fred: *6.270: The MIT LEGO Robot Design Competition*. online. – URL http://fredm.www.media.mit.edu/people/fredm/projects/6270/. – Zugriffsdatum: 7.1.2002

38 Martin u. a. MARTIN, Fred ; MITCHEL, Resnick ; BRIAN, Silverman u. a.: *The MIT Programmable Brick*. online. – URL http://el.www.media.mit.edu/groups/el/projects/programmable-brick/. – Zugriffsdatum: 7.1.2002

39 Neumann 2000 NEUMANN, Bernd ; GÖRZ, Günther (Hrsg.) ; ROLLINGER, Claus-Rainer (Hrsg.) ; SCHNEEBERGER, Josef (Hrsg.): *Handbuch der künstlichen Intelligenz*. dritte vollständig überarbeitete Auflage. München : Oldenbourg Verlag, 2000. – 815ff. S. – ISBN 3–486–25049–3

40 Nilsson 1998 NILSSON, Nils J.: *Artifical Intelligence; A new Synthesis*. San Francisco (CA), USA : Morgan Kaufmann Publisher, Inc., April 1998. – ISBN 1-55860-467-7

41 Proudfoot PROUDFOOT, Kekoa: *RCX Internals*. online. – URL http://graphics.stanford.edu/~kekoa/rcx/. – Zugriffsdatum: 7.1.2002

42 Richter 1997 RICHTER, Jeffrey: *Microsoft Windows Programmierung für Experten*. dritte vollständig überarbeitete deutsche Auflage. Unterschleißheim : Microsoft Press Deutschland, 1997. – Titel der amerikanischen Originalausgabe: Advanced Windows, 3rd ed.. – ISBN 3–86063–389–9

43 Schlicht 1995 SCHLICHT, Hans-Jürgen: *Bildverarbeitung digital: Scanner, Drucker, Video, Multimedia unter Windows*. zweite komplett überarbeitete Auflage. Bonn : Addison–Wesley, 1995. – ISBN 3-89319-889-X

44 Solorzano 2000 SOLORZANO, Jose H.: *Q&A on background of leJOS and TinyVM*. online. Oktober 2000. – URL http://sourceforge.net/forum/forum.php?forum_id=41249. – Zugriffsdatum: 7.1.2002

45 Wischmann u. a. WISCHMANN, Arne ; STOLT, Matthias ; BALZEROWSKI, Rainer: *Lego Robots@FH-Hamburg*. online. – URL http://www.informatik.fh-hamburg.de/~lego/. – Zugriffsdatum: 7.1.2002

46 Young 1997 YOUNG, Michael J.: *Das Programierbuch Visual C++ 5*. erste deutsch Ausgabe. Düsseldorf : SYBEX–Verlag GmbH, 1997. – Titel der amerikanischen Originalausgabe: Mastering Microsoft Visual C++ 5. – ISBN 3–8155–5419–5

Anhang A

Bauanleitung des verwendeten LEGO Roboters

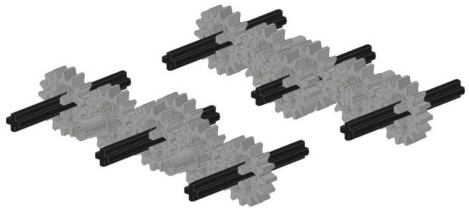

Abbildung A.1: Bauanleitung Schritt 1

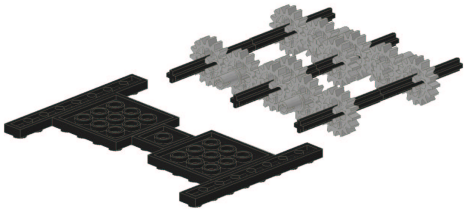

Abbildung A.2: Bauanleitung Schritt 2

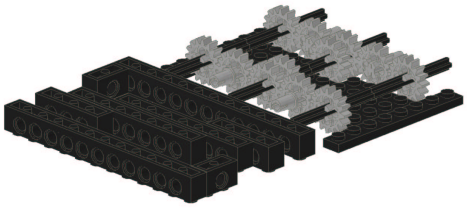

Abbildung A.3: Bauanleitung Schritt 3

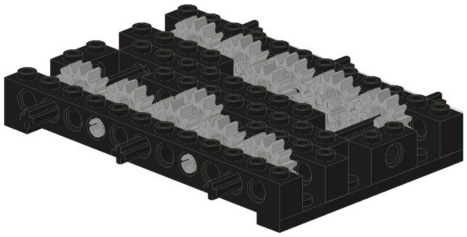

Abbildung A.4: Bauanleitung Schritt 4

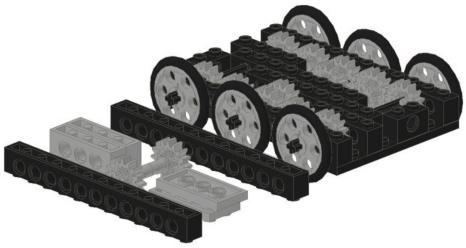

Abbildung A.5: Bauanleitung Schritt 5

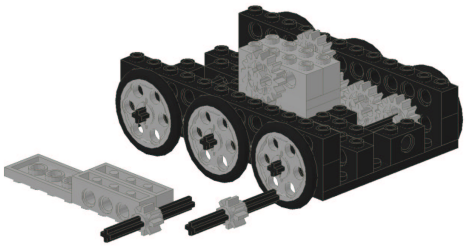

Abbildung A.6: Bauanleitung Schritt 6

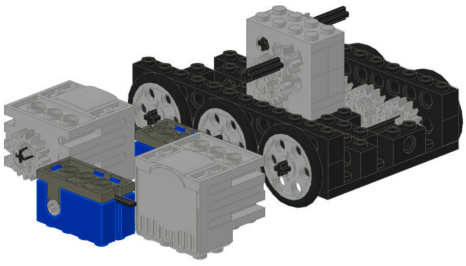

Abbildung A.7: Bauanleitung Schritt 7

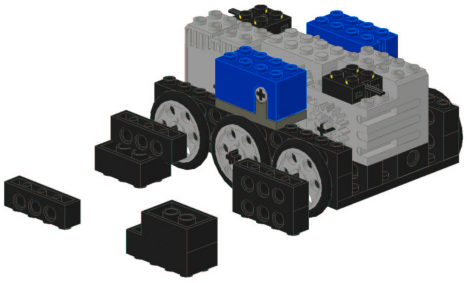

Abbildung A.8: Bauanleitung Schritt 8

Abbildung A.9: Bauanleitung Schritt 9

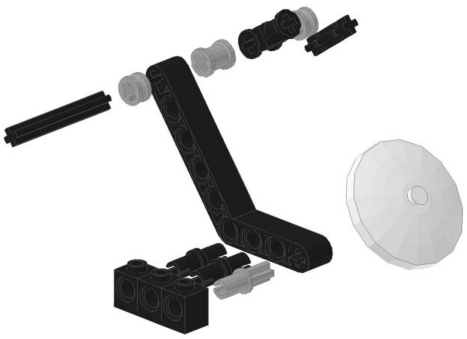

Abbildung A.10: Bauanleitung Schritt 10

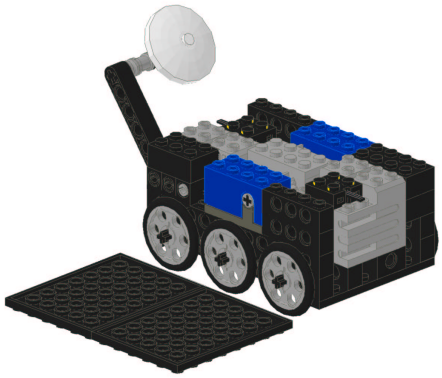

Abbildung A.11: Bauanleitung Schritt 11

Abbildung A.12: Bauanleitung Schritt 12

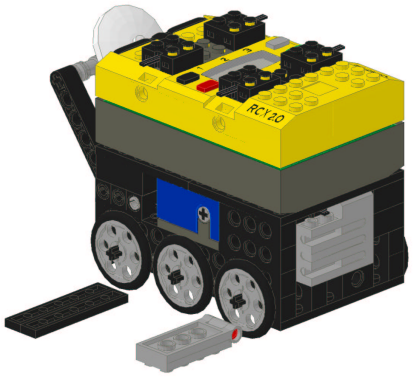

Abbildung A.13: Bauanleitung Schritt 13

Abbildung A.14: Bauanleitung Schritt 14

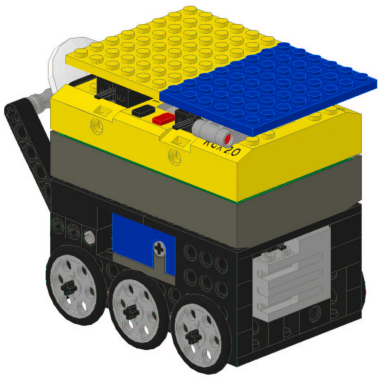

Abbildung A.15: Bauanleitung Schritt 15

Anhang B

Software auf dem verwendeten LEGO Roboter

Der folgende Code befindet sich ebenfalls als File “vorfuehr.nqc” auf der CD.

```
#define grenzwert        5
#define zielgrenzwert   10

#define keine            0
#define nord             1
#define nordost          2
#define ost              3
#define suedost          4
#define sued             5
#define suedwest         6
#define west             7
#define nordwest         8

#define rot              1
#define gruen            2

#define links            1
#define rechts           2

#define dz_links        SENSOR_1
#define dz_rechts       SENSOR_3

#define m_links         OUT_A
#define m_rechts        OUT_C

#define wait_time       200

#define cal_dist        150
#define cal_dreh        180
```

```
// Tabelle mit Tangenswerten
#define G5              9      // -> tan (5°)  =  0.0875 // mal 100 weil kein float
#define G10            18      // -> tan (10°) =  0.1763 // mal 100 weil kein float
#define G15            27      // -> tan (15°) =  0.2679 // mal 100 weil kein float
#define G20            36      // -> tan (20°) =  0.364  // mal 100 weil kein float
#define G25            47      // -> tan (25°) =  0.4663 // mal 100 weil kein float
#define G30            58      // -> tan (30°) =  0.5774 // mal 100 weil kein float
#define G35            70      // -> tan (35°) =  0.7002 // mal 100 weil kein float
#define G40            84      // -> tan (40°) =  0.8391 // mal 100 weil kein float
#define G45           100      // -> tan (45°) =  1      // mal 100 weil kein float
#define G50           119      // -> tan (50°) =  1.1918 // mal 100 weil kein float
#define G55           143      // -> tan (55°) =  1.4281 // mal 100 weil kein float
#define G60           173      // -> tan (60°) =  1.7321 // mal 100 weil kein float
#define G65           214      // -> tan (65°) =  2.1445 // mal 100 weil kein float
#define G70           275      // -> tan (70°) =  2.7475 // mal 100 weil kein float
#define G75           373      // -> tan (75°) =  3.7321 // mal 100 weil kein float
#define G80           567      // -> tan (80°) =  5.6713 // mal 100 weil kein float
#define G85          1143      // -> tan (85°) = 11.4301 // mal 100 weil kein float

// Tabelle mit Quadratwerten
#define Q10            100      // Wurzel 100   = 10
#define Q20            400      // Wurzel 400   = 20
#define Q30            900      // Wurzel 900   = 30
#define Q40           1600      // Wurzel 1600  = 40
#define Q50           2500      // Wurzel 2500  = 50
#define Q60           3600      // Wurzel 3600  = 60
#define Q70           4900      // Wurzel 4900  = 70
#define Q80           6400      // Wurzel 6400  = 80
#define Q90           8100      // Wurzel 8100  = 90
#define Q100         10000      // Wurzel 10000 = 100
#define Q110         12100      // Wurzel 12100 = 110
#define Q120         14400      // Wurzel 14400 = 120
#define Q130         16900      // Wurzel 16900 = 130
#define Q140         19600      // Wurzel 19600 = 140
#define Q150         22500      // Wurzel 22500 = 150
#define Q160         25600      // Wurzel 25600 = 160
#define Q170         28900      // Wurzel 28900 = 170
#define Q180         32400      // Wurzel 32400 = 180

// Wichtig ! Die ersten 12 Variablen repraesentieren die X und Y Koordinaten
// der Farben BLAU, GRUEN, CYAN, ROT, MAGENTA, GELB  und zwar in dieser Reihenfolge.

int blau_x = 0;                  //  0
int blau_y = 0;                  //  1

int gruen_x = 0;                 //  2
int gruen_y = 0;                 //  3
```

```
int cyan_x = 0;                    //  4
int cyan_y = 0;                    //  5

int rot_x = 0;                     //  6
int rot_y = 0;                     //  7

int magenta_x = 0;                 //  8
int magenta_y = 0;                 //  9

int gelb_x = 0;                    // 10
int gelb_y = 0 ;                   // 11

int sync = 1;                      // 12

int robo_richtung;                 // 13
int ziel_richtung;                 // 14

int entfernung = 0;                // 15

int temp_x = 0;                    // 16
int temp_y = 0;                    // 17

int ziel = 0;                      // 18

int fahr_multi = 100;              // 19
int dreh_div = 36;                 // 20

int degree = 0;                    // 21

task main()
{
  while (true);
}

task fahr_zu_rot()
{
  SetSensor (dz_links,SENSOR_ROTATION);
  SetSensor (dz_rechts,SENSOR_ROTATION);
  SetPower(m_links,8);
  SetPower(m_rechts,8);

  SetTxPower(TX_POWER_HI);

  ziel = rot;

  cameracontrol();
  calc_robo();
  calc_ziel();
```

```
  if (ziel_richtung > robo_richtung){
     degree = ziel_richtung - robo_richtung;
     if (degree > 180){
        degree = 360 - degree;
        degree *= -1;
     }
     else {
     }
  }
  else {
     degree = robo_richtung - ziel_richtung;
     if (degree > 180){
       degree = 360 - degree;
     }
     else {
       degree *= -1;
     }
  }
  turn();
  drive();
}

task fahr_zu_gruen()
{
  SetSensor (dz_links,SENSOR_ROTATION);
  SetSensor (dz_rechts,SENSOR_ROTATION);
  SetPower(m_links,8);
  SetPower(m_rechts,8);

  SetTxPower(TX_POWER_HI);

  ziel = gruen;

  cameracontrol();
  calc_robo();
  calc_ziel();

  if (ziel_richtung > robo_richtung){
     degree = ziel_richtung - robo_richtung;
     if (degree > 180){
        degree = 360 - degree;
        degree *= -1;
     }
     else {
     }
  }
  else {
     degree = robo_richtung - ziel_richtung;
     if (degree > 180){
```

```
      degree = 360 - degree;
    }
    else {
      degree *= -1;
    }
  }
  turn();
  drive();
}

task calibrate()
{
  SetSensor (dz_links,SENSOR_ROTATION);
  SetSensor (dz_rechts,SENSOR_ROTATION);
  SetPower(m_links,8);
  SetPower(m_rechts,8);

  SetTxPower(TX_POWER_HI);

  ClearSensor (dz_links);
  ClearSensor (dz_rechts);

  cameracontrol();

  temp_x = gelb_x;              // temp_x ist nur Platzhalter um Wert fuer
                                // gelb_x zwischen zu speichern
                                // (Variablen sparen !!!)
  OnFwd(m_rechts+m_links);

  while (dz_rechts < cal_dist || dz_links > -cal_dist){
    if (dz_rechts > cal_dist)
      Off (m_rechts);
    if (dz_links < -cal_dist)
      Off (m_links);
  }
  Off (m_links+m_rechts);

  ClearSensor (dz_links);
  ClearSensor (dz_rechts);

  cameracontrol();

  if (gelb_x <= temp_x){
    PlaySound(0);
    PlaySound(0);
    PlaySound(0);
  }
  else {
    fahr_multi = (100 * cal_dist) / (gelb_x - temp_x);
```

```
    OnRev(m_rechts+m_links);
    while (dz_rechts > -cal_dist/2 || dz_links < cal_dist/2){
      if (dz_rechts < -cal_dist/2)
         Off (m_rechts);
      if (dz_links > cal_dist/2)
         Off (m_links);
    }
    Off (m_links+m_rechts);

    cameracontrol();
    calc_robo();
    degree = cal_dreh;
    turn ();
    degree = robo_richtung;
    cameracontrol();
    calc_robo();
    if (degree > robo_richtung){
      robo_richtung += 360;
    }
    entfernung = degree;    // nur zum speichern
    degree = robo_richtung - degree;
    dreh_div = (dreh_div * degree) / cal_dreh;  }
}

void calc_ziel ()
{
  int delta_x = 0;
  int delta_y = 0;
  if (ziel == gruen) {
     temp_x = gruen_x;
     temp_y = gruen_y;
  }
  else {
     temp_x = rot_x;
     temp_y = rot_y;
  }
  if (gelb_x != 0 && gelb_y != 0 && temp_x != 0 && temp_y != 0){
  // Lage des Ziel berechnen
    delta_x = temp_x - gelb_x;
    delta_y = temp_y - gelb_y;
    if (delta_x == 0){                    // nord, sued oder Daten nicht OK
      if (delta_y == 0){                  // Daten nicht OK !
        ziel_richtung = -1;
      }
      else {
        if (delta_y < 0){                 // 4. Quadrant -> sued !
          ziel_richtung = 270;
        }
```

```
    else{
      ziel_richtung = 90;             // 2. Quadrant -> nord !
    }
  }
}
else{
  if (delta_y == 0){                  // ost, west !
    if (delta_x < 0){
      ziel_richtung = 180;            // 3. Quadrant -> west !
    }
    else {
      ziel_richtung = 0;              // 1. Quadrant -> ost !
    }
  }
  else{                               // nicht ost, sued, west oder nord !
    ziel_richtung = (abs(delta_y)*100)/abs(delta_x);
    if (ziel_richtung < G5){
      ziel_richtung = 5;
    }
    else if (ziel_richtung < G10){
      ziel_richtung = 10;
    }
    else if (ziel_richtung < G15){
      ziel_richtung = 15;
    }
    else if (ziel_richtung < G20){
      ziel_richtung = 20;
    }
    else if (ziel_richtung < G25){
      ziel_richtung = 25;
    }
    else if (ziel_richtung < G30){
      ziel_richtung = 30;
    }
    else if (ziel_richtung < G35){
      ziel_richtung = 35;
    }
    else if (ziel_richtung < G40){
      ziel_richtung = 40;
    }
    else if (ziel_richtung < G45){
      ziel_richtung = 45;
    }
    else if (ziel_richtung < G50){
      ziel_richtung = 50;
    }
    else if (ziel_richtung < G55){
      ziel_richtung = 55;
    }
```

```
    else if (ziel_richtung < G60){
      ziel_richtung = 60;
    }
    else if (ziel_richtung < G65){
      ziel_richtung = 65;
    }
    else if (ziel_richtung < G70){
      ziel_richtung = 70;
    }
    else if (ziel_richtung < G75){
      ziel_richtung = 75;
    }
    else if (ziel_richtung < G80){
      ziel_richtung = 80;
    }
    else if (ziel_richtung < G85){
      ziel_richtung = 85;
    }
    else{
      ziel_richtung = 90;
    }
    if (delta_x < 0){                 // 2. oder 3. Quadrant
      if (delta_y < 0){               // 3. Quadrant
        ziel_richtung = 180 + ziel_richtung;
      }
      else{                           // 2. Quadrant
        ziel_richtung = 180 - ziel_richtung ;
      }
    }
    else{                             // 1. oder 4. Quadrant
      if (delta_y < 0){               // 4. Quadrant
         ziel_richtung = 360 - ziel_richtung;
      }
      else{                           // 1. Quadrant
      }
    }
  }
}
temp_x = abs (delta_x);
temp_y = abs (delta_y);
if (temp_x >= 180 || temp_y >= 180 ||  // 180^2 ist zu gross fuer short !!
   (temp_x >= 128 && temp_y >= 128)){  // 128^2 + 128^2 ist zu gross fuer short !!
   entfernung = 180;
}
else {
   entfernung = (temp_x*temp_x)+(temp_y*temp_y);
   if (entfernung >= Q180){
      entfernung = 180;
   } else if (entfernung >= Q170){
```

```
          entfernung = 170;
        } else if (entfernung >= Q160){
          entfernung = 160;
        } else if (entfernung >= Q150){
          entfernung = 150;
        } else if (entfernung >= Q140){
          entfernung = 140;
        } else if (entfernung >= Q130){
          entfernung = 130;
        } else if (entfernung >= Q120){
          entfernung = 120;
        } else if (entfernung >= Q110){
          entfernung = 110;
        } else if (entfernung >= Q100){
          entfernung = 100;
        } else if (entfernung >= Q90){
          entfernung = 90;
        } else if (entfernung >= Q80){
          entfernung = 80;
        } else if (entfernung >= Q70){
          entfernung = 70;
        } else if (entfernung >= Q60){
          entfernung = 60;
        } else if (entfernung >= Q50){
          entfernung = 50;
        } else if (entfernung >= Q40){
          entfernung = 40;
        } else if (entfernung >= Q30){
          entfernung = 30;
        } else if (entfernung >= Q20){
          entfernung = 20;
        } else if (entfernung >= Q10){
          entfernung = 10;
        } else {
          entfernung = -1;
        }
      }
    }
    else {
      ziel_richtung = -1;
    }
}

void calc_robo ()
{
  int delta_x = 0;
  int delta_y = 0;
  if (gelb_x != 0 && gelb_y != 0 && blau_x != 0 && blau_y != 0){
  // Stellung des Robis berechnen
```

```
delta_x = gelb_x - blau_x;
delta_y = gelb_y - blau_y;
if (delta_x == 0){                    // nord, sued oder Daten nicht OK
  if (delta_y == 0){                  // Daten nicht OK !
    robo_richtung = -1;
  }
  else {
    if (delta_y < 0){                 // 4. Quadrant -> sued !
      robo_richtung = 270;
    }
    else{
      robo_richtung = 90;             // 2. Quadrant -> nord !
    }
  }
}
else{
  if (delta_y == 0){                  // ost, west !
    if (delta_x < 0){
      robo_richtung = 180;            // 3. Quadrant -> west !
    }
    else {
      robo_richtung = 0;              // 1. Quadrant -> ost !
    }
  }
  else{                               // nicht ost, sued, west oder nord !
    robo_richtung = abs(delta_y)*100;
    robo_richtung /= abs(delta_x);
    if (robo_richtung < G5){
      robo_richtung = 5;
    }
    else if (robo_richtung < G10){
      robo_richtung = 10;
    }
    else if (robo_richtung < G15){
      robo_richtung = 15;
    }
    else if (robo_richtung < G20){
      robo_richtung = 20;
    }
    else if (robo_richtung < G25){
      robo_richtung = 25;
    }
    else if (robo_richtung < G30){
      robo_richtung = 30;
    }
    else if (robo_richtung < G35){
      robo_richtung = 35;
    }
    else if (robo_richtung < G40){
```

```
    robo_richtung = 40;
  }
  else if (robo_richtung < G45){
    robo_richtung = 45;
  }
  else if (robo_richtung < G50){
    robo_richtung = 50;
  }
  else if (robo_richtung < G55){
    robo_richtung = 55;
  }
  else if (robo_richtung < G60){
    robo_richtung = 60;
  }
  else if (robo_richtung < G65){
    robo_richtung = 65;
  }
  else if (robo_richtung < G70){
    robo_richtung = 70;
  }
  else if (robo_richtung < G75){
    robo_richtung = 75;
  }
  else if (robo_richtung < G80){
    robo_richtung = 80;
  }
  else if (robo_richtung < G85){
    robo_richtung = 85;
  }
  else{
    robo_richtung = 90;
  }

  if (delta_x < 0){                 // 2. oder 3. Quadrant
    if (delta_y < 0){               // 3. Quadrant
      robo_richtung = 180 + robo_richtung;
    }
    else{                           // 2. Quadrant
      robo_richtung = 180 - robo_richtung ;
    }
  }
  else{                             // 1. oder 4. Quadrant
    if (delta_y < 0){               // 4. Quadrant
       robo_richtung = 360 - robo_richtung;
    }
    else{                           // 1. Quadrant
    }
  }
}
```

```
    }
  }
  else {
    robo_richtung = -1;
  }
}

void turn()
{
  ClearSensor (dz_links);
  ClearSensor (dz_rechts);

  degree = (degree * 10) / dreh_div;     // mal 10 weil kein float moeglich
  if (degree > 0) { // links drehen !    // und division folgt!
    OnFwd(m_rechts);
    OnRev(m_links);
    while (dz_rechts < degree ||
           dz_links < degree) {
    }
    Off (m_links+m_rechts);
  }
  else { // rechts drehen !
    OnRev(m_rechts);
    OnFwd(m_links);
    while (dz_rechts > degree ||
           dz_links > degree) {
    }
    Off (m_links+m_rechts);
  }
}

void cameracontrol()
{
  Wait (wait_time); // Kamerabild aktualisieren lassen


  cyan_x = 0;          // Alle Werte auf Null, damit Werte
  cyan_y = 0;          // auf alle Faelle neu sind !
  magenta_x = 0;
  magenta_y = 0;
  blau_x = 0;
  blau_y = 0;
  gelb_x = 0;
  gelb_y = 0;
  gruen_x = 0;
  gruen_y = 0;
  rot_x = 0;
  rot_y = 0 ;
```

```
  sync = 1; // Kontrolle an Kamerasoftware uebergeben

  while (sync); // nichts tun solange Kamerasoftware die Kontrolle hat !
}

void drive(){

  ClearSensor (dz_links);
  ClearSensor (dz_rechts);

  OnFwd(m_rechts+m_links);
  entfernung *= fahr_multi;
  entfernung /= 100;
  while (dz_rechts < entfernung || dz_links > -entfernung){
    if (dz_rechts > cal_dist)
      Off (m_rechts);
    if (dz_links < -cal_dist)
      Off (m_links);
  }
  Off (m_links+m_rechts);
}
```

Anhang C

Fotos der Software im Einsatz

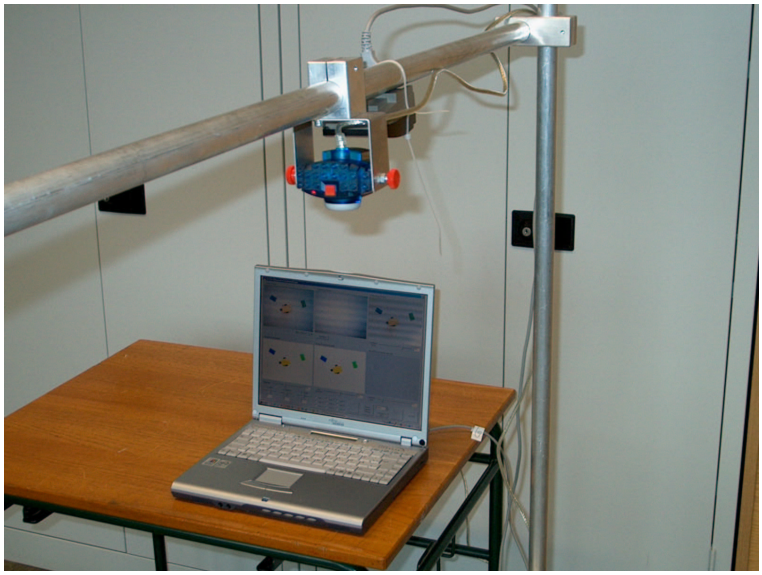

Abbildung C.1: Robot–Lab der HAW Hamburg 1

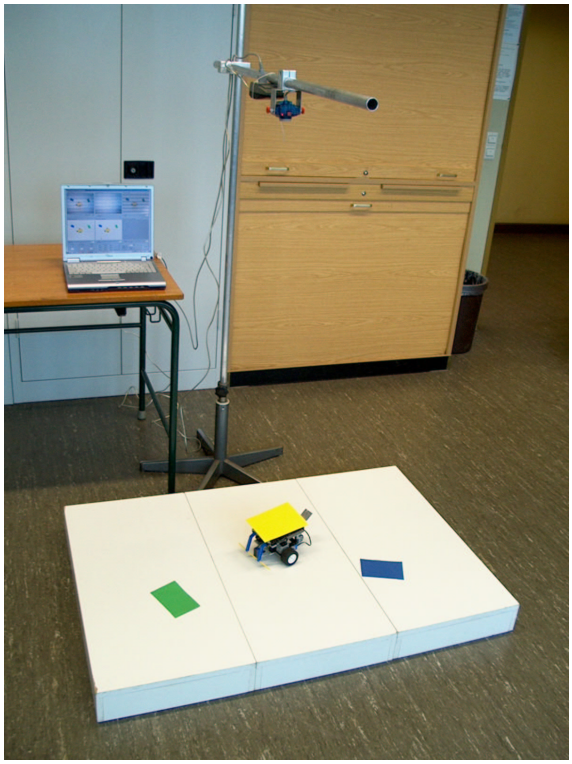

Abbildung C.2: Robot–Lab der HAW Hamburg 2

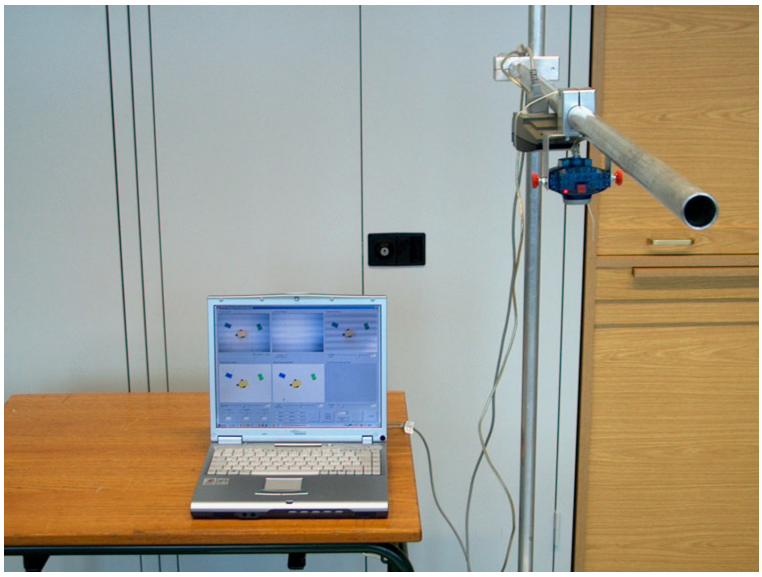

Abbildung C.3: Robot–Lab der HAW Hamburg 3

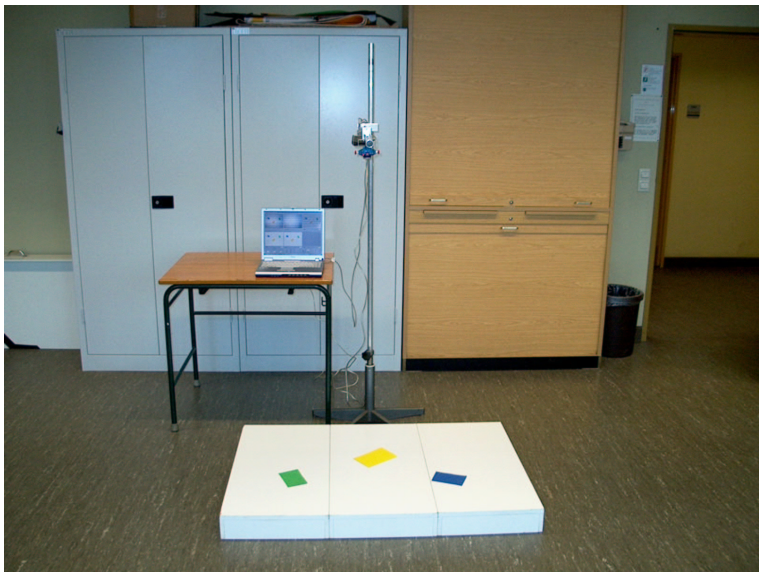

Abbildung C.4: Robot–Lab der HAW Hamburg 4

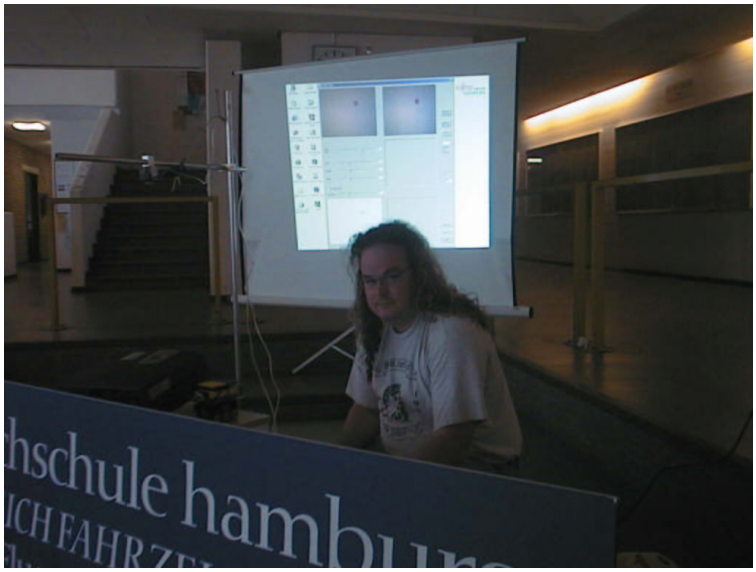

Abbildung C.5: Robot Contest der HAW Hamburg 1

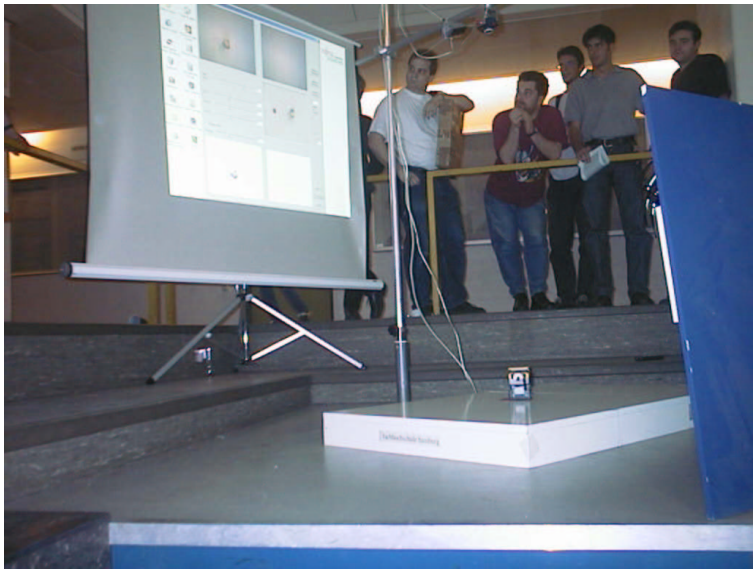

Abbildung C.6: Robot Contest der HAW Hamburg 2

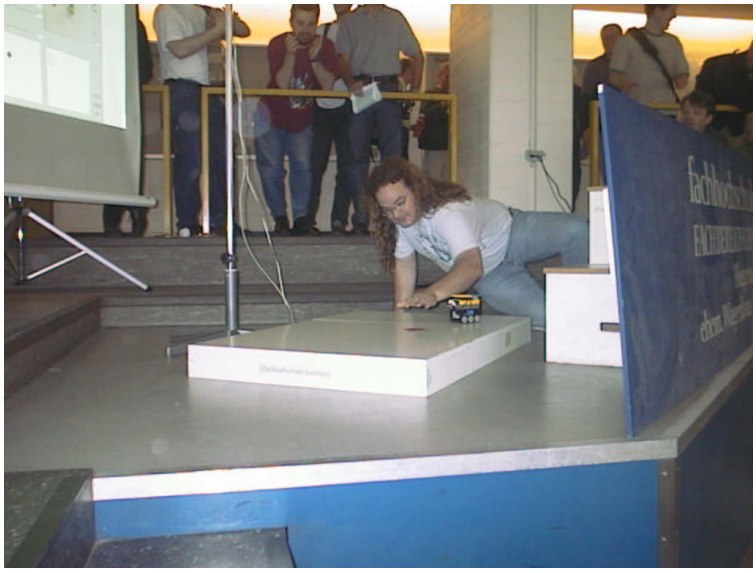

Abbildung C.7: Robot Contest der HAW Hamburg 3

Abbildung C.8: Robot Contest der HAW Hamburg 4

Abbildung C.9: Robot Contest der HAW Hamburg 5

Abbildung C.10: Robot Contest der HAW Hamburg 6

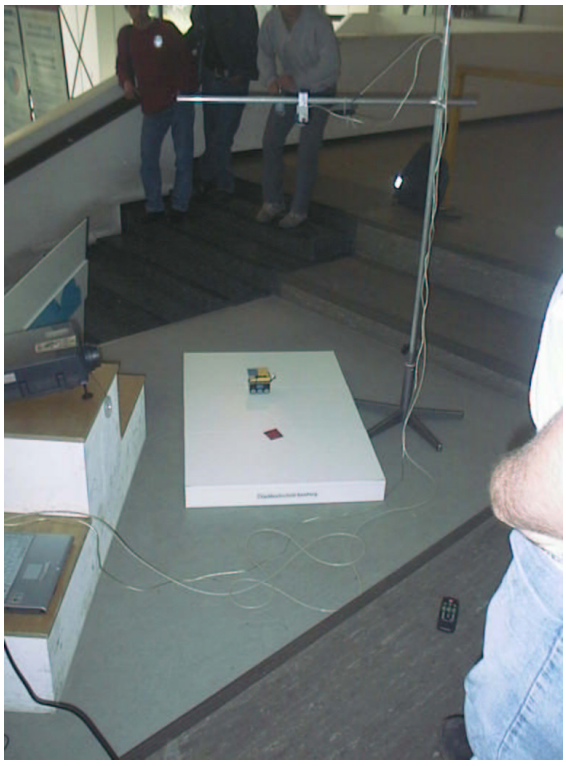

Abbildung C.11: Robot Contest der HAW Hamburg 7

Abbildung C.12: Robot Contest der HAW Hamburg 8

Anhang D

Inhalt der CD–ROM

Die unten stehenden Angaben gehen davon aus, daß der Laufwerksbuchstabe H: das CD–ROM–Laufwerk kennzeichnet.

`H:\Bilder\`	Alle in dieser Diplomarbeit verwendeten Bilder.
`H:\CATCH\`	Das Programm “CATCH” als ausführbare Datei.
`H:\NQC\`	Das eingesetzte Programm des RCX als NQC Datei.
`H:\PDF\`	Diese Diplomarbeit als “Portable Document Format” Datei.
`H:\Projekt\`	Alle Dateien des Microsoft Visual C++ 6.0 Projektes “CATCH”.

Versicherung über Selbständigkeit

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

Ort, Datum	Unterschrift des Studenten

[Newton]
[RSNewton]
[Proudfoot]
[Andersen]
[LEGO]
[von Luck]
[Krabat]
[Pioneer]
[Wischmann u. a.]
[Martin]
[Martin u. a.]
[LUGNews]
[LUGNET]
[SunJDK]
[PLD]
[Logitech 2000]
[LTQCSDK]
[RoboCup]
[LEGO 1998]
[LeJOS 2001]
[Solorzano 2000]
[Mader]
[Bantz]
[VCTech 2001]
[VfW2WDM 1998],
[WDM 1998]
[LegOS]
[Hempel]
[Baum]
[Balzerowski]
[Gasperi]
[Kunze 1994]
[Nilsson 1998]
[Haberäcker 1995]
[Schlicht 1995]
[Lamport 1995]
[Young 1997]
[Richter 1997]
[Ezzel und Blaney 1997]
[Neumann 2000]
[Baum 2000]

[Baum u. a. 2000]
[Kopp 1997]
[Bässmann und Kreyss 1998]
[Jähne 1997]
[DUDEN 1994]