Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Entwicklung eines Objekttrackers für Embedded Systems zur Steuerung mobiler Roboter

vorgelegt von
Lars Brandt
am 27. Juli 2005

Studiengang Softwaretechnik

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Gunther Klemke

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Lars Brandt

Entwicklung eines Objekttrackers für Embedded Systems zur Steuerung mobiler Roboter

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Softwaretechnik

am Fachbereich Elektrotechnik und Informatik

der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Gunther Klemke

Abgegeben am 27. Juli 2005

Lars Brandt

Thema der Diplomarbeit

Entwicklung eines Objekttrackers für Embedded Systems zur Steuerung mobiler Roboter

Stichworte

Embedded Systems, mobile Roboter, Motion tracking, Object tracking, V4L2, Bildverarbeitung

Kurzzusammenfassung

Mobile Roboter bestehen aus mehreren Komponenten, darunter Steuerung, Aktoren und Sensoren. In dieser Diplomarbeit wird ausgehend von der Laborumgebung des RoboLabs der HAW-Hamburg ein komplexer Sensor als Serviceschicht entworfen und implementiert.

Er basiert auf einer an einem Embedded System angeschlossenen Kamera und ist in der Lage, unterschiedlich farbig markierte Objekte in einer künstlichen Umgebung unter den Rahmenbedingungen der weichen Echtzeit zu erkennen und deren Position im Kamerabild zurückzuliefern. Die Spezialisierung der Fähigkeit ist die in dieser Arbeit definierte Zielsetzung. Es geht dabei um die Erkennung von eingefärbten Toren auf der Spielfläche eines Roboterfußballfeldes .

Als Beweis der Funktionsfähigkeit wird ein Motiontracker auf Basis der Serviceschicht entworfen und betrieben.

Lars Brandt

Title of paper

Development of an object tracker for embedded systems used for the controlling of mobile robots

Keywords

Embedded system, mobile robots, motion tracking, object tracking, V4L2, image processing

Abstract

Mobile robots consist of several components, among them control, actuators and sensors.

They are based on a camera attached to an embedded system and are able to recognise objects marked with different colours, which are set in an artificial environment under the basic conditions of soft real time, and to return their position in the camera picture.

In this thesis (diploma), a complex sensor is sketched and implemented as a service layer, based on the laboratory environment used in the robot laboratories of the HAW-Hamburg.

This work is intended to specialise the abilities of mobile robots. It deals with the recognition of a robot football pitch.

As proof of its operability, a motion tracker is sketched and operated on the basis of the service layer.

Danksagung

Danken möchte ich meinen Professor Prof. Dr. Kai von Luck, der mir stets mit Rat und Tat zur Seite stand und sich für 'seine' Studen sehr engagiert. In meinen Augen ist das keine Selbstverständlichkeit.

Danken möchte ich weiterhin meiner Familie und meinen Freunden für ihr Verständnis, mich mehrere Monate nicht oder beinahe gar nicht gesehen zu haben.

Meiner Freundin Chrissy möchte ich dafür danken, dass sie mir in dieser schwierigen Zeit zur Seite stand, wo sie nur konnte und mit mir litt.

Bedanken möchte ich mich auch bei meinen Arbeitskollegen, insbesondere bei Peter Thielmann für die moralische Unterstützung und Hilfe beim Korrekturlesen und bei Sönke Wiech, der mir von der joblichen Seite den Weg freihielt.

Inhaltsverzeichnis

1 Einleitung

Wir leben in einer Zeit, in der Roboter immer stärker in das Licht der Öffentlichkeit rücken. Langsam und zaghaft wagen sie die ersten Schritte auf den Consumermarkt. Damit ist nicht nur Sonys Aibo gemeint, sondern auch die ersten funktionierenden Haushaltshilfen zum Staubsaugen, Rasenmähen oder industriellem Fensterputzen großer Glasfassaden.

1.1.1: Roomba, Quelle: (iRobot)

1.1.2: Aibo, Quelle: (Sony 2005)

Dies macht aber auch deutlich, dass die Träume vom Beginn der KI nicht haltbar sind. Die humanoiden Gesellen mit menschenähnlicher Intelligenz, die uns die perfekten, kompetenten und zuverlässigen Diener sind, wird es auch auf absehbare Zeit nicht geben.

Die Forschung hat sich auf eine Politik der kleinen Schritte zurückgezogen. Grundlagenforschung dominiert noch immer das Feld, doch das Objekt der Forschung sind nicht die endlosen Fertigungsstraßen mit ihren eisernen, aber stupiden Arbeitersklaven. Interessant sind vielmehr die mobilen, intelligenten und selbstständigen Helfer, die spezialisierte Aufgaben übernehmen.

Auch die diversen Meisterschaften des Roboterfußballs gewinnen zunehmend an Zuspruch. Zwar noch immer als Spielerei aus dem Elfenbeinturm der Forschung verschrien, so sind sie doch gute Testfelder neuer Technologien. Mittlerweile finden immer weitere Kategorien Eingang in die Meisterschaften, so dass 'Die Fußball-WM für Roboter' als Umschreibung für den RoboCup längst nicht mehr ausreicht. Robotertänze sprechen auch das nicht-fachliche Publikum an. Der zentrale Bestandteil ist zwar noch immer der Fußball, aber auch Bergungsroboter versuchen in einem aufgebauten Parcour ihr Glück. Sie haben auch eine deutlicher

sichtbare Nähe zur Realität, da Roboter unter für Menschen schwierigen Bedingungen ein wichtiges praktisches Einsatzfeld sind.

Die meisten Menschen denken bei dem Wort 'Roboter' noch immer an die stereotypen und nimmermüden Fließbandarbeiter. Von ihnen gibt es zwei Variationen. Beiden gemein ist ein festes abzuarbeitendes Programm. Der Unterschied zwischen ihnen besteht darin, dass die flexibleren unter ihnen mit Sensoren ausgestattet werden, um auf Veränderungen der Umwelt reagieren zu können. Toleranzen in den zu bearbeitenden Werkstücken wären ein Beispiel für eine (in sehr engen Grenzen) variable Umwelt. Die sprichwörtliche Flexibilität dieser Maschinen bezieht sich auf den einfachen Austausch von Programmen und damit der einfachen Anpassbarkeit auf andere Arbeitsschritte.

Eine weitere Besonderheit dieser Maschinen ist sehr wichtig: Sie sind ortsgebunden und können sich nicht fortbewegen. Ihre Welt liegt in einer Halbsphäre um sie herum und kann über Abstand, Winkel und Höhe exakt definiert werden. Autonome Roboter hingegen müssen mit ihrer Umwelt interagieren, haben also das zusätzliche Problem, wissen zu müssen, wo sie sich zur Zeit befinden.

Sie verwenden dazu diverse unterschiedlich komplexe Sensoren, sind aber genau dadurch auf eine stark künstliche Umwelt angewiesen. Diese Abhängigkeit ihrer Szene beschränkt sie sehr stark in ihren Möglichkeiten. Ihre Umwelt muss Merkmale haben, an denen sich die Roboter orientieren können. Eine für uns Menschen natürliche Umgebung besitzt diese Merkmale nicht. Aus einem einfachen Grund:

Wir können sehen!

Die Frage ist: Können wir nicht auch einfach dem Roboter durch das Sehen seine (unsere) Umwelt mitteilen? Mit all den Vorteilen?

Das wichtige Wort in der Frage ist 'einfach' und die Antwort ist somit 'Nein'. Sehen ist eine Wissenschaft für sich, eine der ältesten Träume der KI und Informatik, übt aber trotzdem oder vielleicht auch gerade deshalb eine ungeheure Faszination aus.

1.1 Motivation

Auf dem Spielfeld der Fußball-Roboter existieren derzeit nur zwei Kategorien: Die kompletten und aufwendigen Eigenentwicklungen und die bereits fertigen Produkte, an denen nichts geändert wird, wie es bei den bereits erwähnten Aibos der Fall ist.

Sieht man sich auf kommerzieller Ebene um, entdeckt man drei Geschäftsfelder für Roboter: Auf der einen Seite die bereits erwähnten Haushaltshilfen, auf einer anderen Seite Spielzeugroboter und Baukästen (noDNA), (iRobot). Die letzte Seite repräsentieren die etablierten Industrieroboter zum Montieren, Schweißen, Heben und sonstigen Bearbeiten von Materialien.

Zum Einstieg fehlt ein tragfähiges Grundgerüst, eine Entwicklungsumgebung für mobile Roboter mit Industriequalität, aber Spielzeugcharakter als Experimentierumgebung. So etwas würde sich als modulares System darstellen, basierend auf kostengünstigen und robusten Standardkomponenten, welche ausgetauscht, ersetzt und verbessert werden können. Das beschränkt sich dabei nicht nur auf die Hardware-, sondern bezieht auch die Softwareebene mit ein.

Wie man es aus der PC-Welt kennt, hängen Hardware und Softwarekomponenten oft untrennbar zusammen. Als Beispiel seien Grafikkarten und ihre zugehörigen Treiber angeführt.

Es wäre somit insbesondere beim Einstieg sehr hilfreich, Ideen und Zeit in die strategischen Komponenten und nicht in das Grundgerüst stecken zu müssen. Auf diese Weise wird dem Robbiteam die $n+1$ - te Entwicklung von Entfernungs-, Berührungs- und sonstigen Sensoren erspart.

Zwar gibt es bereits fertige Komponenten wie das Cognachrome Vision System des Newton Research Labs (Newton), unter anderem verwendet auf dem Pioneer 1 Roboter, hergestellt von ActivMedia Robots Mobile Robots (ActivMedia), doch haben all diese Komponenten auch ihre spezifischen Vor-und Nachteile wie mangelnde Flexibilität und sind somit nur beschränkt einsetzbar.

1.2 Zielsetzung

Diese Diplomarbeit setzt sich zum Ziel, einen komplexen Sensor für die spezifisch angepasste Welt im RoboLab der HAW-Hamburg zu entwerfen. Spätere Benutzer sollen so die Möglichkeit haben, mit dieser vordefinierten Welt und deren speziellen Bedingungen zu interagieren.

Abbildung 1.1: RCube, Quelle: (Boersch b)

Die Aufgabe ist es, eine Objekterkennung auf Farbebene, also ein spezialisiertes Bildverstehen[1] zu entwickeln und zu implementieren. Wesentlicher Aspekt ist dabei die weiche Echtzeitfähigkeit, also eine vorgegebene Antwortzeit, die es einzuhalten gilt, allerdings ohne dass es katastrophale Folgen hat, wenn sie überschritten wird. Die Implementierung soll auf dem RCube erfolgen, einer Embedded Plattform mit integrierter Videoverarbeitung, aber für diese Aufgabe sehr schwachbrüstigen Prozessors. Das Endergebnis stellt eine Serviceschicht dar, die von Entwicklern höherer Schichten verwendet werden kann.

Das Erreichen des Ziel soll über eine Demonstrationsanwendung bestätigt werden, in der alle genannten Aspekte zum Tragen kommen.

Ebenfalls als Zielsetzung definiert ist ein nicht direkt zu der Serviceschicht gehörendes, aber ebenfalls zum 'Roboterbaukasten' zugehöriges Werkzeug. Dabei handelt es sich um eine mechanische Entkopplung der Entwicklungsmaschine (PC) mit dem Roboter (RCube).

[1] Bildverstehen, vergl. (Görz 2003)
Bildverstehen ist das Teilgebiet der KI, das sich mit der Analyse und Interpretation von visuellen Informationen befasst. Die zentralen Fragen dieses Gebietes sind: Wie kann man das Sehvermögen biologischer Systeme durch Berechnungstheorien erklären und maschinell nachbilden? Wie kann man maschinelle Sehsysteme zur Lösung technischer Probleme einsetzten?

1.3 Gliederung der Arbeit

Das Kapitel 2 beschäftigt sich mit der bereits vorhandenen Umgebung und ihren Möglichkeiten bzw. Einschränkungen.

Im Konzept (Kapitel 3) werden die nötigen Grundlagen zum Verständnis des Verfahrens aufgeführt. Auch der Algorithmus selbst wird an dieser Stelle beschrieben.

Design und Realisierung (Kapitel 4) teilen sich in vier Unterkapitel. Der Prototyp (Kapitel 4.1) beschreibt eine Workbench zur Entwicklung von Bildverarbeitungsverfahren hinsichtlich der Qualität des eingesetzten Algorithmus. Der Laufzeit wird dabei wenig Bedeutung beigemessen, da sie nicht oder nur schwer auf das jeweilige Zielsystem übertragbar ist.

Das Kapitel 4.2, die Tools, sind eine Sammlung von Algorithmen zur Informationsextraktion aus Bildern, zusammengefasst in einem Programm. Es ist ein Hilfsmittel, verständlicher zu machen, wie die Kamera bzw. die Bildverarbeitung ihre Umwelt sieht oder sehen kann.

Die RCube basierte Serviceschicht im Kapitel 4.4 beschreibt das wesentliche Ziel dieser Arbeit. Es finden sich an dieser Stelle die Beschreibung, Design und Implementierung der Erkennung farbiger Objekte.

Um zu zeigen, dass die Serviceschicht einsatzfähig ist, entstand eine Anwendung als 'Proof Of Concept', welche farbige Objekte mit Hilfe von Modellbauservos auf zwei Achsen verfolgen kann. Informationen über diesen Teil der Diplomarbeit finden sich im Kapitel 4.5.

2 Rahmenbedingungen

In den folgenden Unterkapiteln werden die Rahmenbedingungen dieser Diplomarbeit näher erläutert. Es wird dabei nur so weit wie zum Verständnis dieser Arbeit nötig auf die einzelnen Aspekte eingegangen.

2.1 Labor

Der Hauptarbeitsplatz war das RoboLab der HAW-Hamburg (HAW-Hamburg). Es existieren sowohl eine kleine Werkstatt als auch ein größerer Raum mit mehreren PCs unterschiedlicher Ausstattung. Eigene Festplatten als Einschub oder über USB beinhalten Daten und Betriebssystem und entkoppeln so von dem Wirken anderer Kommillitonen mit dem Rechner.

Wesentliches Merkmal des Labs ist die Spielfläche für die Roboter (siehe Abbildung 2.1).

2.2 Software

Da die Anleitung des RCubes (Boersch b) auf Linux bezogen war und um die Einarbeitungszeit nicht unnötig zu verlängern, entschied ich mich für ein im RoboLab existierendes Linux, nämlich SuSE 9.0 (Novell). Diese spezielle Version ist in der Lage, über USB zu booten (Jagdmann und Stickel), kann also auf einer externen USB-Festplatte gehalten werden.

Als Editor für Java kam Eclipse (Eclipse) zum Einsatz, dazu das Java Media Framework (Sun b) und die Java Laufzeitumgebung (Sun a). Zum Schreiben des C-Codes verwendete ich anfänglich jEdit (jEdit), stieg dann aber auf den in der Linux-Distribution vorhandenen Conqueror um.

Abbildung 2.1: Spielfeld

2.3 Roboter

Als Basis dieses Projektes dient die bereits vorhandene und von Michael Manger (Manger 2003), (Manger 2004) entwickelte Roboter-Plattform. Sie soll allerdings aus technischen Gründen im Zuge einer parallel zu meiner laufenden Diplomarbeit von Michael Ziener (Ziener 2005) ersetzt werden.

Diese Plattform stellt den Körper, den Aktor und Sensor des RCubes dar, seine einzige Möglichkeit, die reale Welt auszuwerten und auf sie zu reagieren.

Fortbewegen kann er sich durch drei omnidirektionale Räder, welche durch jeweils einen Elektromotor angetrieben werden. Die ursprüngliche Variante verwendete Modellbauservos, die sich allerdigs als zu langsam und drehmomentschwach entpuppten. Die neue Plattform ist modularer ausgelegt und verwendet kleine Elektromotoren aus dem Modellbaubereich mit angeflanschtem Getriebe.

Da die Räder omnidirektional und im 120° Winkel angeordnet sind, ist eine Fortbewegung in jedem beliebigem Vektor möglich, inklusive Drehen auf der Stelle und Drehen während der Fahrt. Üblicherweise werden solche Räder bei Fließbändern verwendet, in denen zum Beispiel Pakete seitwärts vom Band oder auf das Band geschoben werden.

Die derzeitige Torerkennung läuft über modulierte Infrarotstrahlen, die von einer kleinen elektronischen Schaltung mit aufgelöteten IR-Dioden produziert und abgestrahlt werden. Die Trägerfrequenz beträgt 40kHz, die Modulationsfrequenzen sind 100 und 125Hz für jeweils ein Tor.

Zur Zeit wird der Ball über einen drehbaren Infrarotsensor im Roboter erkannt. Der Ball selbst ist eine durchsichtige Plastikkugel mit einem Akku und vielen Infrarot-LEDs.

Als Kollisionserkennung kommen elektromechanische Bumper zum Einsatz. Bei einem Kontakt mit der Bande oder einem anderen Roboter wird ein Mikrotaster betätigt.

Um die Entfernung zu den Wänden in der Umgebung feststellen zu können, ohne direkt gegen sie zu stoßen, werden drehbare Sharp GP2D12-Sensoren (Sharp) verwendet. Da es Distanzsensoren sind, geben sie nicht einfach eine 'erkannt'-/'nicht erkannt'-Antwort, sondern sind in der Lage, den Abstand zum erkannten Gegenstand zurückzuliefern.

2.4 RCube

Der RCube ist das zentrale Element dieser Diplomarbeit. Auf ihm soll die zu entwickelnde Serviceschicht ablaufen. Der RCube besteht aus drei Teilen, denen jeweils ein Kapitel gewidmet werden soll. Um auf die Außenwelt reagieren zu können, werden elektronische Ein- und

Ausgänge benötigt. Diese liefert das AkSen-Board (Boersch a), welches als oberste Plattform auf dem RCube verbaut ist. Eine Etage tiefer, nämlich in der Mitte, sitzt das VIO-Board. Dieses stellt die Funktionalität der Videoeingabe und -ausgabe zur Verfügung. Als letzten und untersten Teil des RCubes findet sich das LART-Board (Bakker u. a.), das Gehirn des RCubes. Auf ihm befinden sich Prozessor, Speicher und einige Schnittstellen.

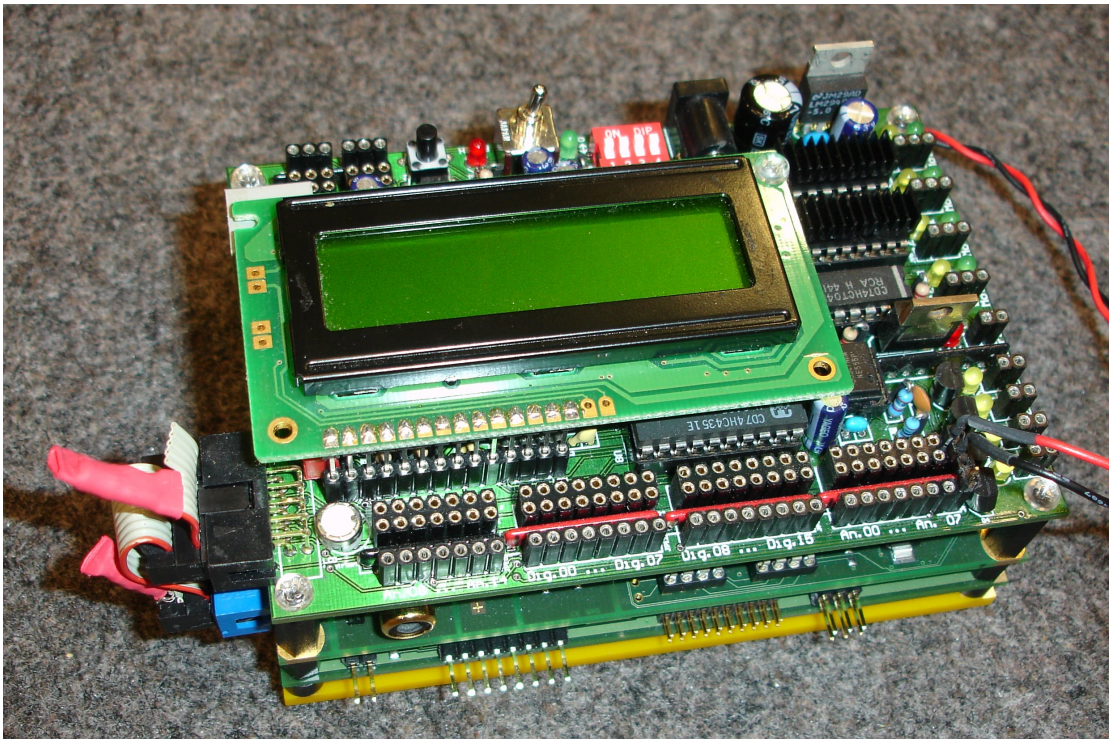


Abbildung 2.2: RCube

AKSen

AkSen steht für **Ak**tor **Sen**sor. Das Board ist auch allein funktionsfähig und kann für kleinere Anwendungen verwendet werden. In diesem Fall wird es aber im RCube zusammen mit dem über einen CAN-Bus gekoppeltem LART- und VIO-Board (Boersch c) verwendet. Letztendlich ist dieses Board die einzige Möglichkeit für das LART-Board, mit der Aussenwelt zu kommunizieren.

Angetrieben wird das Board von einem SAB80C515A (Xepox), welcher extern mit 12MHz getaktet wird. Dieser Prozessor kann von außen geflashed werden. Dazu dient ein RJ45 Port auf dem Board, welcher zwar auch für Ethernet-Twisted-Pair-Netzwerke verwendet wird, in diesem Fall aber nur eine simple serielle Schnittstelle nach RS232-Manier zur Verfügung stellt.

Das Board verfügt über eine reichhaltige Anzahl von Ein- und Ausgängen:

- 16 digitale Ein-/Ausgänge
- 15 analoge Eingänge
- ein alphanumerisches LCD-Display mit 2*8 Zeichen
- ein modulierbarer IR-Ausgang
- 4 Lämpchen-Treiber (max. 600mA)
- 3 Servo-Ausgänge

Zwei weitere Schnittstellen finden sich auf dem Board:

- die bereits beschrieben RJ45-Schnittstelle zum Flashen
- CAN-Bus zur Kommunikation mit Sensoren, anderen AkSens

VIO

Das VIO-Board (**V**ideo **I**nput **O**utput) ist in der Lage, Videosignale von externen Quellen zu verarbeiten und dem LART-Board zugänglich zu machen. Zudem besitzt es eine Videoausgabe, um beispielsweise Debug-Informationen auszugeben.

Es besitzt vier Video-Eingänge, die nicht gleichzeitig verwendet werden können. Das bedeutet, dass dem Video A/D-Wandler ein analoger Umschalter vorgeschaltet ist, welcher die vier Video-Eingänge auf den D/A-Wandlers multipext. Der Multiplexer und der A/D-Wandler befinden sich auf einem Chip mit der Bezeichnung SAA7113H.

Wie in der Beschreibung zum VIO-Board nachzulesen ist, muss nach jedem Umschalten ca. 40ms gewartet werden, bevor ein Bild gesampled wird. Hält man sich nicht an diese Vorgabe, so entstehen defekte Bilder.

Die komplette logische Verarbeitung übernimmt ein FPGA[1] mit der Typenbezeichnung EP1K50QC208-2. Er erhält seine Daten aus einem externen EEPROM, so dass ein Firmwareupdate sehr einfach und schnell von Statten geht.

Das VIO-Board verfügt über keinen eigenen Arbeitsspeicher, so dass die Daten über den High-Speed-Connector des LART-Boards in dessen Hauptspeicher injiziert werden müssen. Die Videoausgabe greift ebenfalls über den Connector auf den Hauptspeicher des LARTs zu. Der Video-D/A-Wandler verfügt über mehrere Ausgänge (insgesamt vier), von denen allerdings nur ein einziger verwendet wird und herausgeführt ist. Verwendet wird der ADV7171. Um Energie zu sparen, kann dieser Ausgang abgeschaltet werden. Im Linux-Betriebssystem des LART-Boards ist verankert, dass der Ausgang ca. 10 Minuten nach dem Booten ausgeschaltet wird, damit im mobilen Betrieb die Energieversorgung nicht unnötig belastet wird.

Der Video-Ausgang liefert ein PAL-Signal, die Video-Eingänge benötigen ebenfalls ein PAL-Signal. Allerdings soll auch NTSC möglich sein.

Somit kann eine beliebige Videoquelle, sei es eine Kamera, ein Videorecorder oder gar ein anderes VIO-Board, als Quelle dienen. Aus dem gleichen Grund können also Videorecorder, Videomonitor, Fernseher mit Videoeingang, Capturekarte im PC oder eben auch ein anderes VIO-Board als Senke dienen.

Das Board ist mit einem SJA1000 CAN-BUS-Controller ausgerüstet. Er wird von dem LART-Board verwendet, um mit dem AkSen-Board zu kommunizieren.

Es soll übrigens Anwendungsmöglichkeiten geben, bei denen das VIO-Board eigenständig arbeiten kann. Werden LART- und VIO-Board zusammen verwendet, werden ein oder mehrere Verfahren den einlaufenden Videostrom verarbeiten und eine neue Ausgabe generieren.

[1] *FPGA (vergl. Wikipedia FPGA)*

Ein FPGA (**F**ield **P**rogrammable **G**ate **A**rray) ist ein frei programmierbarer Logikschaltkreis. Es besteht aus einer Matrix-Struktur aus konfigurierbaren Logikblöcken, ca. neun Metalllayern zur Verbindung der CLBs (Verbindungsebene), Ein- und Ausgabeblöcke (I/O-Blöcke) zur Impedanz- und Logikanpassung der I/O-Pins des Gehäuses (die für die Verbindung zur Außenwelt notwendig sind). Die Logikblöcke werden auch als CLB bezeichnet (**C**onfigurable **L**ogic **B**locks).

Des weiteren existieren SRAM-Blöcke, sogenannte BRAM-Blöcke, für die Speicherung von Daten. Manche Anbieter stellen zusätzlich PLLs (**P**hase **L**ocked **L**oop), DLLs (**D**elay **L**ocked **L**oop), Taktaufbereitungen (DCM: **D**igital **C**lock **M**anager) zur Verfügung.

Da die CLBs der FPGAs einen synchronen Takt erhalten müssen, um unkalkulierbare Laufzeitunterschiede zu vermeiden (synchrones Design), sind Taktverteilerbäume notwendig. In der Regel werden sowohl Multiplizierer oder allgemeinere Recheneinheiten, als auch komplette Prozessoren in den FPGA integriert.

Es gibt sowohl reprogrammierbare flüchtige (SRAM-basierte), nicht flüchtige reprogrammierbare (Flash basierend)) als auch nur einmal programmierbare (OTP, AntiFusetechnik) FPGAs. RAM-basierte FPGAs laden ihre Hardwarekonfiguration nach Anlegen der Betriebsspannung(en) in der Bootphase aus externen Speichern, z.B. FlashROM.

Ist das Verfahren allerdings so einfach, dass man eine Schaltung entwickeln kann, die in das FPGA passt, so kann man auf das LART-Board verzichten.

LART

Das LART-Board ist ein mit einem ARM-Derivat - in diesem Fall ein StrongARM von Digital (Digital) - bestückt. Dieser RISC-Prozessor wurde auf eine hohe Rechnerleistung in Bezug auf den Energiekonsum ausgelegt.

Der interne Takt des Prozessors liegt bei 220MHz und sorgt für eine Performance von über 200MIPS bei einem Energieverbrauch von ca. einem Watt.

Die Schaltpläne sind als OpenHardware frei verfügbar. Das FlashROM, ursprünglich mit einer Größe von 4MByte, wurde auf 8MByte erweitert, der Hauptspeicher mit 32MByte so belassen.

Die Spannungsversorgung erfolgt über einen Adapter vom AkSen-Board. Damit liegt die Betriebsspannung bei 5V. Der mögliche Spannungsbereich ist aber recht breit, da ein interner Schaltregler für eine korrekte interne Betriebsspannung von 3,3V sorgt. Die externe Betriebsspannung darf in einem Bereich von 3,5 bis 16 V liegen. Allerdings wird davor gewarnt, sie über 7,5V zu halten, da Teile des Spannungswandlers nicht für eine höhere Spannung ausgelegt sind.

Glücklicherweise ist das RESET-Signal herausgeführt, da so ein Warmstart, welcher beim Entwickeln häufiger vorkommt, bequem von außen iniziiert werden kann. Dazu reicht ein Taster, zur Not aber auch ein beliebiger leitender Gegenstand. Damit werden alle aktuellen Änderungen gelöscht, der Arbeitsspeicher und das virtuelle Dateisystem werden wieder mit Daten aus dem Flash überschrieben.

Um maximal zwei Konsolen zum Betriebssystem zu bekommen, sind zwei serielle RS232-Schnittstellen herausgeführt. Die erste (TTYs0) hat eine Datenrate von 9600Baud und gibt die Bootmeldungen aus, die zweite (TTYs1) ist mit 115200Baud schneller und bietet sich zum Übertragen von Daten oder schnellen Loggings an.

Betriebssystem

Auf dem LART-Board findet Linux seine Verwendung. Es wird ein Standard-Kernel eingesetzt, gepatcht mit einer Erweiterung von Russel King. Sie enthält Änderungen und Erweiterungen für die ARM-Architektur, welche bisher nicht in den Kernel eingeflosssen sind.

Der Bootloader 'blob' kopiert das Kernelimage aus dem FlashROM in den RAM-Bereich und startet es dort.

Beim Booten entpackt der Kernel eine ebenfalls im FlashROM befindliche komprimierte RAMDisk. Sie enthält das Dateisystem, Dateien, die auf das LART-Board übertragen werden existieren also nur im RAM und werden wie bereits erwähnt beim nächsten Reset gelöscht.

CAN-Bus

An dieser Stelle gibt es eine kurze Einführung in den CAN-Bus, da er in einer späteren Anwendung benötigt wird (siehe Kapitel 4.5, Proof Of Concept).

Der CAN-Bus wurde 1983 von Bosch (Bosch) für den Einsatz in Kraftfahrzeugen entwickelt, um eine störungsfreie Übertragung von Daten zu gewährleisten, die der Echtzeitanforderung unterliegen. Nicht zuletzt durch die geringen Kosten hat er sich jedoch auch in der Automatisierungstechnik, insbesondere bei Textilmaschinen, Aufzugsteuerungen und Landmaschinen durchgesetzt.

Wie der Name bereits deutlich macht, ist die Struktur busförmig, d.h. alle Komponenten werden über kurze Stichleitung an einer gemeinsamen Datenleitung angeschlossen. Aus diesem Grunde ist es problemlos möglich, Komponenten jederzeit zu entfernen oder hinzuzufügen. Die Regelung des Datenflusses auf dem Bus über ein Protokoll soll dafür sorgen, dass auch Geräte unterschiedlicher Hersteller problemlos miteinander kommunizieren können.

Physikalische Basis Die physikalischen Spezifikationen des CAN-Busses sind in ISO 11898 (ISO) definiert. Zur Umsetzung stehen recht viele ICs zur Verfügung, z.B. der PCA82C250 (Philips b) von Phillips (Philips a) oder auch der in diesem Projekt verwendete SJA1000 (Philips c), ebenfalls von Phillips.

Die Störsicherheit wird dadurch erreicht, dass ein Bit auf zwei Adern gegensinnig übertragen wird, eine sogenannte differenzielle Übertragung. Magnetische und elektrische Störfelder wirken sich auf die Signale beider Adern aus, so dass es keinen Unterschied macht, ob sich auf beiden Adern die Spannung erhöht oder nicht (Gleichtaktunterdrückung). Wichtig ist nur die Spannungsdifferenz der beiden Adern zueinander. Die Adern CAN-High und CAN-Low enthalten genau diese beiden eben beschriebenen Signale.

Als *dominalen* Zustand bezeichnet man auf dem CAN-Bus eine Pegeldifferenz zwischen CAN-High und CAN-Low von 3,5V. Bei dem *rezessiven* Zustand beträgt die Pegeldifferenz weniger als 1,5V

Steckerbelegung Als Steckverbindung hat sich der 9-polige SUB-D-Steckverbinder durchgesetzt, auch bekannt durch die serielle Schnittstelle am PC. Benötigt werden als Minimum drei belegte Pins, nämlich CAN-High, CAN-Low und Ground als Bezugspunkt der beiden Signale. Eine Verwendung von geschirmten Leitungen ist nicht vorgeschrieben, bei größeren Entfernungen empfiehlt sich allerdings ein Verdrillen der Adern.

Interessant ist die von der CiA (**C**an **i**n **A**utomatation) (CiA) definierte Doppelpfostenleiste, welche auch beim RCube eingesetzt wird. Die CiA nennt ihn 'multipole connector'. Die Pinbelegung findet sich im Anhang unter 6.1

Bitrate und Leitungslängen Die maximale Geschwindigkeit des CAN-Busses beträgt 1Mbit/s. Voraussetzung ist, dass alle am Bus angeschlossenen Geräte das Signal gleichzeitig verarbeiten können, die Laufzeit des Signals beschränkt hier den Bustakt. Im Anhang (6.1) findet sich eine Tabelle mit der Aufschlüsselung von Leitungslänge und maximaler Übertragungsrate.

Busterminierung Eine Busterminierung ist aus zwei Gründen nötig:

1. Sie arbeitet als kombinierter Pullup/ -down Widerstand, um die Adern mit Spannung zu versorgen
2. Vermeidung von Reflexionen am Busende und damit die Vermeidung der Zerstörung von Daten

Bei kurzen Bussen ist es legitim, nur an einem Ende des Busses zu terminieren. Besser ist es allerdings, immer beide Seiten abzuschließen. Die Widerstände müssen einen Wert von 120Ω haben.

Übertragung und Kollision Da der CAN-Bus seriell überträgt und jeder Knoten ohne Aufforderung senden kann, wird es unweigerlich zu Kollisionen kommen, wenn mindestens zwei Stationen gleichzeitig senden. Die Zeit auf dem Bus ist in Schlitze eingeteilt, während eines Schlitzes kann jedoch nur ein Bit übertragen werden.

Um eine bestimmte Reaktionszeit einhalten zu können, hat jede Nachricht einen Identifier. Die Kollisionserkennung läuft dabei während des Sendens mit und erkennt, ob ein anderer Knoten die Leitung auf einen dominanten Pegel gesetzt hat. Dieser setzt sich auf jeden Fall durch, so dass Echtzeit nur für Nachrichten mit höchster Priorität garantiert werden kann.

Kamera

Die zum RCube mitgelieferte Farbkamera erreicht nicht die volle PAL-Auflösung (720*576). Der Helligkeits- und Weißabgleich geschieht automatisch, Einfluss kann auf diese beiden Parameter nicht genommen werden.

Die Lichtempfindlichkeit beträgt 0,5LUX, normales Tageslicht ist somit zum Betrieb ausreichend. Die Betriebsspannung muss im Bereich 5V $\pm$ 0,5V liegen. Da die Stromaufnahme 10mA beträgt, beläuft sich der Energiekonsum auf
$5V * 0,01A = 50mW$

Der Abgleich der Schärfe geschieht durch ein mit einem Gewinde versehenen Objektiv, das sich durch Drehbewegungen dem CCD-Element nähert oder sich von ihm entfernt.

3 Konzept

In den folgenden Unterkapiteln werden die Prinzipien erläutert, auf denen das spätere Design und die Implementierung aufsetzen. Das Kapitel über Farben und Farbmodelle liefert einige Grundlagen, die zum Verstehen des eigentlichen Konzeptes unabdingbar sind.

3.1 Farben und Farbmodelle

Die menschliche visuelle Wahrnehmung der Umgebung ist stark durch den Aufbau des menschlichen Auges geprägt. Die Welt, wie wir sie wahrnehmen, basiert auf der Rezeption elektromagnetischer Wellen. Genau genommen können wir drei Wellenlängen wahrnehmen, wir haben ihnen die Namen Rot, Grün und Blau gegeben.

Sämtliche von uns Menschen wahrnehmbare Farben beruhen auf einer Mischung dieser drei Wellenlängen (siehe Abbildung 3.1).

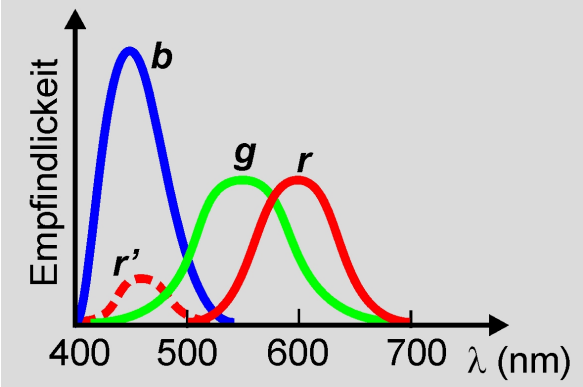


Abbildung 3.1: Spektrale Empfindlicheit des menschlichen Auges

Wie auf dem Bild zu erkennen ist, sind wir Menschen nur in der Lage, einen kleinen Teil des elektromagnetischen Spektrums zu detektieren. Bienen als Beispiel können noch weiter in Richtung kürzere Wellenlänge sehen. Diesem Spektrum wurde der Name UV (Ultraviolett) gegeben.

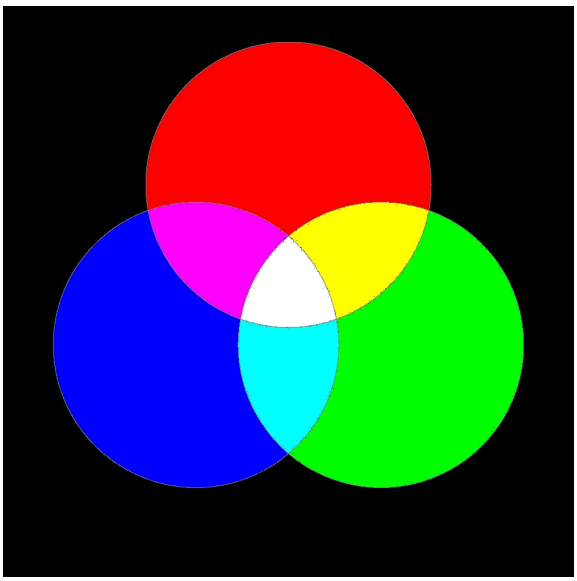
Fernbedienungen, wie sie in TV-Geräten etc. üblich sind, arbeiten in einem für Menschen zu langen Spektralbereich, dessen Wellenlänge größer als die des sichtbaren Rotes. Dieses Spektrum wurde IR (Infrarot) benannt.

Visuelle Elektronik ist auf den Menschen angepasst, Farbgeräte setzen die Farben aus eben diesen drei speziellen Wellenlängen zusammen. Das heißt, dass sich die Bildverarbeitung auf diesen Bereich beschränken muss, da nur Standardkomponenten zur Verfügung stehen.

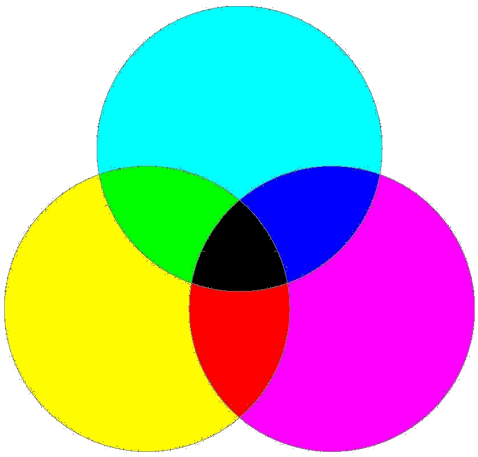
3.2 Farbmischung

Bei dem Mischen der drei Grundfarben gibt es zwei Arten:

- *Additive Farbmischung* bedeutet, dass sich die drei Grundfarben zu Weiß aufaddieren. Das geschieht beispielsweise bei Fernsehern, jeder Bildpunkt setzt sich aus drei einzelnen Bildpunkten zusammen.
- Beim Druck, sowohl industrieell als auch auf dem heimischen Tintenstrahldrucker verhält es sich etwas anders. Bei der *subtrakiven Farbmischung* absorbiert die Farbe Rot aus dem eingestrahlen Licht die Farben Grün und Blau. Eine Mischfarbe kann auf diese Weise nicht erzeugt werden. Deshalb werden die Farben Cyan, Magenta und Yellow verwendet, sie absorbieren nur je eine Wellenlänge. Rot setzt sich aus Magenta und Yellow zusammen. Magenta absorbiert Grün, Yellow absorbiert Blau, somit wird ausschließlich Rot reflektiert



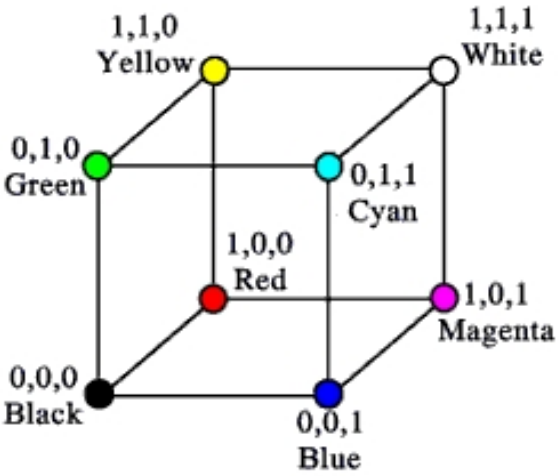
3.2.1: Additive Farbmischung



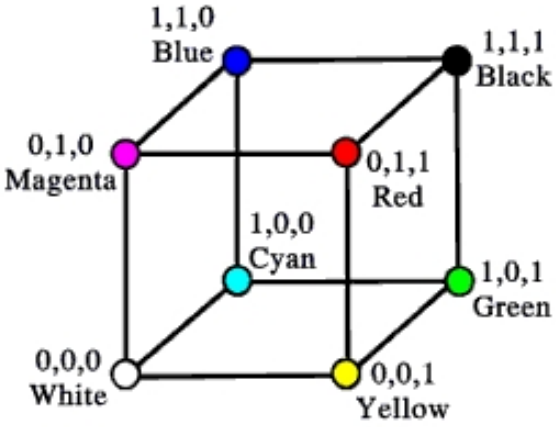
3.2.2: Subtraktive Farbmischung

Farbmodelle

Das RGB-Farbmodell lehnt sich an die additive Farbmischung an. Es stehen pro Pixel drei Kanäle für die Grundfarben zur Verfügung. Der Farbraum ist ein Würfel, eine Farbe wird durch einen Vektor in ihm definiert. Selbiges gilt für das CMY-Farbmodell, welches sich auf die subtraktive Farbmischung bezieht. Illustrationen der Farbmodelle finden sich in Abbildung 3.2.3 und 3.2.4. Die Herkunft der beiden Bilder ist (Otto).



3.2.3: Additive Farbmischung (RGB)



3.2.4: Subtraktive Farbmischung (CMY)

Das HSB- bzw. HSV-Farbmodell hingegen lehnt sich nicht an das Auge von Trichromaten[1] an. Der Informationsgehalt ist äquivalent der vorig beschriebenene Modelle, die Verteilung der Information erfolgt aber auf eine andere Weise.

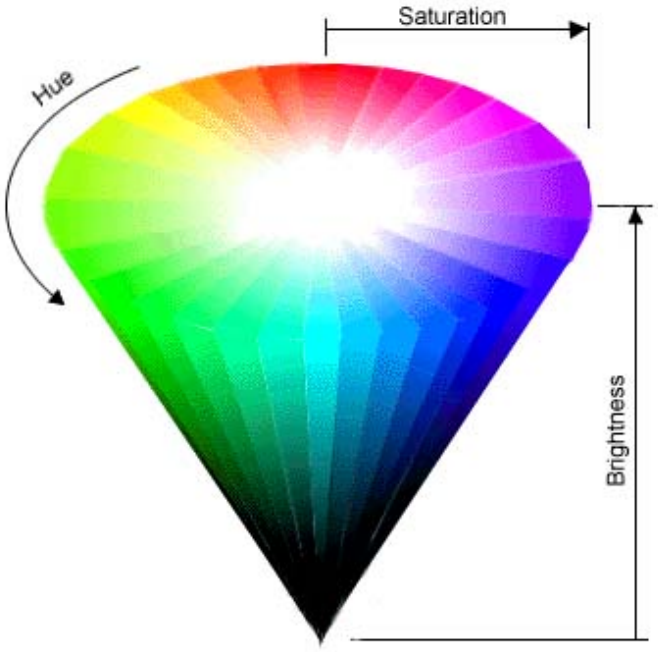
[1] *Trichromaten vgl. Wikipedia*

Trichromaten (griechisch tria chromos = drei Farben) sind Lebewesen, welche drei verschiedene Arten von Zapfen (Farbrezeptoren) in der Netzhaut haben. Es gibt drei Sorten von Zapfen mit unterschiedlichen Absorptionsmaxima: rotempfindliche L-Zapfen (L steht für Long, lange Wellenlänge), grünempfindliche M-Zapfen (M für Medium, mittlere Wellenlänge) und blauempfindliche S-Zapfen (S für Short, kurze Wellenlänge).

Aus den Messwerten der L-, M- und S-Zapfen erstellt das Gehirn ein Farbbild, wie wir es vom Alltag her kennen. Die Zapfen haben eine so geringe Empfindlichkeit, dass sie nur bei guten Lichtverhältnissen arbeiten. Bei Dämmerung oder Dunkelheit nimmt deren Funktionalität dagegen stark ab, so dass die empfindlicheren Stäbchen einspringen müssen. Da die Stäbchen jedoch nur helligkeitssensibel sind, nimmt das Farbsehen mit aufkommender Dunkelheit ab. Deshalb können Menschen nachts keine Farben mehr erkennen, und alle Katzen sind bei Dunkelheit grau. Unter den Säugetieren sind nur der Mensch und einige Altweltaffen Trichromaten.

Die Informationen werden in Farbe (Hue), Sättigung (Saturation) und Helligkeit (Brightness) aufgespalten.

Von besonderen Interesse ist, dass die Farbinformation unabhängig von Helligkeit und Sättigung existiert. Prinzipiell läßt sich so eine farbige Fläche unabhängig von der Helligkeit detektieren, da die Farbe in dem HSV-Modell als getrennte Information vorliegt. In Abbildung 3.2.5 findet sich der Farbkegel für dieses Farbmodell.



3.2.5: HSV-Farbkegel, Quelle: (Jewett)

Die Farbmodelle können mit Hilfe zweier Funktionen problemlos ineinander überführt werden:

f(rgb) = hsv

$MAX = max(R, G, B)$

$MIN = min(R, G, B)$

$$H = \begin{cases} (0 + \frac{G-B}{MAX-MIN}) \times 60, IF(R = MAX) \\ (2 + \frac{B-R}{MAX-MIN}) \times 60, IF(G = MAX) \\ (4 + \frac{R-G}{MAX-MIN}) \times 60, IF(B = MAX) \end{cases}$$

$IF(H < 0)$ H=H+360

$S = \frac{MAX-MIN}{MAX}$

$V = MAX$

Der Wert von H variiert im Beeich [0..360], S und V im Bereich [0..1].

f(hsv) = rgb

$IF(S = 0)$ $R{=}V$
$G{=}V$
$B{=}V$

$H_i = \lfloor \frac{H}{60} \rfloor$

$f = \frac{H}{60} - H_i$

$p = V \times (1 - S)$

$q = V \times (1 - (S \times f))$

$t = V \times (1 - (S \times (1 - f)))$

$IF(H_i = 0)$ $R{=}v$
$G{=}t$
$B{=}p$

$IF(H_i = 1)$ $R{=}q$
$G{=}v$
$B{=}p$

$IF(H_i = 2)$ $R{=}p$
$G{=}v$
$B{=}t$

$IF(H_i = 3)$ $R{=}p$
$G{=}q$
$B{=}v$

$IF(H_i = 4)$ $R{=}p$
$G{=}t$
$B{=}v$

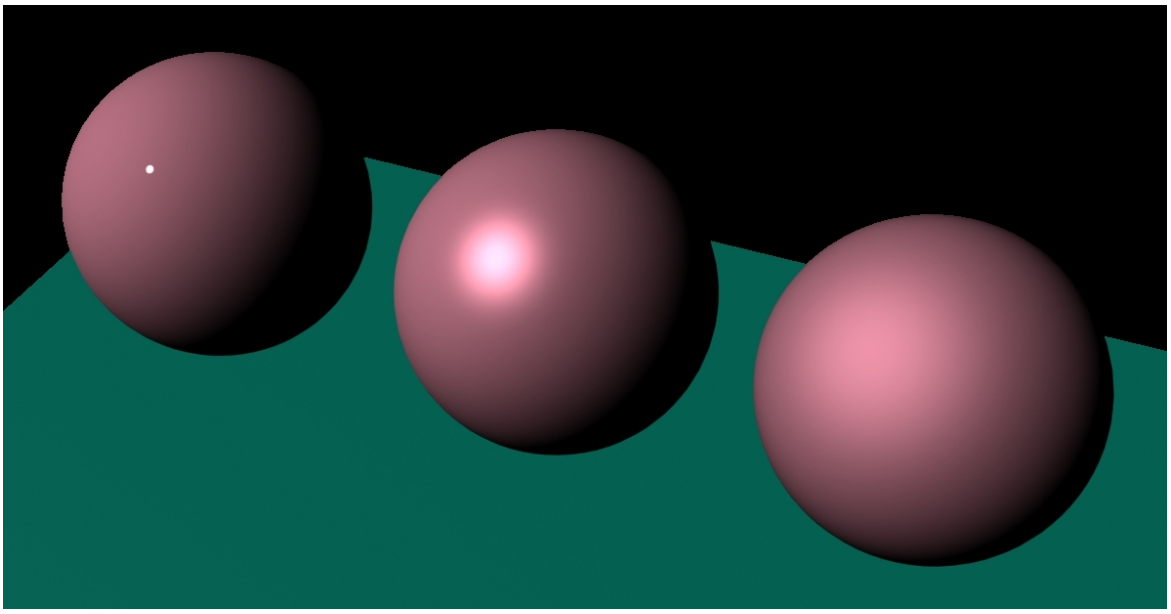
$IF(H_i = 5)$ $R{=}v$
$G{=}p$
$B{=}q$

Wie man erkennen kann, ist der Aufwand zur Umwandlung nicht über alle Maßen groß, allerdings auch nicht vollkommen unerheblich.

3.3 Oberflächen

Ein weiterer Aspekt bei der Farberkennung bezieht sich auf die Oberfläche zu erkennender Objekte. Das von einem Objekt abgestrahlte Licht setzt sich aus drei Komponenten zusammen:

1. Dem *selbst erzeugten Licht* (leuchtende Oberflächen, wie der Glühdraht von Glühlampen, floureszierendes Phosphor bei Leuchtstofflampen oder glühende Kohlenstoffpartikel, welche die Flammen von Kerzen und Lagerfeuer leuchten lassen).
2. *Diffus reflektiertes Licht*. Ein gutes Beispiel ist ein weißes Blatt Papier. Ein Lichtstrahl ohne Aufweitung, wie zum Beispiel ein gebündelter und paralleler Laserstrahl, der auf ein Blatt Papier gerichtet wird, ist als Punkt von allen Seiten aus zu sehen. Das Licht wird in alle Richtungen gestreut.
3. *Spiegelnd reflektiertes Licht*. Wie der Name schon sagt, wird das Licht im negativen Auftreffwinkel wieder abgestrahlt und kann nur von dort aus wahrgenommen werden.



3.2.6: Spiegelnde bis diffuse Reflextion (von links nach rechts)

Wichtig ist dieses Wissen insofern, um einschätzen zu können, auf welche Weise das Umgebungslicht Einfluß auf die Wahrnehmung eines Objektes haben kann. Ein Beispiel mit drei Kugeln soll den Unterschied zwischen spiegelnden und diffusen Oberflächen verständlichen. Noch erwähnt werden soll an dieser Stelle, dass es verschiedene Möglichkeiten gibt, ein Pixel zu berechnen. In der Computergrafik sind das zum einen Verfahren, die sich an

die Physik anlehnen und den Weg des Lichtes von den Lichtquellen zum Ziel (der Kamera) berechnen (Foley u. a. 1994a). Zum Anderen kann der umgekehrte Weg gegangen werden, wie es etwa beim rekursiven Raytracing (Foley u. a. 1994b) der Fall ist. Für jeden Pixel wird vom Ziel aus ein Strahl geworfen, der bis zu den Lichtquellen zurückverfolgt wird.

3.4 Algorithmus

Der zu Grunde liegende Algorithmus ist eine Farbselektion mit einer anschließenden Schwerpunktbildung. Algorithmen mit der Zielsetzung, Farbbereiche in einem Bild/Video zu detektieren, wurden zuhauf entwickelt. Der Grund für die Entwicklung eine weiteren Version ist der, dass laut Zielsetzung das Programm auf dem RCube arbeiten soll und somit nur sehr eingeschränkte Ressourcen zur Verfügung hat.

Deshalb sind Optimierungen nötig, deren Vorhandensein allerdings erst einen Sinn machen, wenn das Problem deutlich ist.

Daher folgt zunächst eine Beschreibung des Grundalgorithmus (siehe Kapitel 3.4), ohne jegliche Optimierung. Daraufhin folgt im Kapitel 3.4 die Problembeschreibung bei der Implementierung auf dem RCube .

Die Optimierungen des Grundalgorithmus (siehe Kapitel 3.4) schließen dieses Unterkapitel ab.

Grundalgorithmus

Die einfachste Variante, um einen Farbschwerpunkt in einem Bild zu erkennen, ist schnell beschrieben:

Zuerst muss ein Farbtoleranzbereich definiert werden, also ein Bereich, in dem eine Farbe liegen muss, um als korrekt erkannt zu werden. Diese Angabe erfolgt im HSV-Farbmodell, da auf diese Weise ein Farbbereich unabhängig von der Sättigung und der Helligkeit definiert werden kann. Ein Bereich für Sättigung und Helligkeit kann ebenfalls angegeben werden und macht zumindest bei der Sättigung einen Sinn, um beispielsweise schwach zu einer Farbe tendierendes Grau herauszufiltern (die Sättigung mit der Farbe ist in diesem Fall nur sehr gering).

Als nächstes läuft ein Iterator über alle Bildpunkte hinweg und wendet eine Funktion zur Markierung des jeweiligen Bildpunktes an.

Der erste Schritt ist eine Umwandlung des RGB-Farbmodells in das HSV-Farbmodell. Danach muss getestet werden, ob das zu testende Pixel in den Farbtoleranzbereich fällt. Ist dem so, wird es als 'korrekt' markiert, andernfalls als 'nicht erkannt'.

Der folgende Durchgang muss nun auf eine beliebige Weise den Schwerpunkt der erkannten Pixel berechnen. Eine Möglichkeit dafür ist zum Beispiel, die Werte jeweils einer Achse aufzuaddieren und durch die Anzahl der Pixel zu teilen (Mittelwertbildung).

Auf diese einfache Weise ist das Problem auf einem leistungsfähigen Rechner auch für Videos zu lösen. Es gibt bereits freie und umfangreiche Bibliotheken (CV), welche diverse Grafikoperationen zur Verfügung stellen, darunter auch zur Farb- und Formanalyse (RWTH Aachen), (Köthe). Unglücklicherweise ist der RCube kein leistungsfähiger Rechner und mit diesen Implementierungen hoffnungslos überfordert. Die Zeit, die benötigt wird, um ein Bild auf diese Weise zu bearbeiten, kann in den Sekundenbereich gehen.

Ressourcenproblem des RCubes

Da der RCube für den Grundalgorithmus nicht genügend Leistung aufbringen kann, soll an dieser Stelle erörtert werden, wo der Algorithmus verbessert werden kann, um ihn stärker auf den RCube abzustimmen.

Eine wichtige Tatsache ist, dass der Prozessor des RCubes nicht in der Lage ist, selbst oder über einen mathematischen Coprozessor Fließkommaoperationen auszuführen. Es ist also dringend angeraten, wann immer es möglich ist, auf Integerberechnung umzusteigen.

Ein Resultat dessen ist es, die Farbmodellkonvertierung, wie sie bereits beschrieben wurde (siehe Kapitel 3), auf jeden Fall integerbasierend zu implementieren. Tests belegten, das die Konvertierung des RGB-Farbmodells in das HSV-Farbmodell um einen hohen Faktor (etwa 10-20) schneller ausgeführt werden kann, wenn beim RCube Integer- statt Fließkommaarithmetik verwendet wird.

Der Beschreibung des RCubes ist zu entnehmen, dass die maximale Übertragungsrate des Datenbusses ca. 100MByte/s beträgt. Zusätzlich ist ihr zu entnehmen, dass sowohl CPU als auch die Videohardware den gleichen Bus verwenden, sich die verfügbare Bandbreite also teilen müssen.

Es ist dementsprechend so, dass sowohl die beiden Videoströme (Ein- und Ausgabe) als auch die Programmdaten bzw. sonstige vom Programm aus erfolgende Speicherzugriffe über den gleichen Datenbus laufen. Es soll nun betrachtet werden, was allein die Videodatenströme für Anforderungen an die Hardware stellen.

Bildgröße:
720 * 576 Pixel, Farbtiefe bei 24Bit = 3Byte/Pixel
720 * 576 * 3 = 1215kByte

Bildablaufgeschwindigkeit (PAL):
Pro Sekunde 50 Halbbilder = 25 Vollbilder pro Sekunde

Der daraus resultierende Videostrom:
1215kB/Bild * 50Bilder/Sekunde 29,7MByte/Sekunde

Bei einer kombinierten Ein-/Ausgabe muss nicht nur der Videostrom in den Speicher geschrieben, sondern von dort auch wieder zur Videoausgabe übertragen werden. Somit verdoppelt sich die Datenmenge auf:

2 * 29,7MByte/Sekunde = 59,4MByte/Sekunde.

Wird nun jedes Bild vom Prozessor Pixel für Pixel gelesen, erhöht sich die Busauslastung auf knapp 90MByte/Sekunde und ist damit nahe am möglichen Maximum von ca. 100MByte/Sekunde.

Das heißt, dass nicht nur der Prozessortakt ein begrenzendes Element darstellt, sondern auch der reine Datenaustausch durchaus von Bedeutung ist.

Aus diesem Grund beschäftigt sich das nachfolgende Kapitel mit den Möglichkeiten, das Datenaufkommen so weit wie möglich zu reduzieren.

Optimierungen des Grundalgorithmus für den RCube

Die Optimierungen beschäftigen sich hauptsächlich damit, die Datenströme, welche über den Speicherbus getrieben werden müssen, so weit wie möglich zu reduzieren. Bereits im Vorwege erwähnt werden soll, dass es nötig ist, einige Annahmen zu treffen, die manche Verbesserungen erst ermöglichen:

- Es gibt nur ein Objekt mit der korrekten Farbe in der kompletten Szene.
- Das Objekt ist vollflächig, also ohne Lücken, mit einer einzigen Farbe versehen.

Diese Punkte gelten für die Erkennung geschlossener farbiger Flächen. Ein Spezialfall dieser Erkennung ist die Torerkennung. Da es passieren kann, dass ein Gegner das Tor nicht einfach teilverdeckt, sondern es durch seine Anwesenheit in einzelne Flächen unterteilt, soll ein weiterer Punkt zugefügt werden:

- Falls die Fläche doch unterteilt ist, soll ein möglichst gutes Ergebnis zurückgeliefert werden. Die zurückgelieferte Koordinate soll auf eine der Teilflächen zeigen und nicht in den Leerraum.

Ein erstes Geschenk liefert uns die Videohardware. Der Videostrom kann in verschiedenen Auflösungen und Farbtiefen gecaptured werden. In der Auflösung gibt es zwei Varianten, zum Einen die Vollauflösung mit 720*576 Pixeln und zum Anderen die halbe Auflösung mit 360*288 Pixeln.

Eine Halbierung der Auflösung auf beiden Achsen hat eine Reduktion der Bildfläche auf 1/4 zur Folge. Damit allein kann die Datenmenge für die Videohardware von etwa 60MB/s auf 15MB/s verringert werden. Eine Verringerung der Farbtiefe wäre zwar möglich, ein möglichst präzises Farbmodell ist allerdings unabdingbar für eine gute Farberkennung, so dass eine Farbtiefe von 24Bit, also 3 Bytes/Pixel beibehalten wurde.

Da die Optimierung der Videohardware ausgereizt ist, folgt eine Verbesserung des Algorithmus selbst.

Die Auflösung des Bildes braucht prinzipiell nur so groß zu sein, dass das kleinste zu erkennende Objekt ein Pixel groß ist. Ein Herunterrechnen (Skalieren) der Bilddaten würde sich zwar auf den ersten Blick anbieten, hätte aber je nach Skalierungsmethode den Nachteil des hohen Rechenaufwandes und/oder der zusätzlich starken Belastung des Datenbusses.

Zudem wäre entweder zusätzlicher Arbeitsspeicher nötig oder eine Zerstörung der Originaldaten, wenn das Ergebnis im gleichen Speicher verbleiben soll.

Aus diesem Grunde macht es Sinn, eine Art virtuelles Skalieren vorzusehen. Sinngemäß wird das Bild in eine Anzahl von Feldern eingeteilt, die von nun an *Blöcke* genannt werden. Da es letztendlich nur wichtig ist, ein 'OK' (oder 'nicht OK') für einen Block zu erhalten, kann

eine beliebige Funktion verwendet werden, die aus einem Cluster von Pixeln (einem Block) ein `true` oder `false` generiert. Experimente ergaben, dass es auch in Hinsicht auf die Störunanfälligkeit durchaus hinreichend ist, in jedem Block nur vier Pixel auf die korrekte Farbe zu testen. Das könnte wie in Abbildung 3.2.7 gemacht werden.

3.2.7: Gemessene Pixel in einem Block

Hat eine angemessene Anzahl der Pixel (mindestens zwei) die richtige Farbe, so wird der Block als erkannt zurückgeliefert.

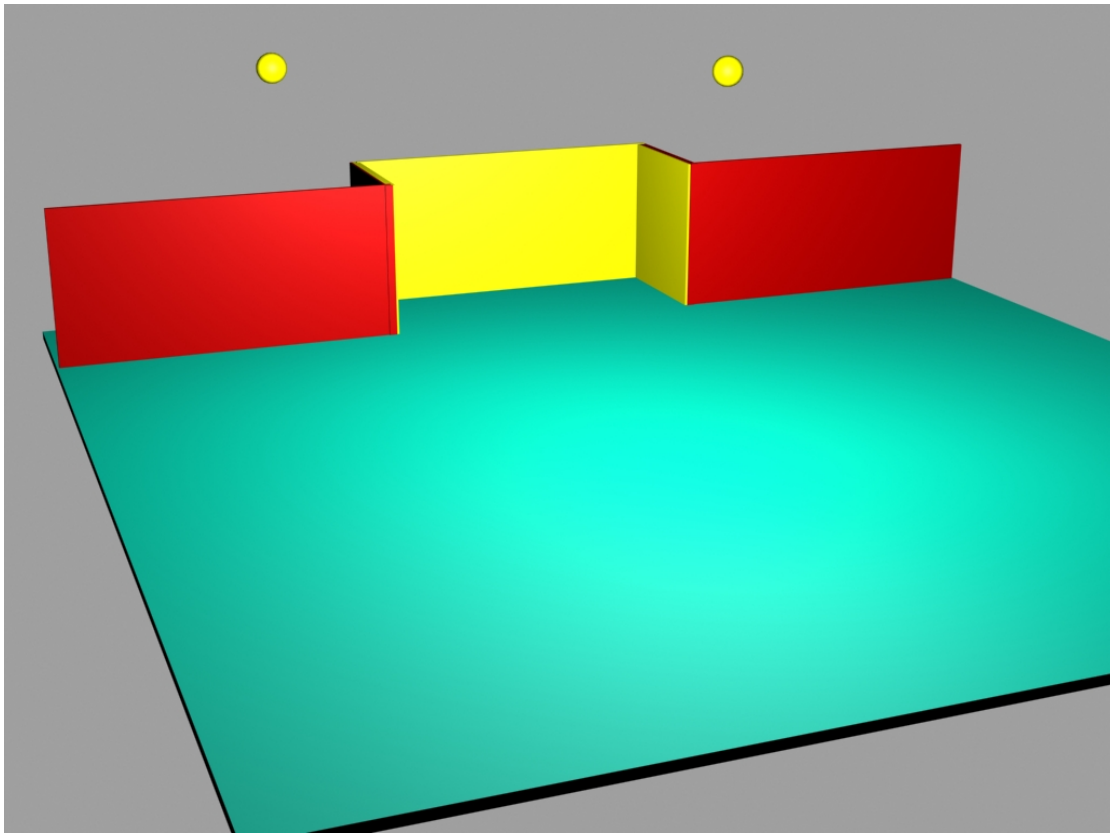
Der nächste Optimierungsschritt basiert auf den weiter oben beschriebenen Annahmen. Da es nur ein Objekt gibt und dieses vollständig mit nur einer Farbe ausgefüllt ist, reicht es, nur die Ränder zu detektieren. Per Definition ist die von den Rändern eingeschlossene Fläche komplett mit der korrekten Farbe gefüllt.

An dieser Stelle bauen sich eine Reihe von Möglichkeiten auf, wie eine Randerkennung vorgenommen werden kann. Man könnte kreisförmig Strahlen von außen nach innen laufen lassen. Um sich die sin-/cos-Berechnung zu ersparen, könnte man auch rechteckig von außen nach innen laufen. Eine Option wäre dabei, bei einem ersten Treffer auf einen anderen Algorithmus umzuschwenken, beispielsweise ein ’Entlanghangeln’ am Rand.

Der Nachteil der ’Strahlenmethode’ für Kreise ist das rechenaufwendige Bestimmen der Suchabfolge und des späteren Füllens. Bei der Rechteckmethode muss eine Lösung gefunden werden, wie nur der Rand geprüft wird und das Innere beim Suchen ausgespart wird. Eine Breiten- oder Tiefensuche könnte hier helfen.

Ich entschied mich allerdings für eine eher etwas unkonventionelle Lösung:

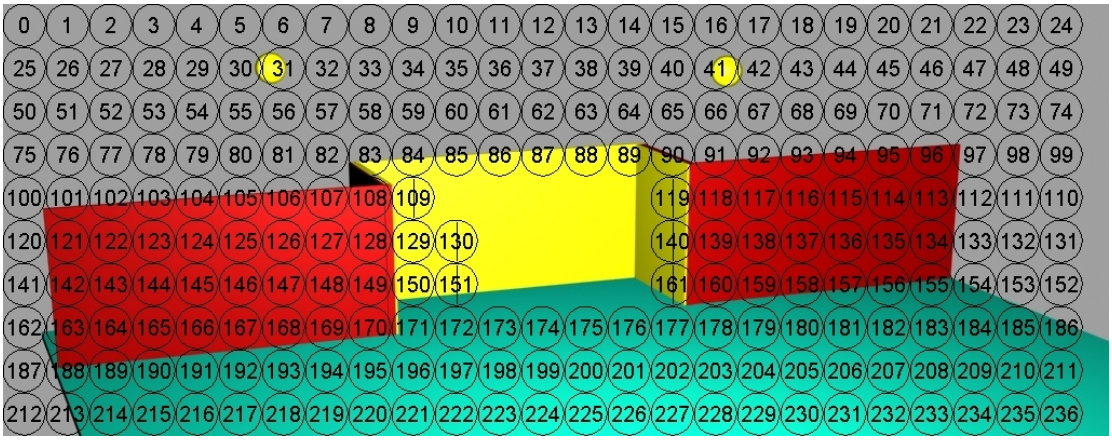
Eine Annäherung an den Rand des Objektes erfolgt von zwei Seiten (links und rechts). Die Beschreibung soll mit Bildern illustriert werden. Das vollständige Ursprungsbild in Abbildung 3.2.8 stellt ein computergeneriertes Modell eines Roboterfußballtores dar. Die beiden gelben Kugeln links und rechts über dem Tor sollen 'Schmutz' symbolisieren, wie er in der Realität durch Störungen in der Versorgungsspannung, Rauschen des Sensors etc. ausgelöst werden kann.



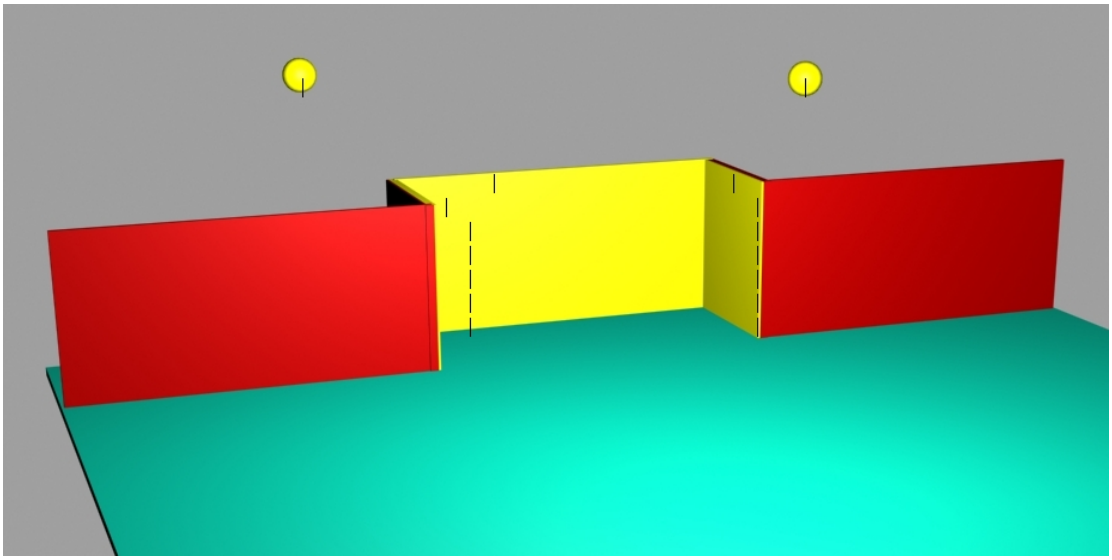
3.2.8: Ursprungsbild

Die Randerkennung arbeitet zeilenweise von oben nach unten. Von links an wird nach einem korrekten Block gesucht. Erreicht die Suche den rechten Rand, so wurde kein Block gefunden und die Suche wird in der nächsten Zeile fortgesetzt.

Wurde allerdings ein Block gefunden, so stoppt das Verfahren und beginnt in der gleichen Zeile vom rechten Rand aus zu suchen, bis ein Block gefunden wird (da von der anderen Seite aus bereits ein Block gefunden wurde, muss es einen geben). Die Abbildungen 3.2.9 und 3.2.10 zeigen die Arbeitsweise noch einmal auf. Die Zahl in den Kreisen ist die Schrittnummer. Ein horizontaler Strich gibt an, dass der Algorithmus einen gültigen Block gefunden und markiert hat. Man bemerke, dass die Blockgröße auf beiden Bildern voneinander differiert. Aus illustratorischen Gründen wurde die Blockgröße in Bild 3.2.9 erheblich größer gewählt, so dass die beiden Kugeln nicht erkannt werden konnten.

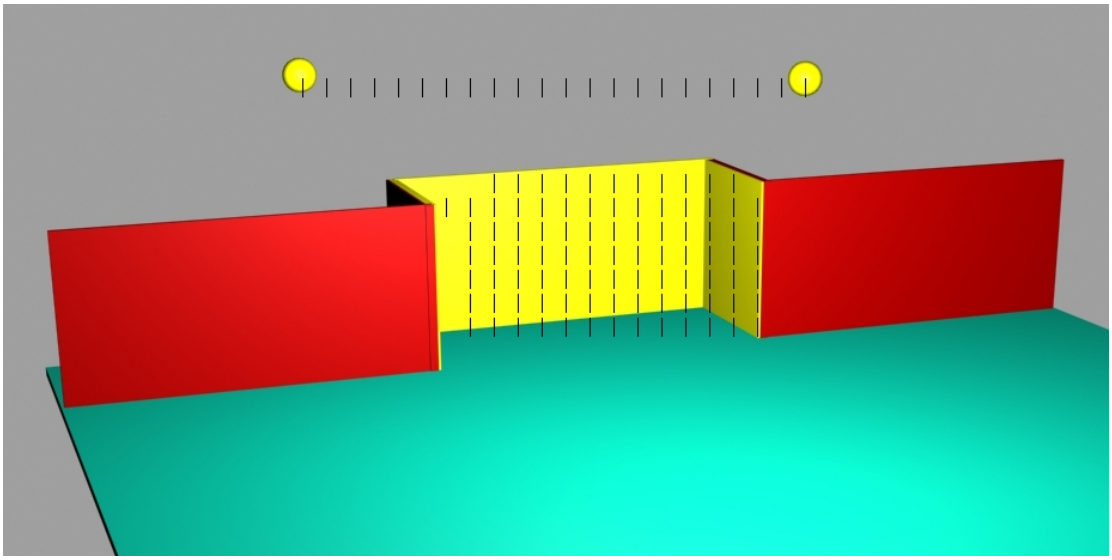


3.2.9: Schrittfolge der Randerkennnung



3.2.10: Detektierte Ränder

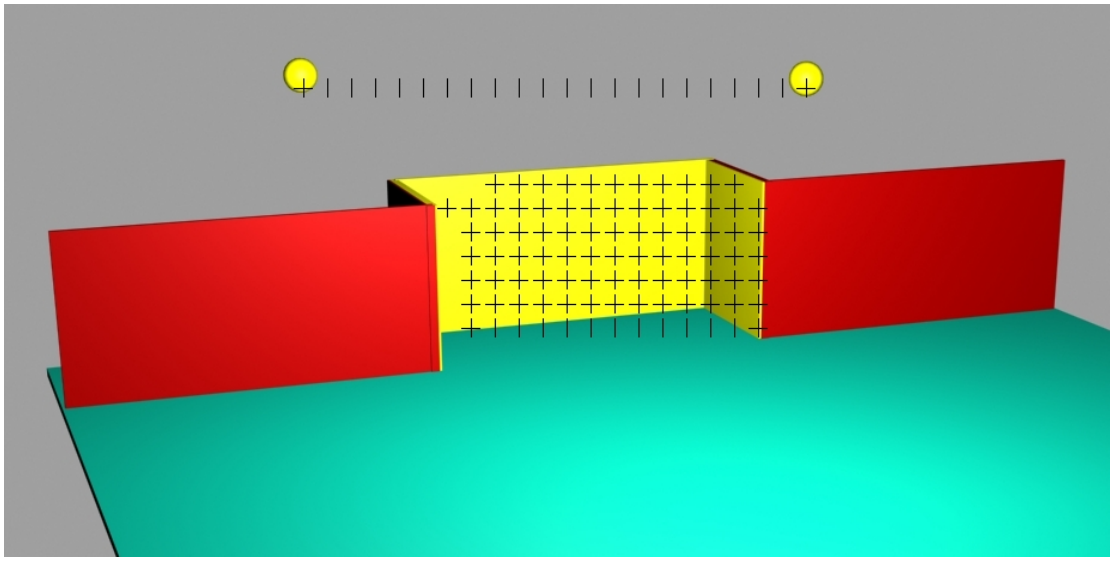
Das Problem ist allerdings, dass z.B. die Einbuchtungen bei einer ∞-Form nicht erkannt werden. Ausserdem ist die Reaktion auf 'Schmutz' etwas kritisch, da der Bereich zwischen den Ränder ausgefüllt wird. Man beachte bei dem Beispielbild 3.2.11 die Füllung zwischen den beiden 'Schmutzkugeln'.



3.2.11: Horizontal gefüllte Ränder

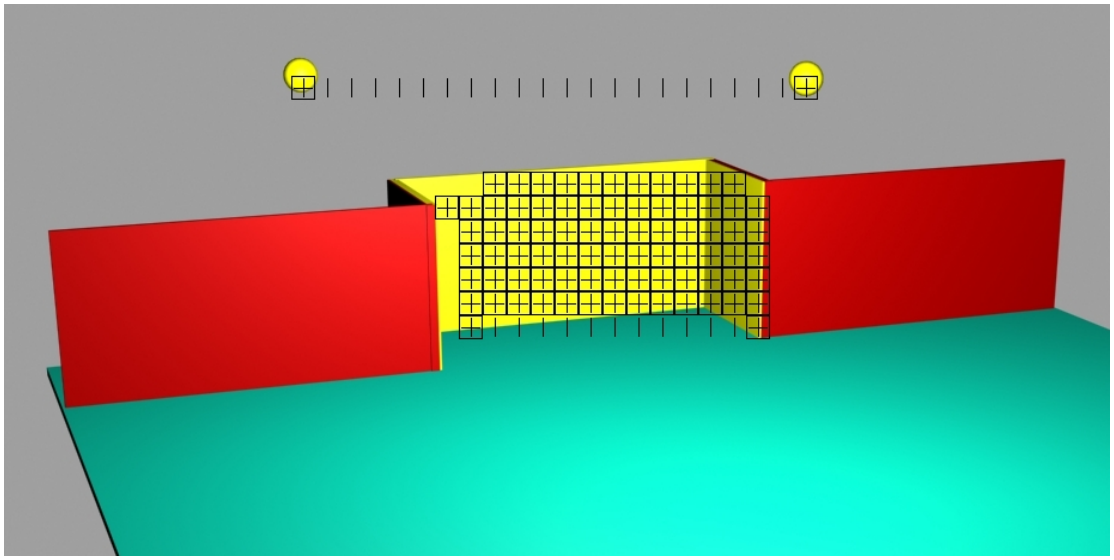
Wäre das Tor etwas kleiner und die Kugel weiter voneinander entfernt, so entstände ein sehr ungünstiges Verhältnis zwischen der Torfläche und der Schmutzfläche.

Die Anwendung des selben Verfahrens um 90° verdreht und eine anschliessende Verknüpfung der Ergebnisse über den logischen AND-Operator sorgen für eine bessere Randerkennung (siehe Abbildung 3.2.12). Der Zwischenraum des Schmutzes wird nur von einem Durchgang erkannt, somit bleibt das Ergebnis nach der AND-Verknüpfung 'nicht erkannt'. Die senkrechten Striche setzt wieder der horizontale Algorithmus. Der zweite vertikale Durchgang generiert die waagerechten Striche.



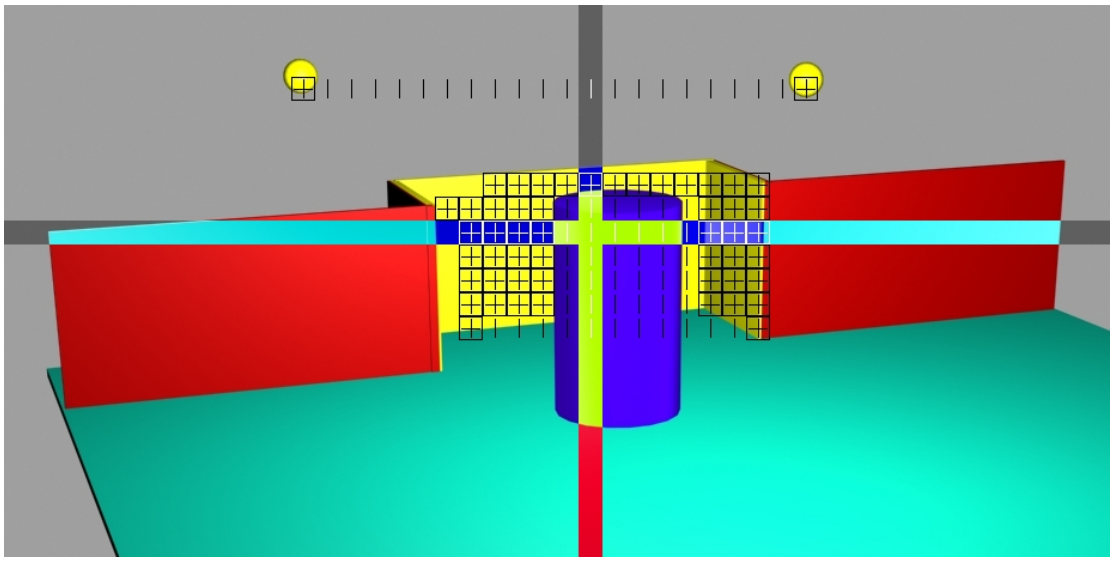
3.2.12: Ränder beider Durchläufe

Das Ergebnis der AND-Verknüpfung steht in Abbildung 3.2.13. Die Rechtecke um die Blockgrenzen markieren das Resultat.



3.2.13: Ergebnis der AND-Verknüpfung

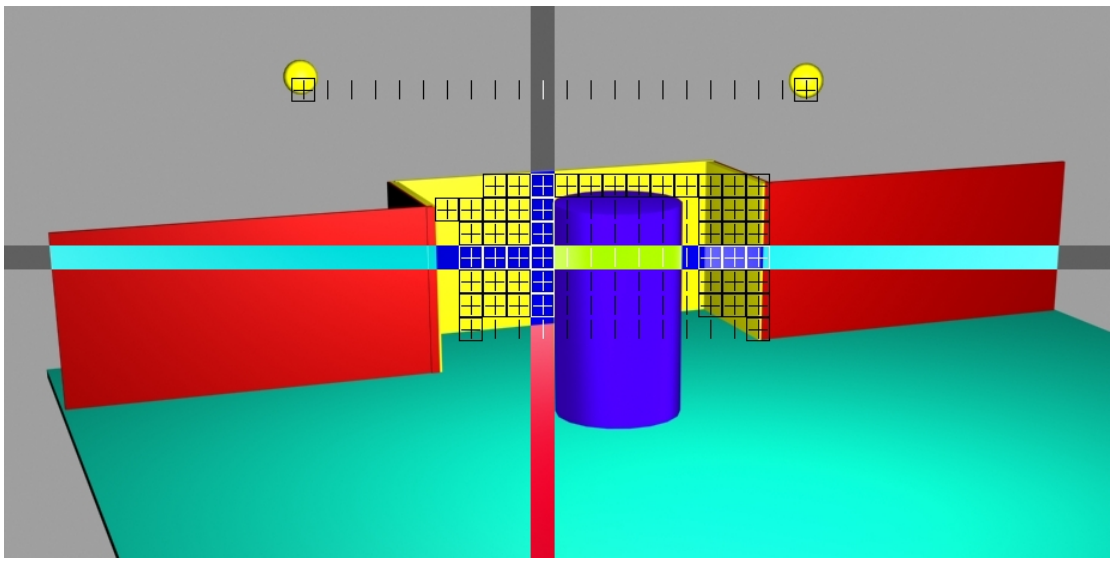
Was nun noch fehlt, ist eine Berechnung des Zentrums der Fläche. Der Mittelwert kann einfach gebildet werden: Die Positionen aller markierten Blöcke für jede Achse werden aufsummiert und durch die Anzahl der markierten Blöcke geteilt. Als kleine Herausforderung steht nun ein Gegner halbwegs im Zentrum des Tores und teilt dieses in zwei Flächen (siehe Abbildung 3.2.14). Erkennbar ist, dass das errechnete Zentrum mitten in den Gegner hinein-



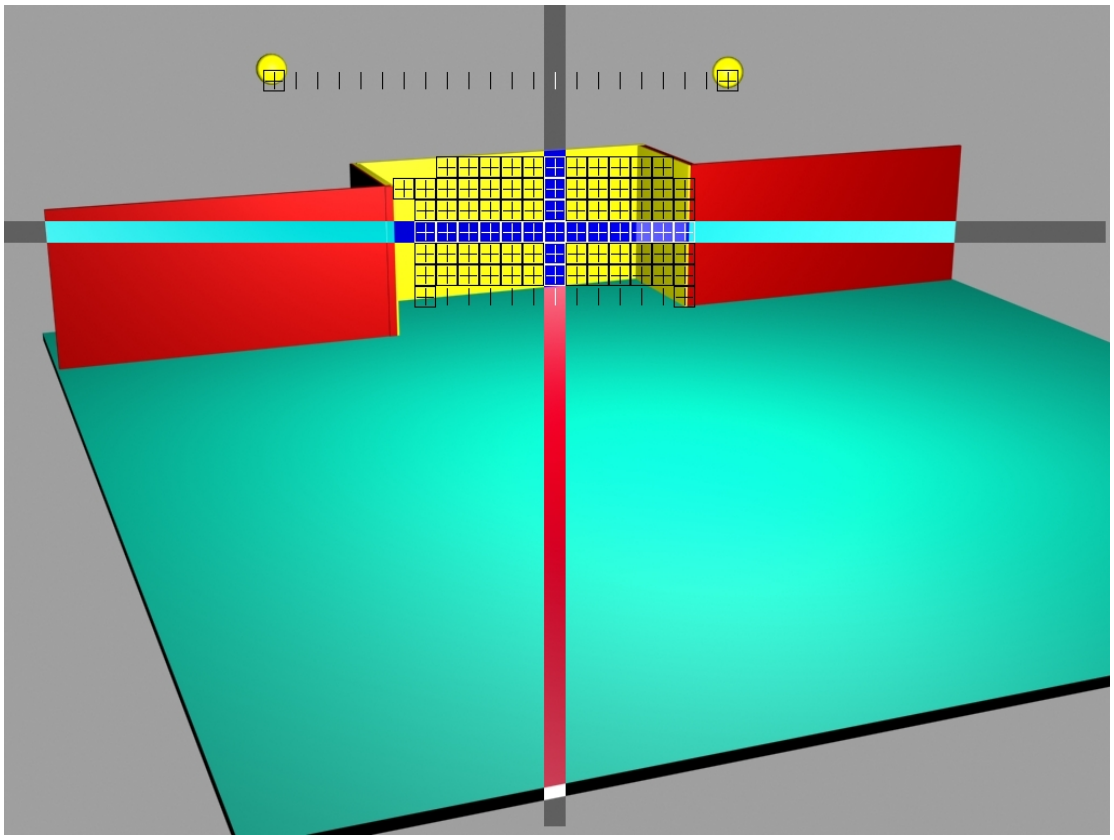
3.2.14: Gegner im Tor - Mittelwert

fällt. Ein Schuss würde ihm direkt den Ball zuspielen und sollte möglichst vermieden werden. Verändert man die Strategie zur Schwerpunktberechnung auf den Median, so wird nur die Fläche berücksichtigt, die auch tatsächlich markiert ist. Diese Strategie ist zwar nicht perfekt, da unter ungünstigen Umständen (die beiden einzelnen Flächen sind annähernd gleich groß) der innere Rand errechnet wird (siehe Abbildung 3.2.15), ist aber eine erhebliche Verbesserung im Vergleich zur Mittelwertbildung.

Der Vorteil dieser Methode ist, dass sich keine Änderung in Bezug auf eine einzige Vollfläche ergibt, für das Problem der Torerkennung konnte aber sehr viel gewonnen werden. Die Abbildung 3.2.16 stellt das Ergebnis mit Medianberechnung im leeren Tor dar.



3.2.15: Gegner im Tor - Median



3.2.16: Endergebnis mit markierten Median

4 Design und Realisierung

Dieses Kapitel beschreibt das Design und die Implementierung der während dieser Arbeit entstandenen Software und deren zugehörige Werkzeuge.

Das Kapitel 4.1 über den Prototypen beschäftigt sich mit einem Entwicklungsframework, welches zum entwicklungszeiteffizienten Testen entwickelt wurde.

Die Tools (Kapitel 4.2) sind zwar nicht untrennbar mit der Anwendung verbunden, stehen aber in einen sehr engen Bezug zu ihr. Sie können allerdings auch losgelöst von der Anwendung verwendet werden.

Im Kapitel 4.4, es handelt von der auf dem RCube basierenden Serviceschicht, wird auf die in der Zielsetzung (siehe Kapitel 1.2) definierten Anwendung Bezug genommen. Dies ist der zentrale Teil.

Zuletzt wird im Kapitel 4.5 noch ein Proof of Concept, also eine Art Beweis der Funktionsfähigkeit, beschrieben. Dieser Beweis manifestiert sich als ein Colortracker, der über Aktoren einer in X- und Y-Richtung drehbaren Kamera ein farbig markiertes Objekt verfolgt.

4.1 Prototyp

Als eine Besonderheit bei der Entwicklung von Software für den RCube stellte sich die geringe Entwicklungsgeschwindigkeit heraus. Diese lag darin begründet, dass als Entwicklungsumgebung nur ein Texteditor und der Kommandozeilencompiler zum Entwickeln und Übersetzen zur Verfügung standen.

Des weiteren verhielt es sich so, dass jede Übertragung des Programmes auf den RCube etwas Zeit in Anspruch nahm, pro Übertragung nicht über alle Maße unangemessen viel, aber in der Summe nicht unerheblich.

Das Debuggen geschah per Ausgabe über die Kommandozeile, wie es bereits beim legendären ZX81 der Fall war (sinclair 1980). Bei Fehlern, insbesondere Speicherzugriffsverletzungen, wie sie gern bei der Verwendung von Pointerarithmetik auftreten, stand immer ein mühevolles Debuggen über diesen Weg bevor.

Um also schnell und unkompliziert Algorithmen entwickeln und testen zu können, war es erforderlich, den Weg abzukürzen.

Die Zielsetzung sollte dabei nur in kleinen Teilen mit der Zielsetzung der eigentlichen Anwendung übereinstimmen. Die Anforderungen lagen nicht im Bereich der Echtzeit und hohen Performance, sondern ausschließlich in der schnellen Implementierung von Grafikalgorithmen, um sich die Umsetzung mit anschließenden langwierigen Debuggingsessions auf dem RCube zu ersparen. Auch sollte keine Benutzerschnittstelle implementiert werden, eine Verwendung in einer Entwicklungsumgebung erachtete ich als hinreichend.

Es geht hier also ausschließlich um die Qualität von Algorithmen. Das Laufzeitverhalten ist nur von sekundärer Bedeutung.

Da ich bereits viele Erfahrungen mit der Programmiersprache Java (Sun a) sammeln durfte und in dieser Sprache recht schnell implementieren kann, entschied ich mich, sie zu verwenden. Als Entwicklungsumgebung stand das freie Eclipse (Eclipse) zur Verfügung, welches sehr viele Möglichkeiten zur Reduktion des Programmieraufwandes bietet.

Damit ich auf Videoquellen, wie es beispielsweise auch Framegrabberkarten sind, zugreifen konnte (wahlweise sowohl unter Windows als auch unter Linux), verwendete ich das JMF (**J**ava **M**edia **F**ramework) (Sun b).

Dieses Framework sollte nicht als fertiges Produkt verwendet werden, sondern eher als Testplattform und Workbench, in der die Qualität von Algorithmen und Verfahren getestet werden können. Die Performance ist auf Grund des sehr unterschiedlichen Designs uns Aufbaus[1] des RCubes gegenüber einem PC nicht oder allenfalls tendenziell vergleichbar und kann deshalb nicht übertragen werden.

[1] Die Unterschiede sind bereits bei dem Prozessordesign eklatant. Während intel-kompatible Prozessoren auf der CISC-Architektur (Complex Instruction Set) basieren, sind ARM-Derivate RISC-Prozessoren (Reduced Instruktion Set). Zudem sind PC-Prozessoren bereits seit einigen Generationen mit einem mathematischen Coprozessor ausgerüstet, welcher Fließkommaoperationen mit hoher Geschwindigkeit ausführen kann. ARMs hingegen haben derzeit überwiegend keinen mathematischen Coprozessor. Fließkommaoperationen müssen durch den Compiler über Integerberechnungen emuliert werden und sind entsprechend zeitaufwendig.

Design

Das Framework sollte vom Prinzip her eine Pipeline sein, eine Abfolge von Mechanismen zur Bildmodifikation, 'Filter' genannt.

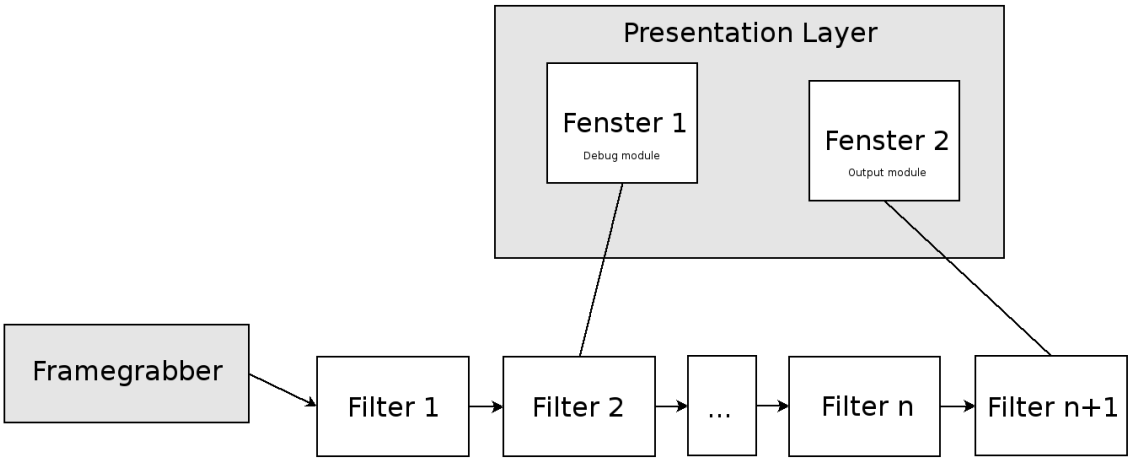


Abbildung 4.1: Prizipieller Aufbau des Prototypen

Ein Filter erhält Bilddaten, wendet auf ihnen seinen Algorithmus zur Verarbeitung an und erzeugt aus dem Resultat neue Bilddaten.

Eine Ausnahme stellt ein Augabefilter dar. Er schreibt die ihm übergebenen Bilddaten in ein Fenster und macht sie so dem Benutzer sichtbar. Die Daten werden dabei nicht modifiziert und im Originalzustand zurückgegeben. Es können somit beliebig viele Ausgaben in die Pipeline gehängt werden [2].

Eine Klasse mit dem Namen `Filter` sollte die Oberklasse für alle Filerimplementationen bilden. Sie definiert im wesentlichen das Interface und bietet darüber hinaus noch einige Annehmlichkeiten, wie z.B. eine Parameterverwaltung.

Es stellte sich heraus, dass generische Filterklassen erzeugt werden konnten, von denen die eigenen Filter ableitbar sind, wie z.B. dem Pixelfilter. Der Entwickler braucht nur die Methode zu überschreiben, welche für ein Pixel angewendet werden soll. Das Lesen und Schreiben der Bilddaten, Generierung neuer Bilddaten usw. wird von der Oberklasse abgenommen.

[2] Dies ist die erste Stufe eines automatisierten Bildbearbeitungsprogrammes. Mit einer grafischen Oberfläche könnte man Filter anordnen und miteinander verbinden. So könnte man automatisch mehrere Bilder die selbe Filterfolge durchlaufen lassen.

Es wurde recht schnell deutlich, dass Filter mit Pixelbezug nur eine sehr unzureichende Basis für die Aufgabenstellung darstellen. Es entstand somit eine weitere Basisklasse, welche wiederum die Pixelklasse als Oberklasse hatte und somit auf der bereits investierten Arbeit aufbaute. Ihre Funktionsweise sollte auf der Verarbeitung von Bildblöcken (rechteckige Ansammlungen von Pixeln) beruhen.

Eine weitere überschreibbare Methode bietet die Möglichkeit, die Abfolge der Blockabarbeitung zu beeinflussen. Somit wurde die die Funktionalität bereitgestellt, um Umrissfilter implementieren zu können.

Unter den implementierten Filtern befinden sich sowohl triviale Filter, wie zum Beispiel das Anzeigen der Blockgrenzen, als auch komplexere Filter, nämlich die Umrissfilter. Unter ihnen ist auch die Version, die in gleicher Weise auf dem RCUBE reimplementiert wurde.

Zusätzliche Filter zur Farbkonvertierung (nach Graustufen) oder der Farbmodellkonvertierung (nach HSB/HSV) sind ebenfalls im Repertoire enthalten. Zu Dokumentationszwecken wurde ein Filter zum Abspeichern von Bildern entwickelt.

Realisierung

Die Realisierung erfolgte wie im Design beschrieben. Damit den Filtern Daten übergeben werden können, bzw. Daten gelesen werden, ohne das Interface zu verändern, ist in der Filterklasse eine Map definiert, welche Parameter speichert und zurückliefert. Per Definition soll die abgeleitete Filterklasse Klassenattribute besitzen, die einen sprechenden Namen haben und den Key für die Map bereitstellen. Auf diese Weise ist es möglich, Positionsdaten oder ähnliche Dinge auszutauschen.

Man hätte das auch durch einfache Getter und Setter lösen können. Zum Zeitpunkt der Implementierung erschien die Idee recht gut, die generischen Getter und Setter zu verwenden, damit die einzelnen Filter Daten untereinander automatisiert austauschen können. Das erwies sich aber als zu hohes Konzept für ein Testframework. Gemäß dem Motto ’keine Technik auf Vorrat’ wurde die Idee fallengelassen, die generischen Getter und Setter - da sie ja bereits vorhanden waren - blieben allerdings bestehen.

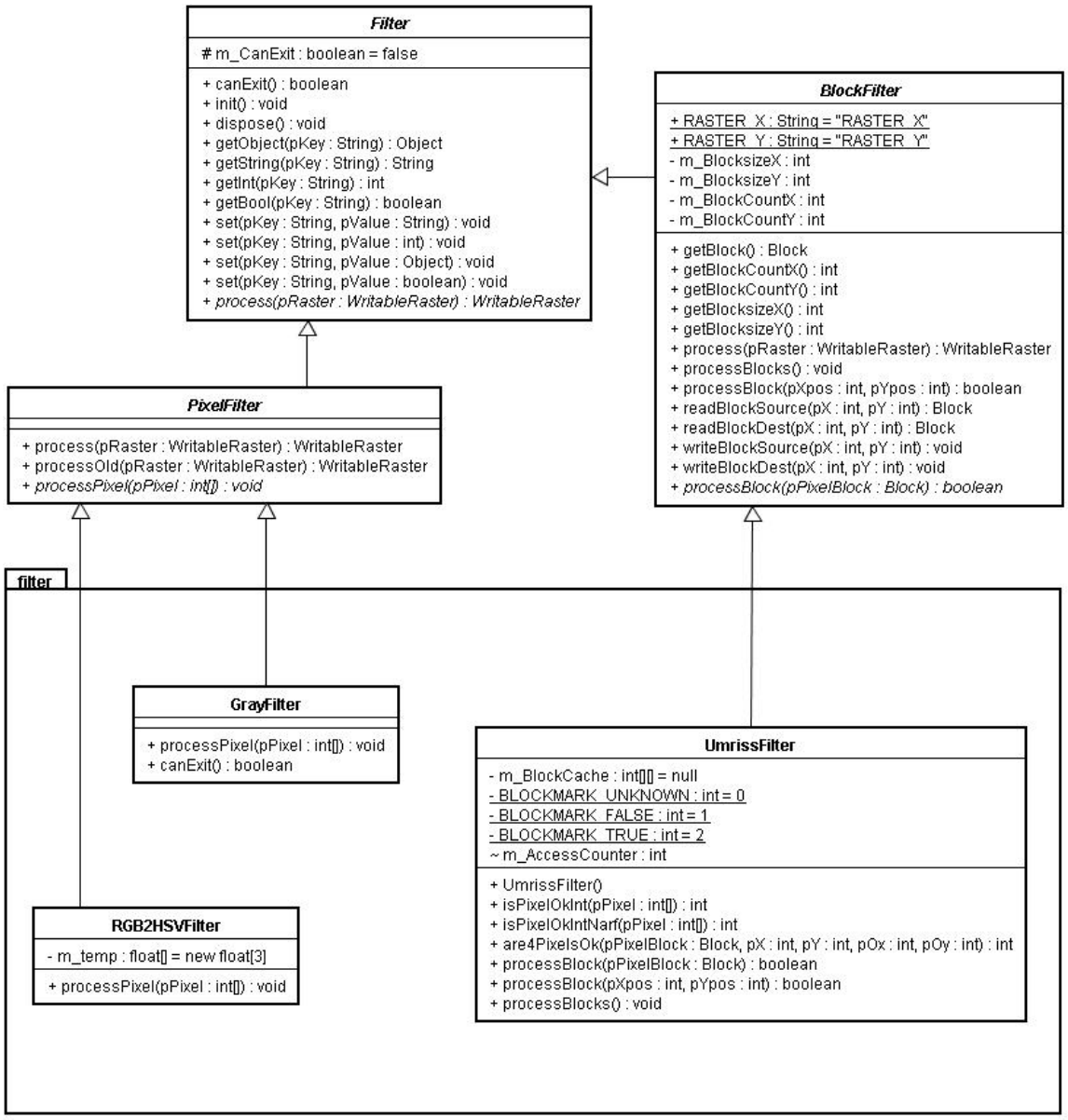


Abbildung 4.2: Klassendiagramm der Filter des Filterframeworks

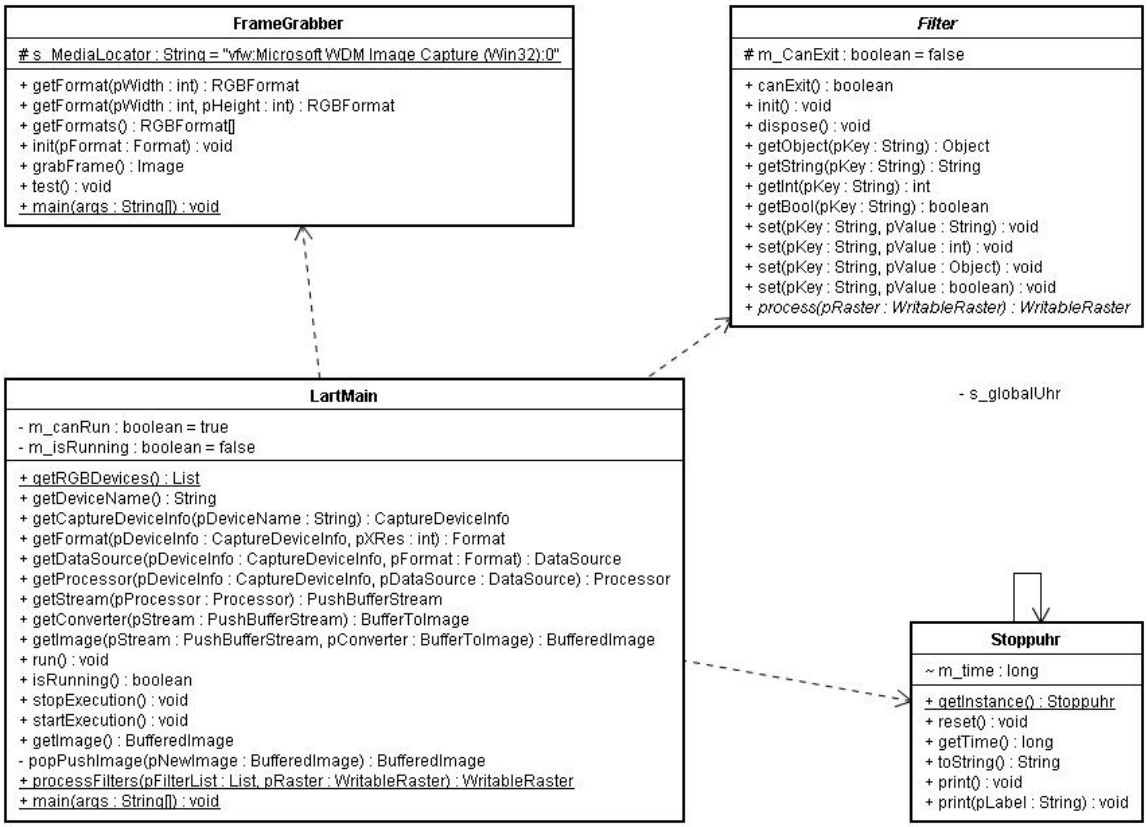


Abbildung 4.3: Klassendiagramm des Filterframeworks

Das Hauptprogramm - `LartMain` - besteht hauptsächlich aus dem Initialisierungsteil für den `Framegrabber` und den einzelnen Filtern. Darauf folgt eine Endlosschleife, die ein Bild aus dem Grabber liest und es anschließend durch die Filterpipeline schickt. Eine Klasse namens `Stoppuhr` kann Zeitmessungen vornehmen. Die Zeitmessungen sind nur relativ zu sehen und zum Abschätzen gedacht. Eine Prognose zur Laufzeit auf dem RCube kann daraus nicht entstehen. Vielmehr besteht die Möglichkeit, zu schätzen, ob ein Algorithmus schneller, deutlich schneller oder möglicherweise sogar langsamer ist.

Der `Framegrabber` wurde an ein Beispiel der Java-Seite von Sun (Sun Microsystems) angelehnt.

Dieses Framework war ausgesprochen hilfreich, aus dem ersten simplen Algorithmus den im nächsten Kapitel beschriebenen Algorithmus zu evolutionieren. Zudem ist es recht praktisch, sich 'schnell mal eben' einen Filter schreiben zu können, um bestimmte Aspekte der Bildverarbeitung zu evaluieren, etwa auf reines Farbsehen umzustellen, in dem Sättigung und Helligkeit maximiert werden. Dass die Ausführungsgeschwindigkeit teilweise zu wünschen übrig lässt, tut der Sache keinen Abbruch. Performancefragen sind an dieser Stelle nebensächlich, die generelle Funktionsfähigkeit ist wichtig.

Praktisch war es außerdem, Bilder zur Dokumentation generieren zu können, wie im Kapitel 3.4 über die Optimierung des Grundalgorithmus ersichtlich ist.

4.2 Tools

Es erwies sich als umständlich, mit den üblichen Programmen zur Grafikbearbeitung wie etwa dem freien Gimp (Gimp), Bilder zur Dokumentation oder Analyse zu erzeugen. Daher sollte eine allgemein gehaltene Software entstehen, die anschließend auf den spezifischen Anwendungsfall zugeschnitten werden kann.

Die Bedienung sollte einfach sein und ohne eine grafische Oberfläche auskommen können. Folgende Bedingungen sollten erfüllt werden:

- Ein Bild soll in seine RGB-Farbebenen zerlegt werden. Die Ebenen sollen entsprechend ihrer Farbe gefärbt werden und nicht in Graustufen, wie es bei den gängigen Grafikprogrammen üblich ist.

  Das ist insbesondere zur Dokumentation interessant, um Bilder nicht zusätzlich mit der Farbe beschriften zu müssen. Dem ungeübten Betrachter fällt es leichter, den Farbanteil direkt zu sehen, als über die Helligkeit auf eine Farbe schließen zu müssen.
  Zusätzlich ist es für Analysezwecke interessant, um Farbanteile abschätzen zu können.

- Die einzelnen Ebenen des HSV-Farbmodells sollen in Graustufen abgespeichert werden.

  Die Farbebene ist dabei nur dem geübten Betrachter hilfreich, der die Farbe über die Helligkeit schätzen kann. Interessanter ist die Sättigungsebene. Hier lässt sich erkennen, welche Teile des Bildes besonders stark oder schwach mit Farbe gesättigt sind. Die Helligkeitsebene ist je nach Anwendungsfall auch interessant, speziell bei der Torerkennung aber eher weniger von Interesse.

- Im HSV-Farbmodell sollen Sättigung und Helligkeit maximiert werden. Die Farbe bleibt damit bestehen und kann analysiert werden.

  Auf diese Weise kann man deutlich machen, welche Farbe dunkle Bildteile haben, die auf das menschliche Auge schwarz wirken. Für Kameras und eine hinter ihr liegende Bildverarbeitung muss das nicht der Fall sein. Auf diese Weise kann der Mensch sehen, was die Kamera sieht und das Wissen in die Verarbeitung, Beurteilung und Analyse einfließen lassen

- Alle Bildpunkte mit einer angegebenen Farbe und Farbtoleranz sollen im HSV-Farbmodell markiert werden.

  Auf diese Weise kann schnell deutlich gemacht werden, ob eine gewählte Farbe tatsächlich den Erwartungen entspricht. Es lässt sich sehr schnell erkennen, ob zu viele, zu wenige oder gar die falschen Pixel markiert werden.

- Es sollen alle Pixel einer Sättigung (HSV-Farbmodell) $\geq$ einem angegebenen Wert markiert werden.

  So lässt sich feststellen, ob interessante Bildbereiche tatsächlich genug gesättigt sind, um fehlerfrei verarbeitet werden zu können.

- Die vorigen beiden Punkte sollen über ein logisch UND verknüpft werden. Es müssen also sowohl Farbe und Sättigung korrekt sein.

  Dies ist beinahe schon eine Farbobjekterkennung, nur die Berechnung des Zentrums fehlt noch. Es lässt sich so direkt feststellen, ob die Parameter verändert werden müssen, wenn zum Beispiel die falschen Pixel markiert werden.

- Ein Bild soll geladen und in einem Fenster dargestellt werden. Eine pixelgenaue Abfrage der Farbwerte soll möglich sein. Ein Zoom wäre angebracht, um sehen zu können, welcher Pixel tatsächlich abgefragt wird.

  Sehr praktisch, um 'mal eben schnell' eine interessante Farbe anzusehen und sie dann in die Farbmaskierung einspeisen zu können.

Design

Da in der Anwendung zwar verschiedene Aufgaben erfüllt werden sollen, diese aber durchaus Ähnlichkeiten besitzen - alle bekommen ein Bild als Quelldaten und liefern 0..n Bilder als Ergebnisdaten zurück - bot sich ein Plugin-ähnliches Klassenmodell an. Eine Oberklasse (`ImgProcessor`) konnte so grundlegende und allen Plugins gemeine Funktionalitäten kapseln.

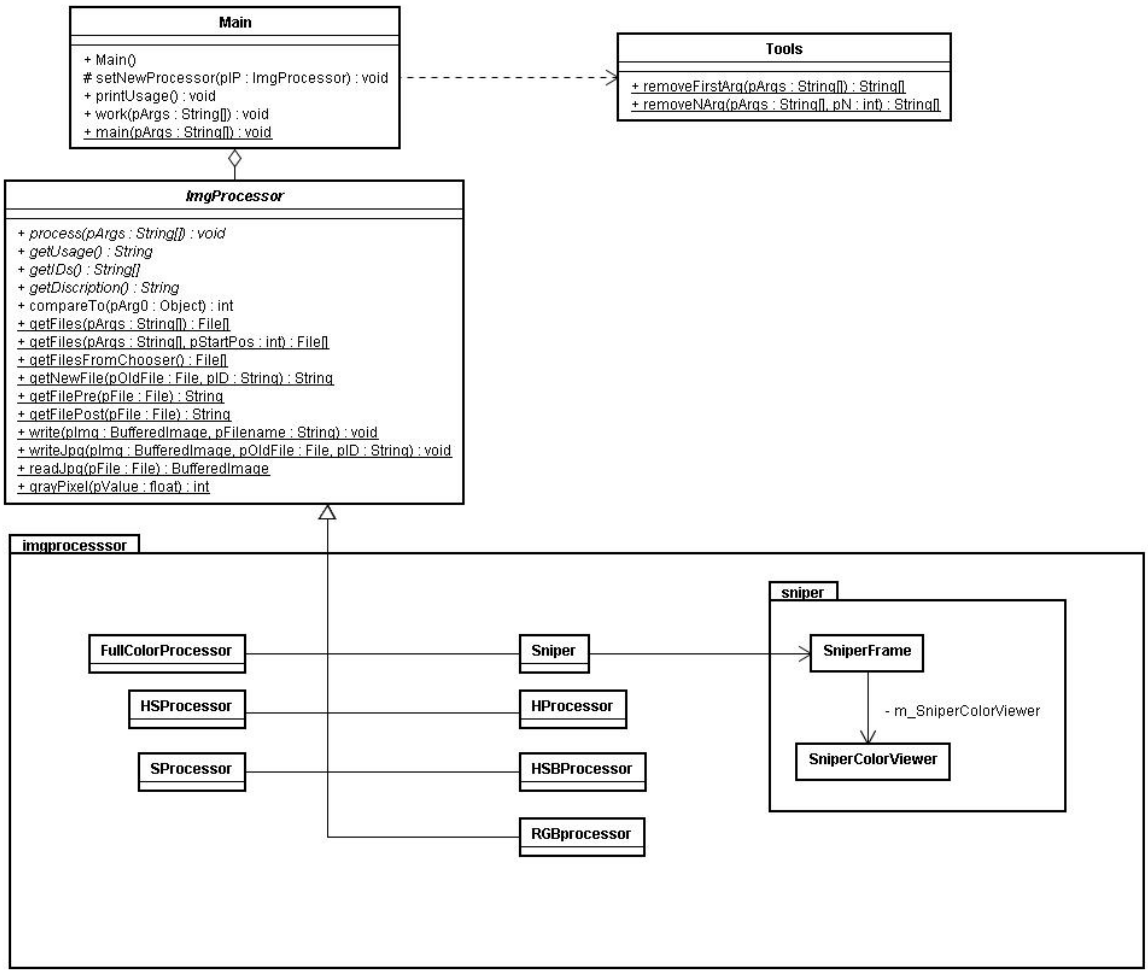


Abbildung 4.4: Klassendiagramm des Zerlegers

Die Plugins geben die Kommandozeilen-IDs an, mit denen sie aufgerufen werden möchten. Das eigentliche Parsen übernehmen sie allerdings selbst, um auch speziellen Konfigurationen gerecht werden können.[3].

Eine Klasse (`Main`) bietet einen Programmeinsprungspunkt und die Verwaltung der Plugins. Somit ist es ohne großen Aufwand möglich, die Funktionalität des Programmes zu erweitern.

[3] Im Nachhinein gesehen hätte eine Parserklasse, welche die Kommandozeilenoptionen zerlegen kann und die als Basis für spezielle Kommandozeilenparser hatte dienen können, durchaus ihre Berechtigung gehabt. Eventuell hätten über eine Factory Plugin und Parser ausgeliefert werden können.

Implementierung

Die Implementierung selbst erfolgte in Java. Zur einfacheren Verwendung unter Windows (Microsoft) wurde der freie EXE-Wrapper jSmooth (JSmooth) verwendet, da er sich als ANT-Task (Apache) in Eclipse einbinden lässt und auch alle sonstigen Anforderungen in Hinblick auf Konfigurierbarkeit und Lizenzmodell erfüllt.

Ein Aufruf des Programms:

```
>zerleger.exe
```

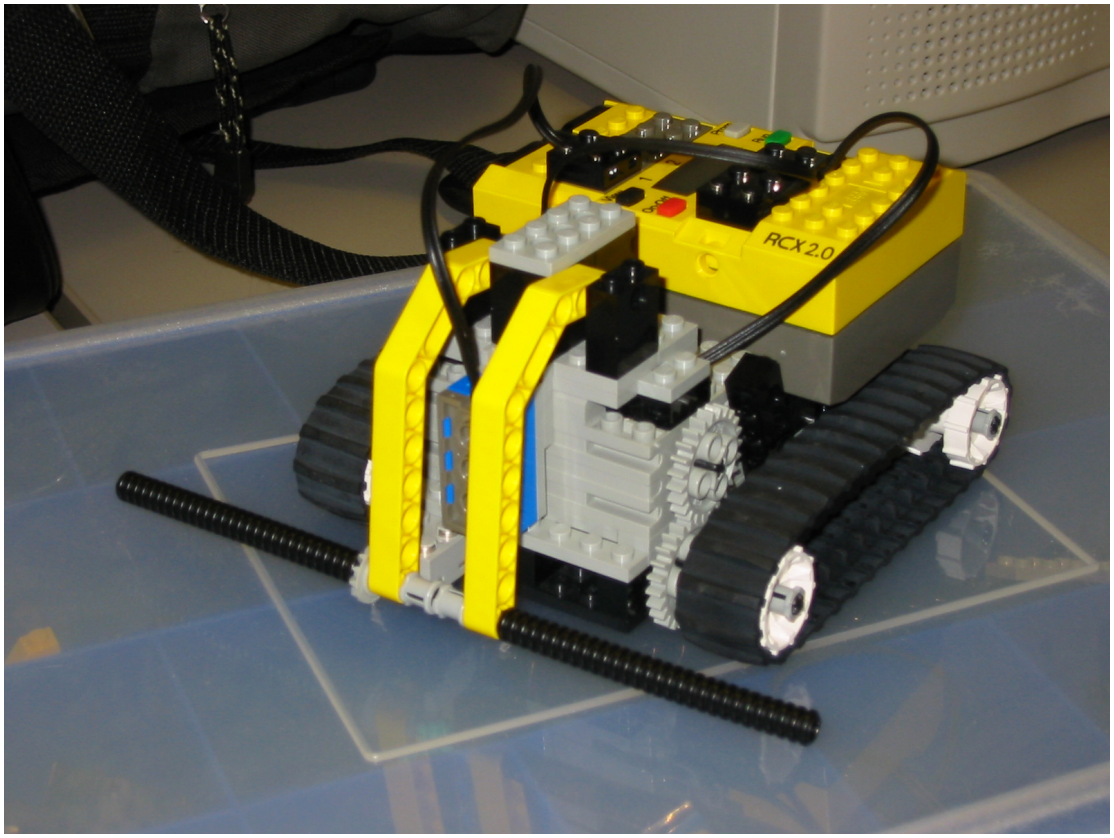
erzeugt folgende Ausgabe:

```
usage:
  <Program> -fc <File>*
  <Program> -h <Color[0..255]> <Tolerance[0..128]> <File>*
  <Program> -hsb <File>*
  <Program> -hs <Color[0..255]> <Tolerance[0..128]> <min saturation[0..255]> <File>*
  <Program> -rgb <File>*
  <Program> -s <min saturation[0..255]> <File>*
  <Program> -sniper <File>
  <Program> -help <Option>
```

Abgesehen von der Option `-sniper` wird kein grafisches Interface benötigt. In den nachfolgenden Unterkapiteln sollen die Optionen einzeln erklärt werden. Die Basis der Erklärungen bildet neben dem Programm selbst ein Bild, auf dem ein LEGO-Roboter[4] zu sehen ist.

Abbildung 4.5.1 zeigt eine Ansicht des originalen Quellbildes.

[4] Sein Name ist übrigends Arnold und seines Zeichens Sieger des Roboterpraktikums im WS 2003. Der Name hat übrigens mit seiner Angewohnheit zu tun, marodierend über das Spielfeld zu pflügen und sämtliche Aufbauten aus ihren Verankerungen zu reißen. Oft überquerte er nicht die zum Spielfeld gehörige Brücke, sondern riss sie aus ihrer Verankerung und brachte sie in die Basis zurück.



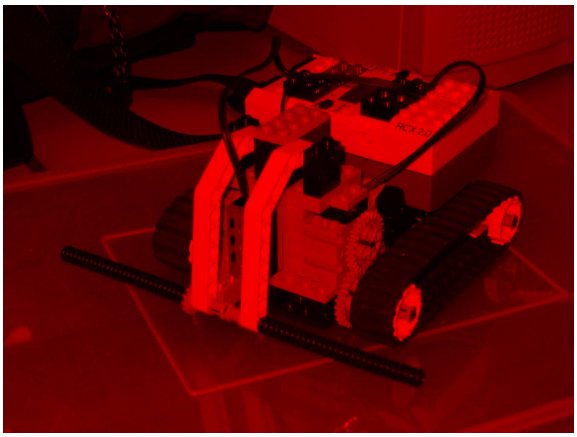
4.5.1: Arnold in Originalfarben

RGB-Ebenen Zuerst der simple Teil, die Zerlegung eines RGB-Bildes in seine drei Ebenen. Die Farben sollen dabei erhalten bleiben, die Ebenen sollen also in ihrer natürlichen Farbe dargestellt und nicht grau anonymisiert werden.

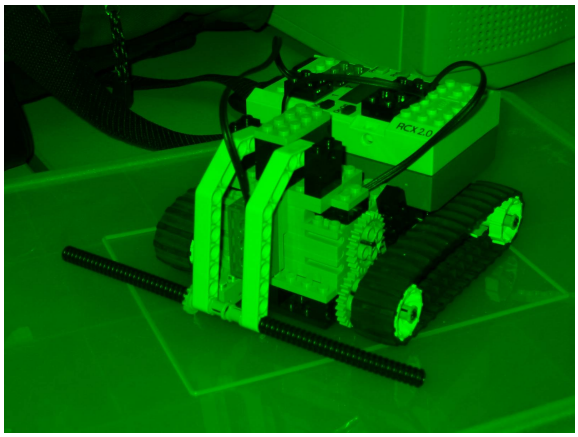
Ein Aufruf von

```
>zerleger.exe -rgb IMG_0331.jpg
```

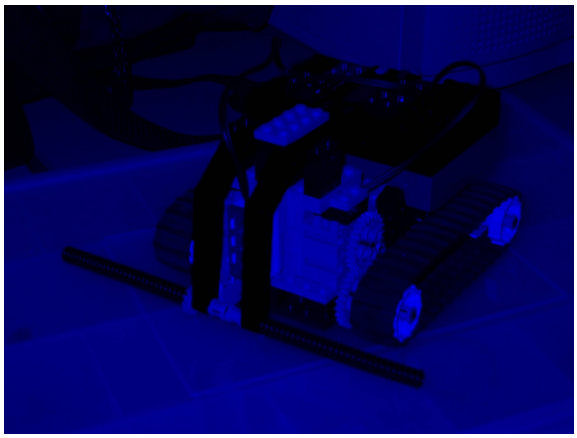
sorgt dafür, dass nach einem erfolgreichen Programmablauf drei weitere Dateien im gleichen Verzeichnis stehen (siehe Abbildung 4.5.2, 4.5.3 und 4.5.4)



4.5.2: IMG_0331_R.jpg



4.5.3: IMG_0331_G.jpg



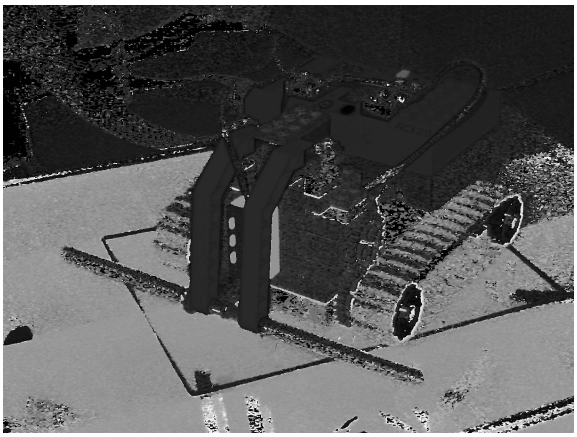
4.5.4: IMG_0331_B.jpg

HSV-Ebenen Eine Zerlegung in die einzelnen Ebenen des HSV-Farbmodelles erfolgt äquivalent zu der Zerlegung in RGB-Ebenen.

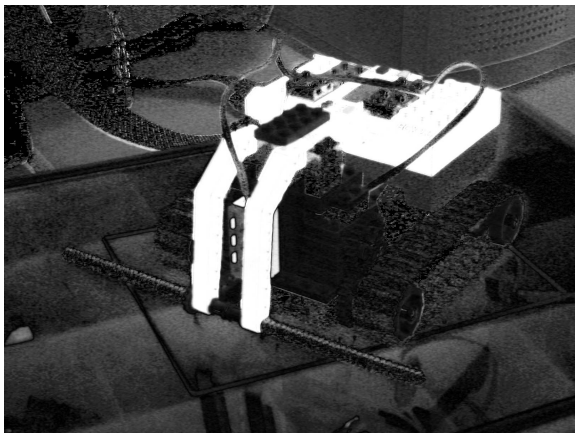
Ein Aufruf von

```
>zerleger.exe -hsb IMG_0331.jpg
```

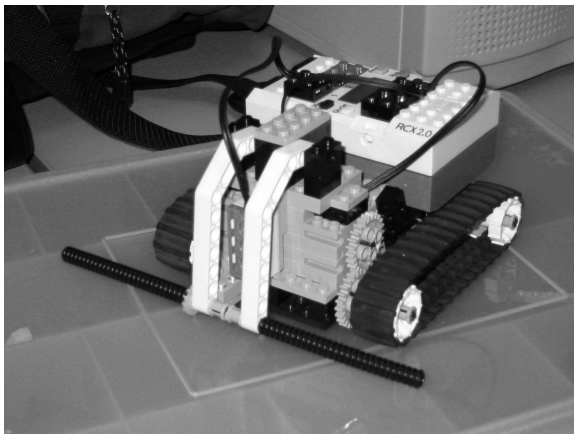
sorgt dafür, dass nach einem erfolgreichen Programmablauf drei weitere Dateien im gleichen Verzeichnis stehen. Zu Erinnerung: H steht für die Farbe, S für die Sättigung und V für die Helligkeit. Für das Ergebnis siehe Abbildung 4.5.5, 4.5.6 und 4.5.7.



4.5.5: IMG_0331_H.jpg



4.5.6: IMG_0331_S.jpg



4.5.7: IMG_0331_V.jpg

Farbe mit Toleranz markieren Setzen wir uns eine kleine Aufgabe:

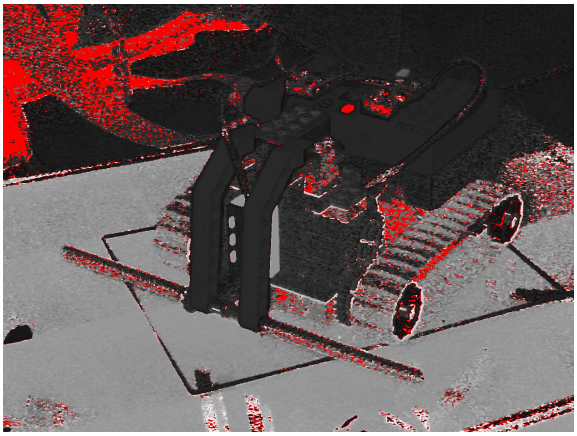
Ziel ist es, den roten und den grünen kleinen Knopf auf dem RCX einzeln zu selektieren. Außerdem soll die blaue Reflexlichtschranke vorne am Roboter maskiert werden. Eine erste Lösung ist es, die entsprechende Farbe mit einer Toleranz auf das Bild anzuwenden. Wie ich auf die Werte gekommen bin, wird in Kapitel 4.2 erklärt.

Drei Aufrufe sorgen für das Aufkommen dreier neuer Bilder im Verzeichnis:

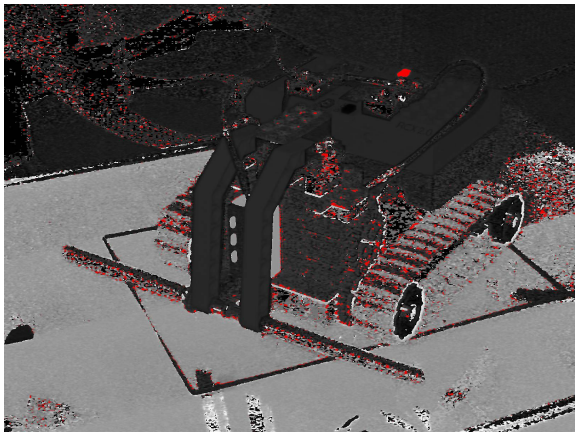
```
>zerleger.exe -h 0 10 IMG_0331.jpg
>zerleger.exe -h 85 20 IMG_0331.jpg
>zerleger.exe -h 160 20 IMG_0331.jpg
```

Der erste Wert steht für die Farbe im Bereich [0..255], der zweite gibt die Toleranz an. Eine Farbe von 100 mit einer Toleranz von 15 markiert also die Farben [85..115]. Da diese Werte einen geschlossenen Farbkreis bilden, schließt sich an die 255 wieder die 0 an. Rot (0) mit einer Toleranz von 10 bedeutet also ein Bereich [246..10].

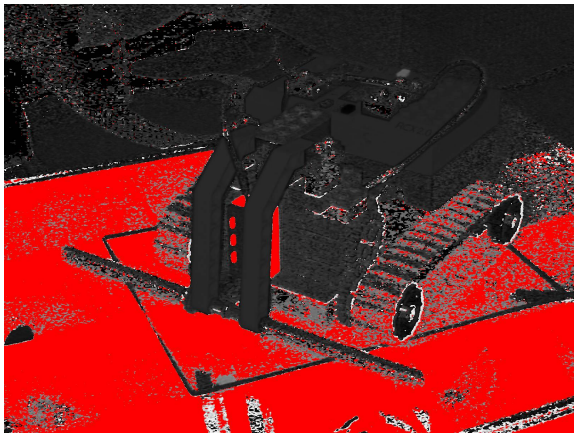
Die Ergebnisse der Aufrufe kann man sich in Abbildung 4.5.8, 4.5.9 und 4.5.10 verdeutlichen.



4.5.8: IMG_0331_H(h0t10).jpg



4.5.9: IMG_0331_H(h85t20).jpg

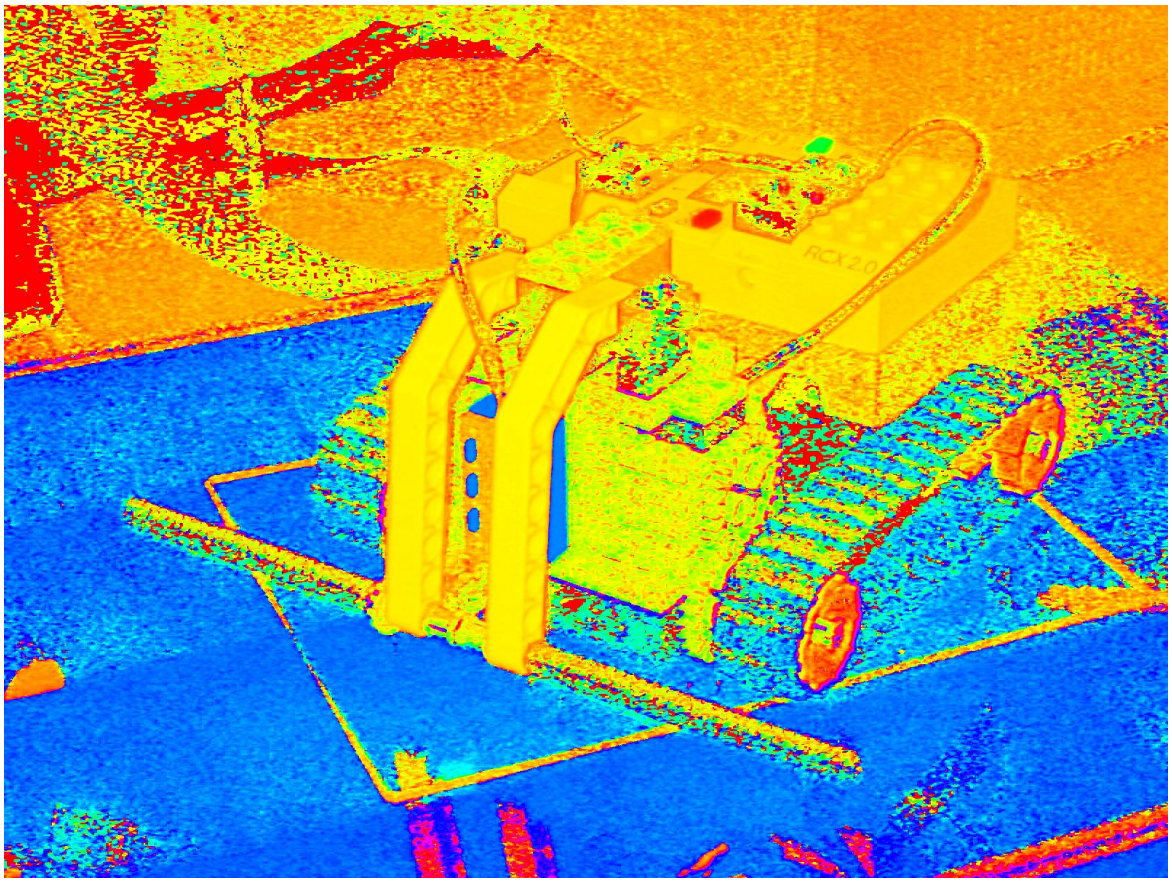


4.5.10: IMG_0331_H(h160t20).jpg

Reine Farben Im letzten Beispiel wurde sichtbar, dass auch eine Menge anderer Bereiche markiert wurden, die auf den ersten Blick gar nicht so recht in das Schema passen. Die Erklärung ist recht einfach:

Sehr dunkle Farben, sehr helle Farben und auch Bereiche dazwischen, also Grau mit einem leichten Farbstich, haben in dem HSV-Modell eine eindeutige Farbe. Eine Option, in der Sättigung und Helligkeit auf Maximum gestellt werden, gibt Aufschluss über den reinen Farbwert eines Bildes (vergleiche Abbildung 4.5.11)

```
>zerleger.exe -fc IMG_0331.jpg
```

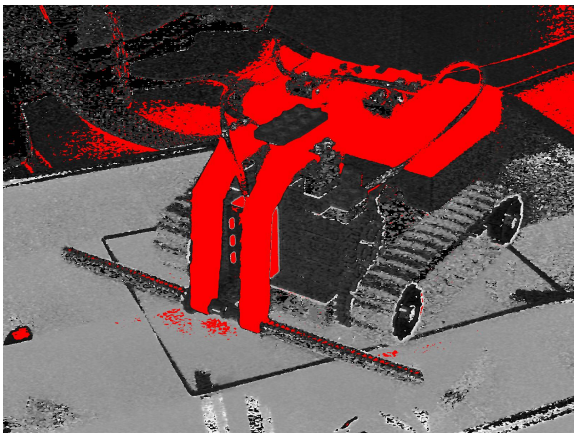


4.5.11: IMG_0331_FULLCOLOR.jpg

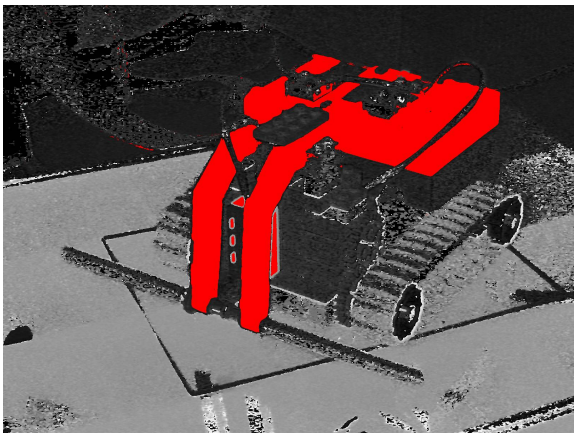
Sättigung mit Schwelle Unter den Gesichtspunkten des letzten Beispiels macht es Sinn, einen Blick auf die Sättigung zu werfen. Sättigung bedeutet, wie stark ein Pixel mit Farbe gesättigt ist. Schwarz, Weiß und Grau haben keinerlei Sättigung. Reines Rot als Beispiel hat eine Sättigung von 255. Um sich ansehen zu können, welche Bereiche im Bild mit einem Mindestwert gesättigt sind, werden nun folgende Aufrufe ausgeführt:

```
>zerleger.exe -s 100 IMG_0331.jpg
>zerleger.exe -s 200 IMG_0331.jpg
```

Die Ergebnisse finden sich in Abbildung 4.5.12 und 4.5.13 wieder.



4.5.12: IMG_0331_S(s100).jpg



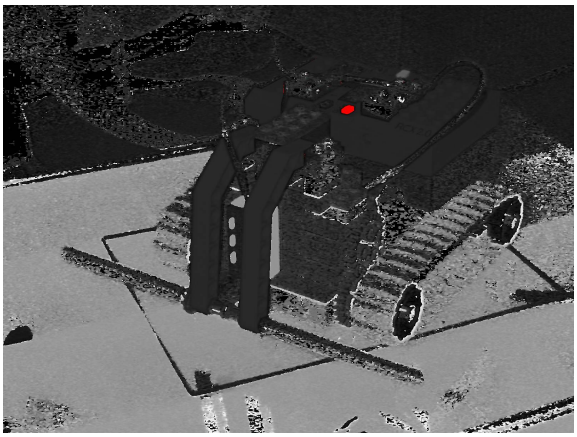
4.5.13: IMG_0331_S(s200).jpg

Wie man sehen kann, sind alle interessanten Bereiche selektiert, die falsch angezeigten hingegen nicht. Ein Heraufsetzen der Schwelle von 100 auf 200 hat das Ergebnis sogar noch verbessert.

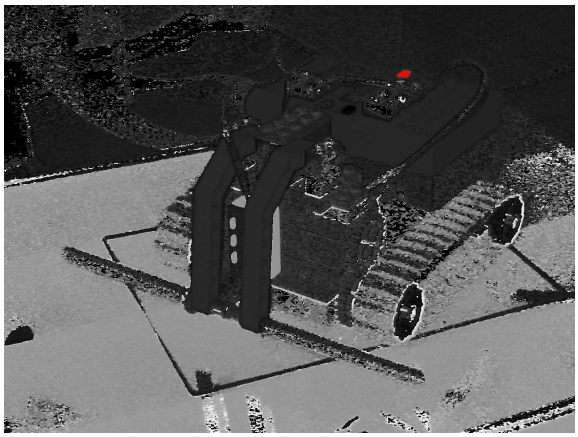
Farbe und Sättigung An dieser Stelle bleibt ein letzter Schritt, nämlich das Zusammenfassen der Ergebnisse aus den vorigen Kapiteln. Wenn die Farbe mit der Sättigung verknüpft wird, entsteht ein gutes Ergebnis. Das Programm unterstützt auch diese Möglichkeit. Der Aufruf gestaltet sich wie folgt:

```
>zerleger.exe -hs 0 10 200IMG_0331.jpg
>zerleger.exe -hs 85 20 200IMG_0331.jpg
>zerleger.exe -hs 160 20 200IMG_0331.jpg
```

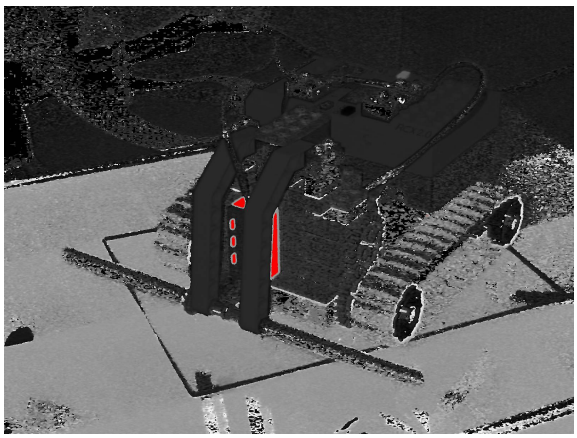
Die Ergebnisse stehen in Abbildung 4.5.14, 4.5.15 und 4.5.16.



4.5.14: IMG_0331_HS(h0t10s200).jpg



4.5.15: IMG_0331_HS(h85t20s200).jpg



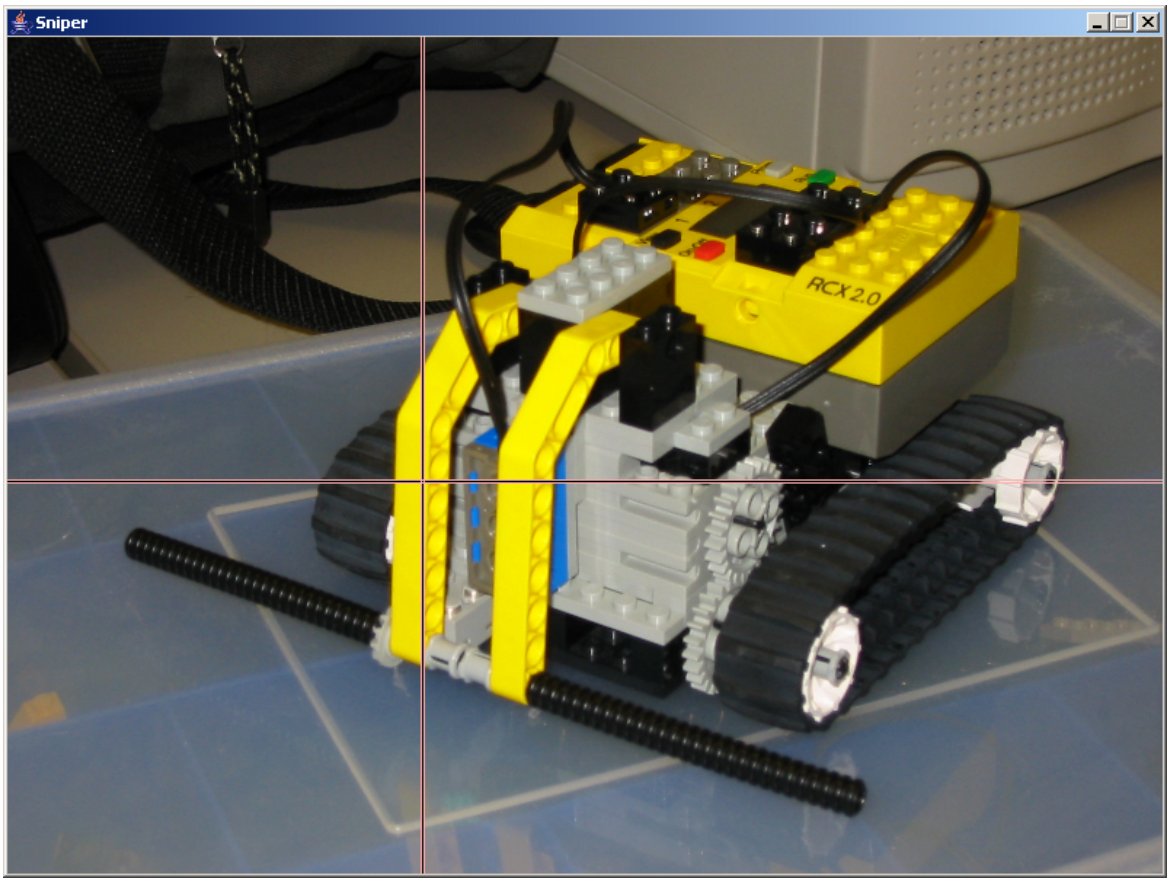
4.5.16: IMG_0331_HS(h160t20s200).jpg

Man kann erkennen, dass das Ergebnis exakt das ist, was wir uns als kleine Aufgabe vorgenommen haben.

Sniper Etwas, das fehlt, ist ein Programm, mit dem man in der Lage ist, schnell und einfach die Farbwerte von einzelnen Pixeln anzeigen zu können. Ein Aufruf von

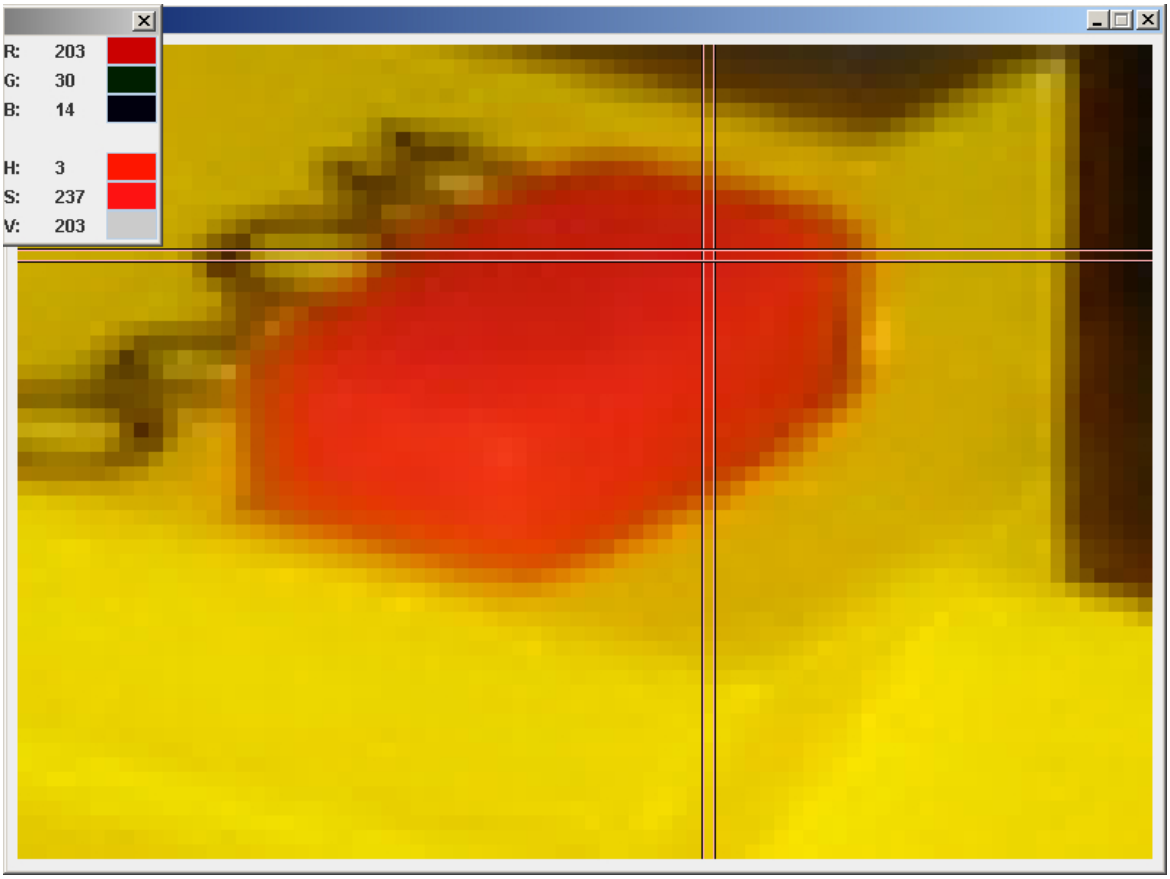
```
>zerleger.exe -sniper  IMG_0331.jpg
```

startet eine grafische Oberfläche (siehe Abbildung 4.5.17).



4.5.17: Hauptfenster

Ein Betätigen der `Strg`-Taste sorgt für einen Zoom in das Bild um den Mauszeiger herum (siehe Abbildung 4.5.18). Gleichzeitig wird ein Fenster eingeblendet, in dem das nun mit der Maus angewählte Pixel in die einzelnen Farbebenen aufgesplittet dargestellt wird.



4.5.18: Zoom

4.3 Hilfsmittel

An dieser Stelle möchte ich ein Hilfsmittel beschreiben, das nur indirekt mit der eigentlichen Arbeit verbandelt ist. Es geht um die Realisierung einer Bluetooth-Verbindung unter SuSE-Linux (Novell) als Ersatz für die beim RCube beiliegende serielle Leitung, welche immer eine physikalische Anbindung des RCubes an den Entwicklungsrechner verlangt.

Dieses Hilfsmittel entstand aus dem starken Wunsch meines Komillitonens Timo Storjohann heraus, welcher sich in seiner Diplomarbeit mit genetischen Algorithmen zur Steuerung des Roboters über den RCube befasste (Storjohann 2005).

Vor der Beschreibung des Aufbaus möchte ich noch einen kurzen Überblick über Bluetooth bieten, um deutlich zu machen, was die Möglichkeiten und was die Grenzen dieser drahtlosen Verbindung sind.

Bluetooth

Bluetooth ist ein Kurzstreckenfunk mit mäßiger Datenübertragungsgeschwindigkeit und kann dazu verwendet werden, sehr unterschiedliche Geräte in einem Piconetz zu verbinden.

Je nach Klasse beträgt die Reichweite 10m (Klasse 2) oder 100m (Klasse 1). Diese Angaben gelten für eine störungsfreie Umgebung im Freien und ohne Hindernisse. Mauern und elektromagnetisch strahlende Geräte können die Reichweite erheblich reduzieren.

Benannt wurde Bluetooth übrigens nach dem dänischen König Blatant. Ihm gelang es vor ca. 1000 Jahren, sein Volk mit dem von Norwegen zu vereinen.

Geschwindigkeit

Der derzeitig gültige Standard hat eine Übertragungsrate von 723,2Kbit/s brutto. Überlegt man, dass große Dokumente, welche an einen Drucker geschickt werden, mehrere Megabyte groß werden können, so wird schnell klar, dass auch Bluetooth gewissen Einschränkungen unterliegt. Die Übertragung von Audiodaten in Stereoqualität beispielsweise benötigt ohne Komprimierung bereits eine Bandbreite von

44100 Samples/s * 2 Kanäle * 16 Bit/Kanal = 1411,2KBit/s.

Das ist etwa doppelt so viel, wie eine störungsfreie und unbelastete Verbindung via Bluetooth liefern kann. Ein Bluetooth-Headset mit einer bidirektionalen Verbindung ist ohne Kompressionsalgorithmen nicht möglich. Von Bilddaten und Videostömen braucht also gar nicht erst

philosophiert werden, da die derzeitige Geschwindigkeit solcherlei Gedanken im Keim erstickt.

Kanäle

Bluetooth arbeitet im ISM-Band (**I**ndustrial, **S**cientific, **M**edical), die Frequenz beträgt 2,4 GHz und kann weltweit lizenzfrei genutzt werden. Die Symbolrate beträgt 1MBit/s. Der Kanal ist in Slots aufgeteilt, welche eine Länge von 625μs besitzen.

Bluetooth unterstützt folgende Möglichkeiten zur Datenübertragung:

1. Ein asynchroner Datenkanal
2. Maximal drei gleichzeitige Sprachkanäle
3. ein Kanal mit gleichzeitiger asynchroner Daten- und synchroner Sprachübertragung

Jeder Sprachkanal besitzt einen synchronen Kanal mit 64Kbit/s in jede Richtung. Der asynchrone Datenkanal kann maximal 723,2KBit/s asymetrisch und bis zu 57,6KBit/s in die entgegengesetzte Richtung übertragen, symmetrisch sind es 433,9KBit/s in beide Richtungen.

Netz

Bluetooth unterstützt Punkt-zu-Punkt-Verbindungen, kann aber auch mit mehreren Geräten gleichzeitig kommunizieren. Es gibt pro Verbindung zwei Möglichkeiten, nämlich die Betriebsart als Master oder als Slave.

Ein Master kann in einem Piconet maximal sieben aktive Slaves haben. Ist die Anzahl der Slaves größer, so werden sie im Parked-State gehalten. Sie sind dann mit dem Master synchron, sind aber nicht aktiv. Der Kanalzugang der Slaves wird durch den Master gesteuert.

Überlappen sich mehrere Piconets, entstehen sogenannte Scatternets. Ein Master aus einem Piconet kann durchaus als Slave in einem anderen Piconet auftreten. Die einzelnen Piconets verhalten sich dabei nicht zeit- oder frequenzsynchron zueinander.

Protokollstack

Die Bluetooth Special Interest Group (BSIG) hat mit der Bluetooth Spezifikation 1.0 einen Standard geschaffen, der es den Entwicklern ermöglichen soll, problemarm miteinander entwickeln zu können.

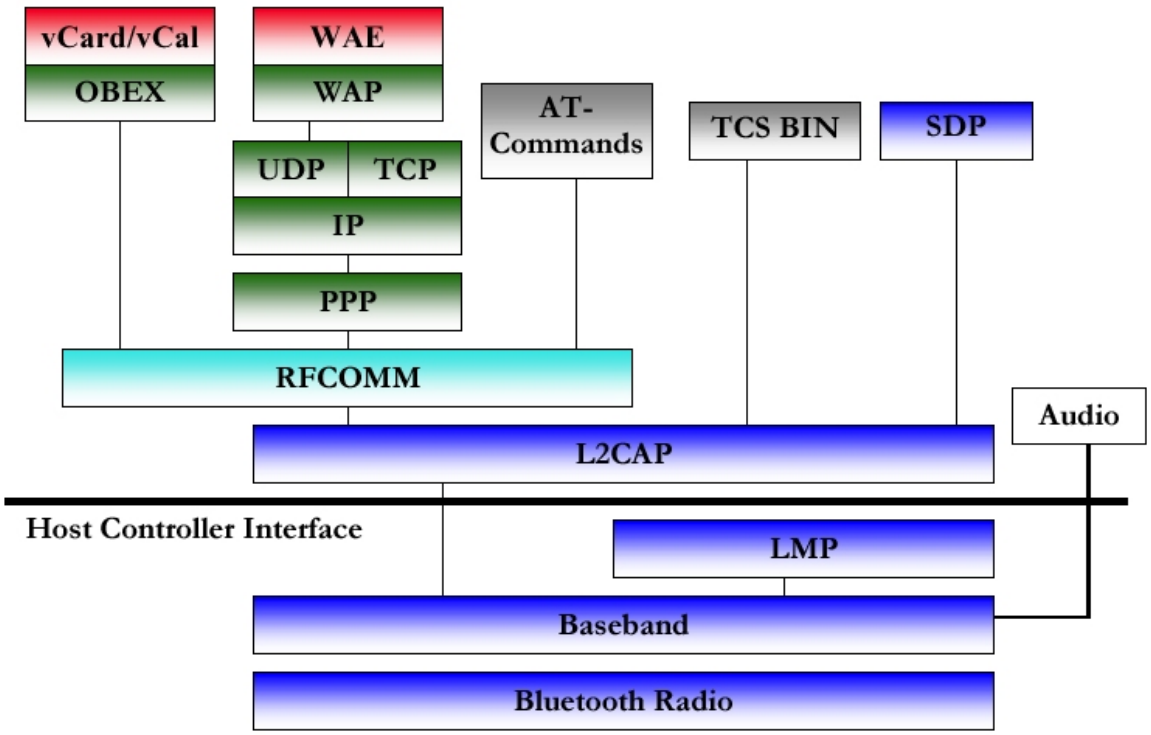


Abbildung 4.5: Bluetooth Protokollstack

Über die Schnittstellen RFCOMM und L2CAP setzen Protokolle der höheren Ebene wie z.B. IP, WAP oder Kartenterminalanwendungen auf. Die Anwendungen laufen dabei über ein oder mehrere Kanäle des Protokoll-Stacks. Der Grundstack, nämlich bis RFCOMM, wurde so entwickelt, dass es problemlos möglich ist, bereits vorhandene Protokolle darauf aufzusetzen, um eine maximale Kompatiblität zu erreichen.

Der folgende kurze Überblick beschreibt, welche Aufgaben die einzelnen Komponenten haben:

Basisband:
Das Basisband realisiert die physikalische HF-Verbindung zwischen den einzelnen Stationen in einem Piconet.

Audio:
Für Kopfhöreranwendungen gibt es ein PCM-Rohdatenformat, welches im Direktmodus an allen Protokollen vorbeigeführt werden kann (SCO).

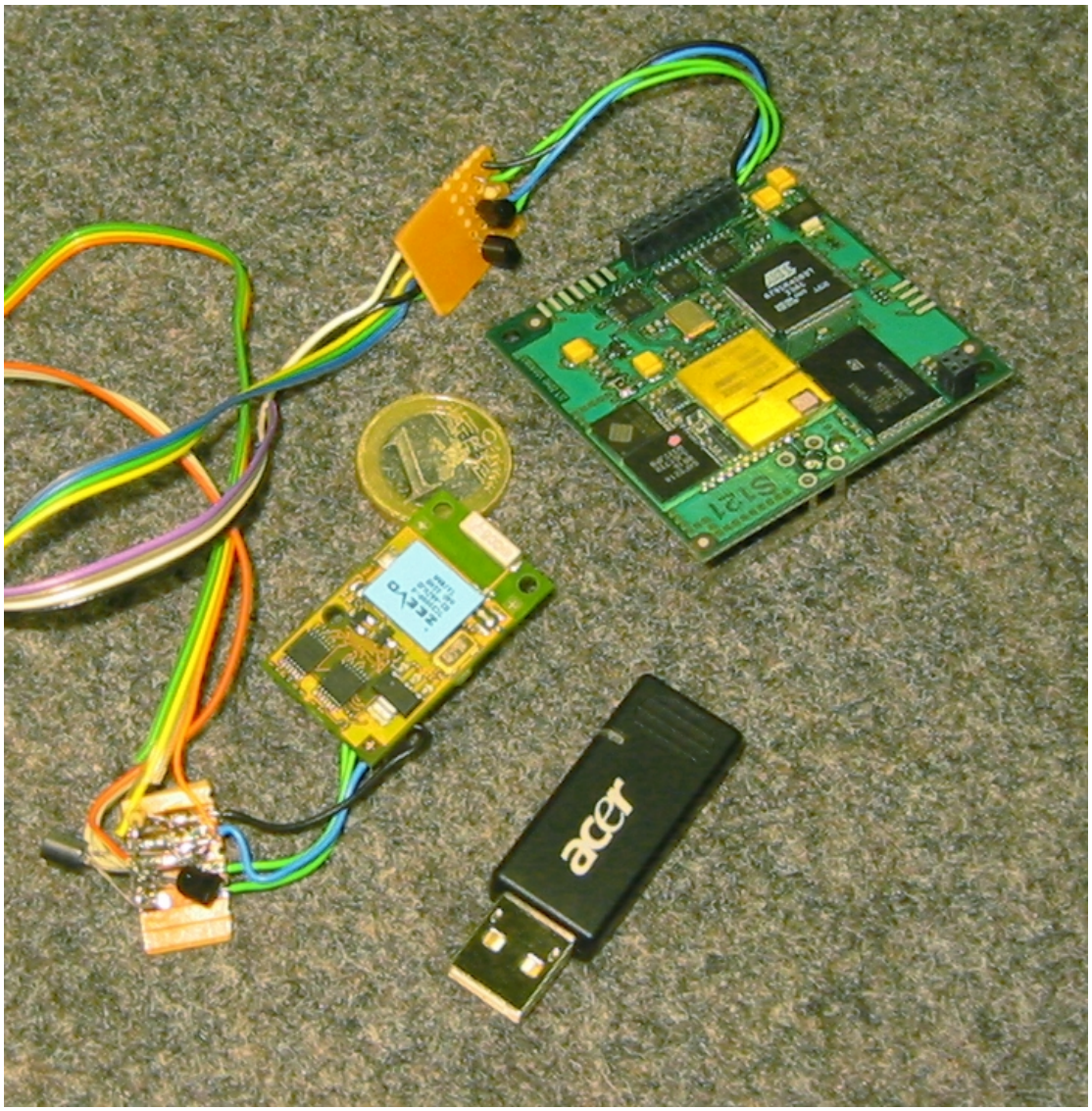
LMP (Link Manager Protocol):
Das LMP ist verantwortlich für den Verbindungsaufbau und die Sicherheit von Bluetooth-Links.

L2CAP (Logical Link Control and Adaptation Protocol):
Dieses Protokoll verbindet die höherliegenden Schichten des Stacks mit dem darunter liegenden Basisband. L2CAP bietet verbindungsorientierte und verbindungslose Services an

RFCOMM (Cable Replacement Protocol):
Dies ist ein Protokoll zur Emulation einer seriellen RS-232-Kabelverbindung über eine drahtlose Verbindung. RFCOMM wird von höherliegenden Protokollschichten genutzt, die Transportkapazität über eine serielle Leitung benötigen.

Hardware

Um sowohl den zeitlichen als auch finanziellen Rahmen möglichst wenig zu beanspruchen sollte - sofern möglich - auf vorhandene Hardware zurückgegriffen werden. Erfreulicherweise fanden sich Bluetooth-Dongles für den USB-Port des PCs an. Ebenso waren zwei unterschiedliche Bluetooth-to-Serial-Module von Stollmann (Stollmann) verfügbar. Da die Module zur Verbindung mit einem RS232-Treiber vorgesehen sind (welche die Signale invertieren), mussten sie mit einem fliegenden Aufbau aus zwei Transistoren und einem Widerstand verdrahtet werden. Diese kleine Schaltung sorgte dann für eine Invertierung der Datenleitungen und verhalf so zu einer einwandfreien Kommunikation zwischen dem RCube und den Stollmann-Modulen. Da zwei Module verfügbar waren, baute ich auch beide um. In Abbildung 4.6.1 ist links unten der erste Prototp mit einer etwas labilen Verdrahtung des Inverters zu sehen. Rechts oben liegt die zweite Version mit einem etwas stabiler verdrahteten Inverter auf einer Lochrasterplatine. Unten in der Mitte ist der verwendete USB2Bluetooth-Wandler zu sehen.



4.6.1: Stollmann Bluetooth-to-Serial Module

Software

Um das Bluetooth-Dongle ansteuern zu können, waren noch einige Bibliotheken nötig. Die Software zur Datenübertragung ist die gleiche, als wenn die Daten über Kupferadern fließen. Der einzige Unterschied ist der, dass als serielle Schnittstelle die virtuelle serielle Schnittstelle der Bluetooth-Treiber eingestellt werden muss. Ein Tutorial, in dem auch näher darauf eingegangen wird, welche Bibliotheken nötig sind, wie sie eingebunden werden und wie die Verbindung zum Bluetooth-to-Serial-Modul aufgenommen wird, findet sich im Anhang dieser Arbeit (siehe Kapitel 6.4).

4.4 RCube basierte Serviceschicht

Das Ziel dieser Programmbibliothek war es, eine möglichst simple Schnittstelle zur Funktionalität der Farberkennung zur Verfügung zu stellen. Sie sollte zugleich performant, komfortabel und leicht erweiterbar sein. Meist widersprechen sich diese Forderungen, so dass ein Kompromiss zumindest zwischen Komfort und Performance gefunden werden muss.

Design

Da auf jeden Fall das Lesen und Schreiben von Bilddaten gekapselt werden sollen, es aber keinen Sinn macht, pro Funktion (Torerkennung, Ballerkennung, . . .) ein Bild zu lesen, musste der Framegrabber von der Grafikverarbeitung getrennt werden. Selbiges gilt natürlich auch für die Bildausgabe.

Die Einzelkomponenten werden hinter einer Fassade[5] (siehe Abbildung 4.6 (Sedgewick 2002)) versteckt.

[5] *Fassade (Entwurfsmuster, vgl. Wikipedia Facade)*
Das in der Softwareentwicklung eingesetzte Entwurfsmuster Fassade (engl. facade) bietet eine einheitliche und oft vereinfachte Schnittstelle zu einer Menge von Schnittstellen eines Subsystems.

Wenn ein Subsystem viele selten von außen verwendete technisch orientierte Klassen enthält, hilft es, eine Fassade zu verwenden. Die Fassade ist eine Klasse mit ausgewählten Methoden, die eine häufig benötigte Untermenge an Funktionalität des Subsystems umfasst. Sie vereinfacht den Umgang mit dem Subsystem.

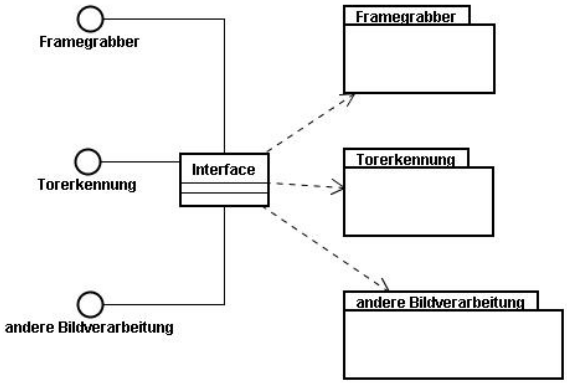


Abbildung 4.6: Fassade

Das Subsystem Framegrabber verwendet die V4L2[6] (Knorr) Schnittstelle. Diese muss inizialisiert werden, um festzulegen, auf welche Weise die Bilddaten in den Speicher gelangen sollen. Es gibt zwei Möglichkeiten, Bilddaten zu capturen. Auf der einen Seite kann die Hardware nonstop Bilder in den Hauptspeicher schreiben, auf der anderen Seite können Bilder bei Bedarf gecaptured werden.

Die erste Lösung hat den Vorteil, dass immer aktuelle Daten bereitliegen. Bereits im Kapitel über den Algorithmus (siehe 3.3) wurde allerdings über die nicht sehr üppige Datenbusbandbreite des RCubes gesprochen. Diese wird beim Streaming mit Daten belastet, die unter Umständen irrelevant sind, die Performance aber dennoch verringern.

Einzelne Bilder zu capturen hat den Nachteil, dass es kein aktuelles Bildmaterial gibt. Es muss gewartet werden, bis das Eingangssignal ein neues Bild beginnt und es komplett eingelesen wurde. Das bedeutet, dass im unglücklichsten Fall (wenn kurz vor der Anfrage nach einem neuen Bild begonnen wurde) bei 25Hz Vollbildfrequenz knapp 80ms gewartet werden muss, bis Bildmaterial verfügbar wird.

Ich entschied mich für letztere Lösung, um den Datenbus möglichst wenig zu belasten. Ob das auch für die Torerkennung an einem Roboter im Wettkampf ausreichen ist, wird die Praxis zeigen. Durch Multithreading kann die Wartezeit auf ein Bild für andere Aufgaben genutzt werden, zur Strategieberechnung zum Beispiel.

Zum Debuggen sollen Bilddaten auch wieder ausgegeben werden können. Auch das muss die Framegrabberkomponente unterstützen[7].

[6] Video for Linux two

[7] An dieser Stelle könnte darüber diskutiert werden, ob der Name Framegrabber für die Komponente korrekt ist oder Videointerface besser wäre. Die absolute Hauptaufgabe liegt allerdings im Capturen bzw. Grabben von Bildern, die Ausgabe ist als nachträgliche Randerscheinung für das Debugging zu bewerten.

Nach Beendigung der Anwendung oder vor einer längeren Pause sollte die Videoschnittstelle geschlossen werden:

Einerseits um Ressourcen auf softwaretechnischer Seite wieder freizugeben, andererseits benötigt die Videohardware elektrische Energie, mit der bei autarken Systemen möglichst sparsam umgegangen werden muss.

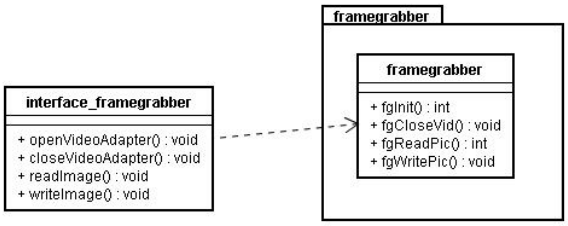


Abbildung 4.7: Framegrabber

Die Torerkennung hingegen ist aufwändiger als der Framegrabber. Der Algorithmus selbst wurde in der 'Klasse' `torerkennung` implementiert.

Um nicht auf den rohen Bilddaten arbeiten zu müssen, wird von ihnen abstrahiert. Die erste Abstraktionsebene gegenüber dem simplen Array mit je drei `char` für die drei RGB-Grundfarben ist eine 'Klasse' namens `img`. Mit ihr werden lesende und schreibende Pixeloperationen ausgeführt.

Da der Algorithmus nicht auf Basis von Pixeln arbeitet, werden sie mit Hilfe der nächsten Abstraktionsebene hinter Blöcken versteckt. Auch die Farbmodellkonvertierung und Entscheidung auf 'richtige Farbe'/'falsche Farbe' ist hier verborgen.

Die 'Klasse' `cachedimg` stellt die nächste Ebene dar, sie dient ausschließlich zur Performancesteigerung. Wie der Name andeutet, werden Abfragen auf Blöcke gecached, damit aufwendige Berechnungen so selten wir möglich durchgeführt werden müssen.

Der Cache selbst ist nicht in `cachedimg` enthalten, sondern ist eine eigene 'Klasse' mit Namen `cache`. Sie stellt eine bequeme Schnittstelle zu den Daten im Cache zur Verfügung

Die 'Klasse' `imgtools` ist für alle Operationen zuständig, die mehrfach gebraucht werden oder keinen festeren Zusammenhang mit einer bestimmten Verarbeitung haben.

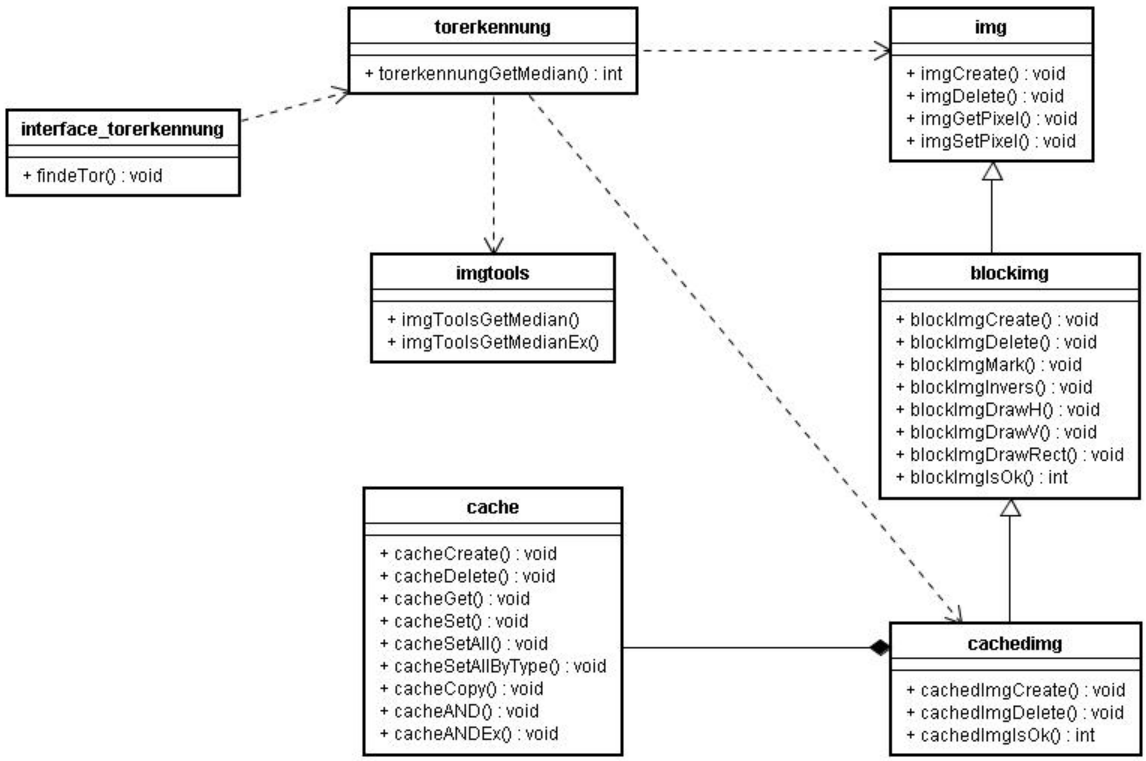


Abbildung 4.8: Torerkennung

Schnittstelle

An dieser Stelle wird die Schnittstelle kurz erläutert werden:

```
int
openVideoAdapter();
```

initialisiert den Videoadapter. Wird 0 zurückgeliefert, so traten keine Fehler auf.

```
void
closeVideoAdapter();
```

schließt den Videoadapter.

```
int
readImage(
  PIMAGE pPImage);
```

lässt sich einen gefüllten Puffer aus dem Framegrabber liefern und schreibt das Ergebnis in das übergebene `IMAGE`. Wird 0 zurückgeliefert, hat alles geklappt.

```
int
writeImage();
```

schreibt die Daten des internen Puffers (er wird über `readImage` zugreifbar) auf die Bildausgabe. Wird 0 zurückgeliefert, gab es keine Fehler.

```
int
findeTor(
  PIMAGE pPImage,
  PPIXELMATCH pPPixelMatch,
  int pBlockSizeX,
  int pBlockSizeY,
  int pDebug,
  int *pX,
  int *pY);
```

Sucht nach einem Farbfleck (speziell: dem Tor).

Es muss ein Zeiger auf ein `IMAGE` als Berechnungsgrundlage mitgeliefert werden (`pPimage`).

Die PIXELMATCH-Struktur beinhaltet min- und max-Werte für Farbe, Sättigung und Helligkeit.

Die beiden Blocksize-Parameter geben die Ausdehnung eines Blockes an.

Wird für `pDebug` ein Wert ungleich 0 übergeben, so erzeugt das Verfahren eine Ausgabe und modifiziert dabei die Quellbilddaten.

In denen als Zeiger übergebenen Variablen `pX` und `pY` werden die Rückgabekoordinaten geschrieben. Sie sind allerdings nur gültig, wenn als Rückgabewert der Funktion eine 1 geliefert wurde. Bei einer 0 konnte keine Fläche erkannt und somit auch kein Median berechnet werden.

Implementierung

Die Implemtierung hält sich strikt an das Design[8]. Der Framegrabber wurde aus einem Beispiel der RCube-CD entlehnt und modifiziert, um eine möglichst schmale Schnittstelle zu erreichen, die aber trotz allem noch einen möglichst großen Teil der Funktionalität abdeckt.

Da die bereits beschriebene Art des Grabbings (es gibt Bilddaten nur auf Anforderung) verwendet wird, müssen Bildpuffer erzeugt werden. Es muss mindestens einer vorhanden sein, vier sind maximal erlaubt.

Intern existieren zwei Queues. Die erste wird mit zu füllenden Bildpuffern gefüllt, in die zweite werden die mit frischen Bilddaten beschriebene Puffer geschoben und können von der Anwendung abgeholt werden. Soll nun ein Puffer wieder zum Füllen freigegeben werden, so muss er wieder explizit in die erste Queue gestellt werden.

In dieser Implementierung wird mit nur einem Puffer gearbeitet, was sich als ausreichend herausgestellt hat (siehe Kapitel 4.5, Proof Of Concept). Bei der Initialisierung wird der Puffer bereits in die Wartequeue gestellt. Ein `fgWrite` sorgt dafür, dass die Daten aus dem Puffer in den Ausgabespeicher der Videohardware kopiert werden. Außerdem wird der Puffer als leer markiert und wieder in die Wartequeue gestellt.

Ein Aufruf der Interfacemethode zum Lesen von Bilddaten liefert ein `img`-Objekt zurück. Wie schon erwähnt, ist die Anwendung objektorientiert-ähnlich geschrieben. Die Objektorientierung bietet diverse Aspekte, die sich in C nicht alle simulieren lassen, auch nicht per Definition[9]:

Ein Objekt kapselt normalerweise Daten in seinem Inneren. Ebenfalls im Objekt enthaltene Methoden bieten einen kontrollierten Zugriff auf die Daten. Es kann also Objekte vom gleichen Typ mit unterschiedlichen Daten geben. Da sich die Funktionen nicht ändern, sondern nur die Daten für den Zustand eines Objektes verantwortlich sind, kann man die Instanziierung eines Objektes simulieren.

Da es in C möglich ist, Datenstrukturen als Typ zu definieren, Speicher dynamisch zu alloziieren und auch wieder freizugeben, kann man auf diese Weise das Instanziieren und Löschen von Objekten simulieren.

Bei jeder Funktion dieses Objekts muss ein Zeiger auf eine ihm zugehörige Datenstruktur mit übergeben werden:

[8] Trotz der Verwendung von Klassendiagrammen ist die Library in C geschrieben und somit nicht objektorientiert. Allerdings ist es möglich, in C objektorientiert-ähnlich zu entwerfen. Daher ist es durchaus gerechtfertigt, dieses Modelierungswerkzeug einzusetzen, um die Anwendung zu designen

[9] Zugriff über `public` oder `private` sind Beispiele für Modifier, die in C nur per Definition verwendet werden können, etwa durch bestimmte Kommentare

```
typedef struct
{
  int sizeX;
  int sizeY;
  int sizePixel;

  unsigned char* pData;
} IMAGE;
typedef IMAGE* PIMAGE;

void
imgCreate(
  PIMAGE pPImgData,
  unsigned char* pPData,
  int sizeX,
  int sizeY);

void
imgDelete(
  PIMAGE pPImgData);

PPIXEL_BGR
imgGetPixel(
  PIMAGE pPImgData,
  int pPosX,
  int pPosY);

void
imgSetPixel(
  PIMAGE pPImgData,
  int pPosX,
  int pPosY,
  PPIXEL_BGR pPPixelBgr);
```

Eine Instanz der Datenstruktur wird von Hand erzeugt, ein anschließender Aufruf von `imgCreate` mit der neuen Datenstruktur sorgt für deren Inizialisierung. Es ist auf diese Weise nicht möglich, eine Inizialisierung der Daten zu garantieren, da es quasi manuell geschehen muss und somit vergessen werden kann. Eine Kapselung der Daten ist ebenfalls nicht vorhanden, da auf die externen Strukturen ohne weiteres zugegriffen werden kann.

Alle Definitionen und Funktionen einer ’Klasse’ befinden sich in der selben .c/.h Datei, um ihre Zugehörigkeit zur selben ’Klasse’ zu symbolisieren.

Auch ein weiterer Aspekt der Objektorientierung, nämlich Ableitung bzw. Vererbung ist möglich.

```
typedef struct
{
  BLOCKIMAGE blockImage;
  CACHE cache;
}CACHEDIMAGE;
typedef CACHEDIMAGE* PCACHEDIMAGE;

void
cachedImgCreate(
  PCACHEDIMAGE pPCachedImage,
  PBLOCKIMAGE  pPBlockImage,
  int pBlocksizeX,
  int pBlocksizeY);
```

Dazu muss im Konstruktor das Objekt der Elternklasse mitgegeben werden. Somit beinhaltet die neue Datenstruktur sowohl die alten als auch die neuen Daten.

Ein Problem ist leider nicht ohne größere Umstände zu lösen:

Polymorphismus, also das Überladen und Überschreiben von Funktionen. Sicherlich könnte man sich Konstruktionen mit Funktionspointern ausdenken, welche auch diese Anforderungen erfüllen könnten. Da aber auch die Performance im Vordergrund steht und nicht die Erwartung, in C objektorientiert zu designen, wurde auf eine Implementierung dieses Aspektes verzichtet.

4.5 Proof Of Concept

Die Überprüfung der prinzipiellen Funktionsfähigkeit sollte zwei Fliegen mit einer Klappe schlagen. Zum einen sollte gezeigt werden, dass die Serviceschicht unter den geforderten zeitlichen Rahmenbedingungen prinzipiell funktionfähig ist, zum anderen auch, dass sich das Ergebnis in einer realen Echtzeitanwendung als tauglich zeigt und somit als Teil einer Roboter-Experimentierumgebung dienen kann.

Design

Die Idee ist, ein bewegliches und einfarbiges Objekt mit einer Kamera über zwei Achsen zu verfolgen[10].

Auf der Softwareseite sollte die Serviceschicht der Torerkennung verwendet werden. Das Konzept ist simpel:

Da die Kamera immer auf den Mittelpunkt des zu erkennenden Flecks schaut, muss ihre Position entgegen der Abweichung korrigiert werden.

Da diese Rückkopplung eine Regelung darstellt, werden gleichzeitig auch alle Probleme mitgeliefert, die eine einwandfreie Funktion beeinträchtigen können. Im Klartext bedeutet dies, dass ein Kompromiss zwischen Geschwindigkeit, Präzision und Überschwingen gefunden werden muss. Eine langsame Nachführung bietet eine größere Präzision und vermeidet ein Überschwingen, ist aber eben auch langsam und träge. Bei einer schnellen Nachführung kommen die physikalischen Eigenschaften aller Komponenten zum Tragen: Die Regelung fängt an zu Überschwingen und beginnt ab einem bestimmten Punkt auch zum Aufschaukeln der Schwingungen zu neigen, so dass die Kamera unaufhörlich um die Mitte des Farbfleckzentrums pendelt.

Realisierung

Ein Metronom aus dem Musikzubehör und ein Stück aufgeklebter Pappe dienten als ein eigenständig bewegliches Ziel. Die Kamera selbst wurde an zwei Modellbauservos befestigt, eines für die Drehung, ein anderes zum Kippen.

Da LART und AkSen über den CAN-Bus verbunden sind und dieser gleichzeitig die einzige Möglichkeit zum Datenaustausch bietet, musste ein Interface erstellt werden, welches auf

[10] Die Anwendung wurde als Blickfang für eine Präsentation auf einer hausinternen Messe der HAW konzipiert.

dem LART-Board das Senden von Positionswerten, auf dem AkSen-Board dagegen das Einlesen von Positionswerten ermöglicht. Ein Beispielprogramm vom RCube konnte dafür zum Teil verwendet werden.

Das Programm auf dem AkSen läuft in einer Endlosschleife, liest den CAN-Bus aus und schreibt die Werte auf die Servo-Ports, zum Debuggen zusätzlich noch auf das LCD.

Das LART-Programm läuft ebenfalls in einer Endlosschleife. Es ruft die Bibliothek auf, um ein Bild zu lesen, ruft die Torerkennung auf, um den Mittelpunkt zu finden und zieht das Quadrat der Abweichung vom Mittelpunkt von internen Variablen ab bzw. zählt sie dazu. Das neue Ergebnis der internen Variablen wird zum AkSen und somit an die Servos gesendet, welche daraufhin die Position der Kamera korrigieren.

Es wurde das Quadrat der Abweichung gewählt, um einerseits eine hohe Regelgeschwindigkeit am Rande des Blickfeldes zu erreichen, bei Zentrumsnähe aber nur noch kleine Korrekturbewegungen zu vollziehen. Somit ist ein sehr dynamisches Verhalten zu beobachten, ohne dass Probleme wie Überschwingen oder Instabilitäten wie Oszillieren auf unangenehme Weise zu Tage treten.

Die Funktion der Torerkennung unter den gegebenen Rahmenbedingungen konnte auf diese Weise eindrucksvoll bestätigt werden.

Als äußerst lästig stellte sich das automatische Einschalten des BLANKs[11] 10 Minuten nach der Inbetriebnahme des LARTs oder nach einem RESET heraus. Es wäre angebracht, den Kernel beispielsweise so zu modifizieren, dass die Videoausgabe von Grund auf ausgeschaltet ist. Die jeweilige Anwendung - so sie eine Ausgabe benötigt - sorgt dann für das Ein- und Ausschalten.

Die Abbildungen 4.9 und 4.10 zeigen den Aufbau in Detailansicht bzw. komplett.

Der breite dunklere Streifen auf dem Ausgabemonitor ist das Ergebnis des Elektronenstrahls der Monitorröhre in Verbindung mit einer kurzen Belichtungszeit der Digicam (auf Grund der hohen Umgebungshelligkeit).

[11] *BLANK*
Ausschalten der Videoausgabe zur Verringerung des Stromverbrauchs. Üblicherweise verwenden autonome Roboter keine Videoausgabe, sie würde nur unnötig die Akkus belasten.

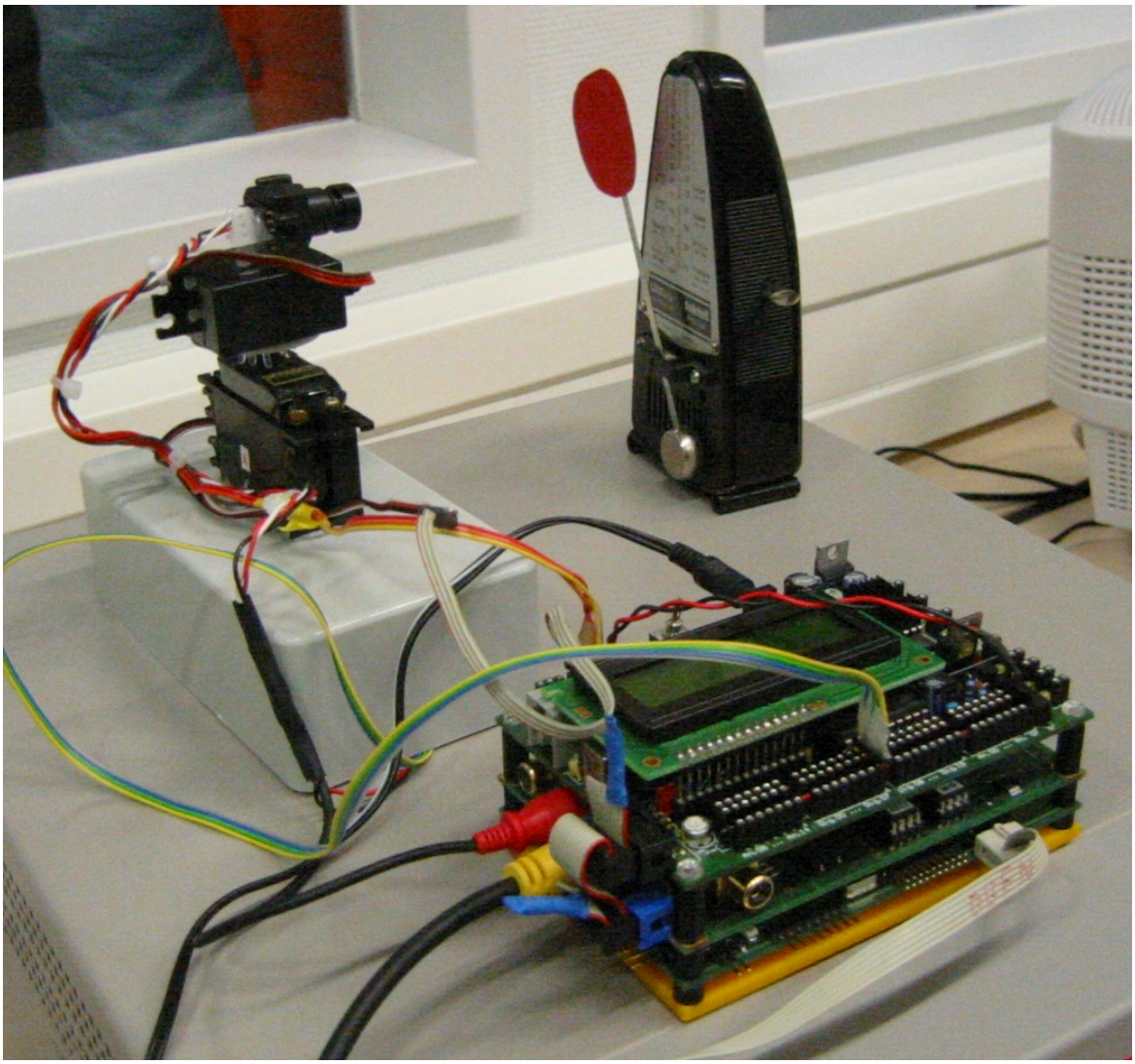


Abbildung 4.9: Der Aufbau

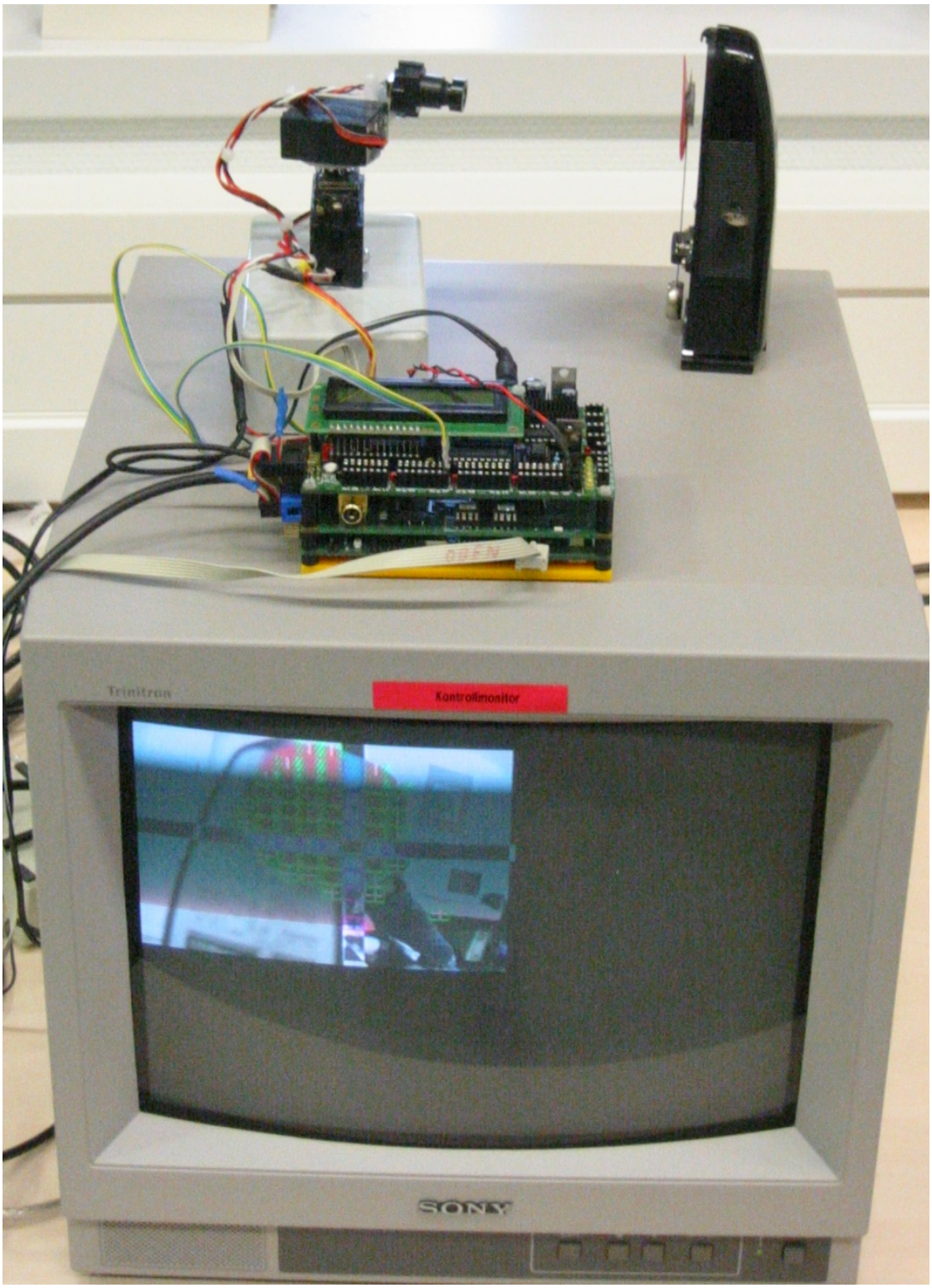


Abbildung 4.10: Während der Arbeit

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Ziel dieser Arbeit war es, eine Komponente zu entwickeln, die unter den Rahmenbedingungen der Echtzeit auf dem RCube eine Bildanalyse, genauer gesagt, eine Torerkennung, bewerkstelligen kann. Dieses Ziel wurde mit der vorliegenden Arbeit erreicht.

Dazu wurde ein festgelegtes Szenario - das Spielfeld des RoboLabs der HAW-Hamburg - als Basis für die Analyse verwendet. Es wurde ein Algorithmus entworfen und implementiert, der zum einen generell in der Lage ist, einen zusammenhängenden Farbfleck in einem Videostream zu finden und dessen Zentrum zu bestimmen. Zusätzlich ist er in der Lage, für den Spezialfall (ein Gegner steht vor dem Tor und teilt es optisch in zwei Hälften) ein gutes Ergebnis zu erstellen.

Zusätzlich entstand eine Workbench, um beliebige Verfahren der Bildverarbeitung einfach implementieren und sie hinsichtlich ihrer Qualität beurteilen zu können.

Als weiterer Zusatz wurde eine Sammlung von Tools zur Bildanalyse und Dokumentation entwickelt. Diese enthalten Methoden, die in herkömmlichen Bildverarbeitungsprogrammen nicht oder nur sehr kompliziert nachzuvollziehen sind.

Eine kleine Anwendung, nämlich ein Objekttracker auf Basis der Farberkennung, rundet das Paket ab und beweist eindrucksvoll die Praxistauglichkeit dieser Lösung.

5.2 Fazit

Mit dieser Arbeit konnte gezeigt werden, dass nicht nur den großen und leistungsfähigen Systemen das Sehen möglich ist, sondern im eingeschränkten Rahmen auch den kleinen, langsamen und energiesparenden.

Ein wichtiger Aspekt ist dabei allerdings die Optimierung. Standardoperatoren der Bildverarbeitung können nicht verwendet werden. Jede Lösung ist speziell auf die verwendete Plattform angepasst.

Sehr vorteilhaft wäre eine Verbesserung der Hardware des RCubes. Auf der einen Seite in Form einer leistungsfähigeren CPU, wie sie auch in modernen PDAs arbeitet (derzeit liegt der obere Bereich der Taktfrequenz bei etwa 500-600MHz). Auf der anderen Seite hingegen wäre ein mathematischer Coprozessor eine gute Lösung, um schneller präzise Berechnungen durchführen zu können. Eine weitere Seite wäre die Implementierung anderer Farbmodelle wie z.B. HSV in das VIO-Board. Dadurch kann der CPU sehr viel Arbeit abgenommen werden. Prinzipiell wäre das durchaus machbar, da das VIO-Board auf rekonfigurierbarer Logik beruht. Ein Beschreiben der FPGA-Konfiguration über das LART böte die Möglichkeit, das Videointerface dynamischer zu verwenden.

Die verwendete Kamera hat zwar den Vorteil, dass sie preisgünstig ist, leider aber auch den Nachteil des nicht geringen Rauschens. Eine hohe Störanfälligkeit und nicht zuletzt einen unbeeinflussbaren Farb- und Helligkeitsabgleich sorgen für zusätzliche Schwierigkeiten in der nachfolgenden Verarbeitung. Insbesondere der Farbabgleich verlangt eine große Farbtoleranz beim Erkennen von farbigen Objekten. Ein Kompromiss ist schwer zu finden, da zwischen den Varianten 'genug richtiges Objekt erkennen' und 'wenig falsches Objekte sehen' der Spielraum sehr klein ist.

Die serielle Bluetooth-Verbindung ist zwar hilfreich, aber instabil. Ein Grund für diese Instabilität war nicht auszumachen. Indizien weisen auf den spezifischen Adapter, welcher allerdings noch ein Prototyp war und somit nicht mit Endprodukten verglichen werden kann.

5.3 Ausblick

Auf der technischen Seite gibt es da einige Punkte für Weiterentwicklungen:

- Ein Filter für einen automatischen Anti-Weißabgleich der Kamera damit die Störsicherheit verbessert wird und somit der Einsatz einer kostengünstigen Kamera möglich bleibt.
- Eine Verbesserung der Ballerkennung. Um zu erkennen, ob und wie der Ball verdeckt ist, kann das in dieser Arbeit entwickelte Verfahren verwendet werden, um den Ball zu finden. Ein anderes Verfahren würde dann die Koordinaten benutzen, um eine Formerkennung durchzuführen. Diese Möglichkeit befindet sich derzeit in der Entwicklung von Oliver Köckritz (Köckritz 2005).
- Interessant wäre es weiterhin, die 4. Dimension mit aufzunehmen. Ausgehend von der Annahme, dass sich das Bild nur um eine bestimmten Faktor bewegt, wäre man so in der Lage, Bewegungsvektoren abzuschätzen und die Suche zu verkürzen.
- Um abschätzen zu können, wie groß das erkannte Objekt ist, wäre es schön, die Größe der erkannten Fläche zurück zu liefern.

- Ob die Medianberechnung auch in der Praxis auf einem realen Roboter korrekt arbeitet, muss sich erst zeigen. Es könnte sich als besser erweisen, immer nur das Zentrum der größten zusammenhängenden Fläche zurück zu liefern. Dadurch würde das Risiko, den Ball zu verlieren, bei einem durch den Gegner teilverdeckten Tor minimiert werden.

Da die Entwicklung stetig voranschreitet, ist absehbar, dass in einigen Jahren die Embedded Systems mit gleichen technischen Randbedingungen eine Performance im Bereich der heutigen stationären Systeme bieten. Das heißt, dass sich das Handling, die Präzision und Geschwindigkeit des Sehens im Computerbereich erheblich verbessern werden.

Heutzutage gibt es bereits einen Blindenstock mit einem eingebautem Ultraschallsensor zur Entfernungsbestimmung. Denkbar wäre ein elektronischen Blindenhund im Rucksack, der später möglicherweise in eine Brille integriert wird.

In dieser Arbeit wurde nur ein kleiner Schritt auf dem langen Weg des Sehens gegangen, den gegebenen Bedingungen angemessen. Die Weiterentwicklung der Technik wird es uns aber ermöglichen, weitere, möglicherweise auch größere Schritte zu vollziehen.

Aber auch mit größeren Fortschritten werden wir sobald keine zweite Rasse (die der Androiden) neben uns stehen haben. Vielmehr werden wir uns mit immer mehr kleinen Helfern für dedizierte Aufgaben umgeben, die aber immer unsichtbarer im Hintergrund werkeln. Ständig schrecklich gut gelaunte Geräte, die sich darum drängen, uns behilflich zu sein zu können, werden es hoffentlich nicht sein.

Mein persönlicher Wunsch eines Widgets wäre eine Suchhilfe für Gegenstände, also ein Gerät das alles, was wir sehen, katalogisiert und abspeichert. Damit wäre es nämlich auch in Zukunft nicht nötig, meinen Dachboden aufzuräumen, da mir dieser kleiner Helfer auf Nachfrage stets sagen kann, in welchem Haufen ich denn nun diesmal das Buch mit der Antwort auf alle Fragen finden kann. . .

6 Anhang

6.1 CAN-Bus

Pinbelegung des RCube-CAN-Steckers

Die Pinbelegung (Kabelitz) des CAN-Pfostensteckers des RCubes:

PIN	Signal	Beschreibung
1	reserviert	
2	CAN-L	negiertes CAN-Signal (Dominant Low)
3	CAN-GND	Masse
4	reserviert	
5	CAN-SHLD	Schirmung (optional)
6	GND	Gerätemasse (optional)
7	CAN-H	positives CAN-Signal (Dominant High)
8	reserviert	
9	VCC	Versorgungsspannung (optional)
10	nc	nicht verbunden

Geschwindigkeit und Leitungslänge

Die Tabelle zeigt die Abhängigkeit der Bitrate von der Buslänge (Kabelitz):

Bitrate	Kabellänge
10 kbits/s	6,7 km
20 kbits/s	3,3 km
50 kbits/s	1,3 km
125 kbits/s	530 m
250 kbits/s	270 m
500 kbits/s	130 m
1 Mbits/s	40 m

6.2 Algorithmus HSV to RGB

```
   /**
     * Converts the components of a color, as specified by the HSB
     * model, to an equivalent set of values for the default RGB model.
     * <p>
     * The <code>saturation</code> and <code>brightness</code> components
     * should be floating-point values between zero and one
     * (numbers in the range 0.0-1.0).  The <code>hue</code> component
     * can be any floating-point number.  The floor of this number is
     * subtracted from it to create a fraction between 0 and 1.  This
     * fractional number is then multiplied by 360 to produce the hue
     * angle in the HSB color model.
     * <p>
     * The integer that is returned by <code>HSBtoRGB</code> encodes the
     * value of a color in bits 0-23 of an integer value that is the same
     * format used by the method {@link #getRGB() <code>getRGB</code>}.
     * This integer can be supplied as an argument to the
     * <code>Color</code> constructor that takes a single integer argument.
     * @param     hue   the hue component of the color
     * @param     saturation   the saturation of the color
     * @param     brightness   the brightness of the color
     * @return    the RGB value of the color with the indicated hue,
     *                              saturation, and brightness.
     * @see       java.awt.Color#getRGB()
     * @see       java.awt.Color#Color(int)
     * @see       java.awt.image.ColorModel#getRGBdefault()
     * @since     JDK1.0
     */
  public static int HSBtoRGB(float hue, float saturation, float brightness) {
int r = 0, g = 0, b = 0;
     if (saturation == 0) {
    r = g = b = (int) (brightness * 255.0f + 0.5f);
} else {
    float h = (hue - (float)Math.floor(hue)) * 6.0f;
    float f = h - (float)java.lang.Math.floor(h);
    float p = brightness * (1.0f - saturation);
    float q = brightness * (1.0f - saturation * f);
    float t = brightness * (1.0f - (saturation * (1.0f - f)));
    switch ((int) h) {
    case 0:
r = (int) (brightness * 255.0f + 0.5f);
g = (int) (t * 255.0f + 0.5f);
b = (int) (p * 255.0f + 0.5f);
break;
    case 1:
r = (int) (q * 255.0f + 0.5f);
```

```
g = (int) (brightness * 255.0f + 0.5f);
b = (int) (p * 255.0f + 0.5f);
break;
    case 2:
r = (int) (p * 255.0f + 0.5f);
g = (int) (brightness * 255.0f + 0.5f);
b = (int) (t * 255.0f + 0.5f);
break;
    case 3:
r = (int) (p * 255.0f + 0.5f);
g = (int) (q * 255.0f + 0.5f);
b = (int) (brightness * 255.0f + 0.5f);
break;
    case 4:
r = (int) (t * 255.0f + 0.5f);
g = (int) (p * 255.0f + 0.5f);
b = (int) (brightness * 255.0f + 0.5f);
break;
    case 5:
r = (int) (brightness * 255.0f + 0.5f);
g = (int) (p * 255.0f + 0.5f);
b = (int) (q * 255.0f + 0.5f);
break;
    }
}
return 0xff000000 | (r << 16) | (g << 8) | (b << 0);
    }
```

6.3 Algorithmus RGB to HSV

```
    /**
     * Converts the components of a color, as specified by the default RGB
     * model, to an equivalent set of values for hue, saturation, and
     * brightness that are the three components of the HSB model.
     * <p>
     * If the <code>hsbvals</code> argument is <code>null</code>, then a
     * new array is allocated to return the result. Otherwise, the method
     * returns the array <code>hsbvals</code>, with the values put into
     * that array.
     * @param     r   the red component of the color
     * @param     g   the green component of the color
     * @param     b   the blue component of the color
     * @param     hsbvals  the array used to return the
     *                     three HSB values, or <code>null</code>
     * @return    an array of three elements containing the hue, saturation,
     *                     and brightness (in that order), of the color with
     *                     the indicated red, green, and blue components.
     * @see       java.awt.Color#getRGB()
     * @see       java.awt.Color#Color(int)
     * @see       java.awt.image.ColorModel#getRGBdefault()
     * @since     JDK1.0
     */
    public static float[] RGBtoHSB(int r, int g, int b, float[] hsbvals) {
float hue, saturation, brightness;
if (hsbvals == null) {
    hsbvals = new float[3];
}
     int cmax = (r > g) ? r : g;
if (b > cmax) cmax = b;
int cmin = (r < g) ? r : g;
if (b < cmin) cmin = b;

brightness = ((float) cmax) / 255.0f;
if (cmax != 0)
    saturation = ((float) (cmax - cmin)) / ((float) cmax);
else
    saturation = 0;
if (saturation == 0)
    hue = 0;
else {
    float redc = ((float) (cmax - r)) / ((float) (cmax - cmin));
    float greenc = ((float) (cmax - g)) / ((float) (cmax - cmin));
    float bluec = ((float) (cmax - b)) / ((float) (cmax - cmin));
    if (r == cmax)
hue = bluec - greenc;
```

```
    else if (g == cmax)
        hue = 2.0f + redc - bluec;
            else
hue = 4.0f + greenc - redc;
    hue = hue / 6.0f;
    if (hue < 0)
hue = hue + 1.0f;
}
hsbvals[0] = hue;
hsbvals[1] = saturation;
hsbvals[2] = brightness;
return hsbvals;
    }
```

6.4 Linux-Howto

Anbindung des Bluetooth2Serial-Adapters unter SUSE-Linux 9.0:

1. Benötigte Bibliotheken:

   ```
   bluez-libs-2.4-31.i586.rpm
   bluez-pan-1.1-32.i586.rpm
   bluez-sdp-121-31.i586.rpm
   bluez-utils-2.3-57.i586.rpm
   bluez-utils-3.3-57.i586.rpm
   ```

   in ein Verzeichnis speichern und sie dort mit

   ```
   #>rpm -i *.rpm
   ```

   installieren.

2. Falls benötigt, den automatischen Start des Bluetooth-Dienstes erlauben:

   ```
   #>chkconfig bluetooth on
   ```

3. Folgende Zeilen müssen noch in die Dateien

   ```
   /etc/modutils/aliases
   ```

   und

   ```
   /etc/modules.conf
   ```

   einfügt werden:

   ```
   alias net-pf-31 bluez
   alias bt-proto-0 12cap
   alias bt-proto-2 sco
   alias bt-proto-4 bnep
   alias tty-ldisc-15 hci_uart
   alias char-major-10-250 hci_vhci
   ```

4. Im `/dev`-Verzeichnis müssen jetzt noch die notwendigen virtuellen Dateien erzeugt werden:

   ```
   #>mknod -m 666 /dev/rfcomm0 c 216 0
   #>mknod -m 666 /dev/rfcomm1 c 216 1
   ...
   ```

5. In der Datei

   ```
   /etc/bluetooth/pin
   ```

   ist ein Passwort gespeichert, welches wir der Einfachheit halber löschen. Dafür löschen wir den Inhalt der Datei bis auf einen Zeilenumbruch. Dieser ist wichtig: lässt man ihn weg, scheint eine Verbindung unmöglich.

6. Zusätzlich sollte man in der Datei

   ```
   /etc/bluetooth/hcid.conf
   ```

   nach dem Eintrag

   ```
   security auto;
   ```

   suchen und ihn durch

   ```
   #security auto;
   security none;
   ```

   ersetzen.

7. Nun können folgende daemons gestartet werden (möglicherweise ist das bereits beim Installieren geschehen):

   ```
   #>hcid
   #>sdp
   ```

8. Jetzt können wir mit Hilfe der *hcitools* nach in Reichweite befindlichen Geräten suchen:

   ```
   hcitool scan
   Scanning ...
        00:03:7A:0A:97:8F       Stollmann BlueRS+C1
   ```

   Die durch Doppelpunkte getrennten hexadezimalen Zahlen sind die BDA (**B**luetooth **D**evice **A**ddress)

9. Über von *rfcomm* werden wir jetzt eine Verbindung herstellen. Mit

   ```
   #>rfcomm bind 0 00:03:7A:0A:97:8F
   ```

   binden wir das Bluetooth-Gerät mit der BDA = 00:03:7A:0A:97:8F an `/dev/rfcomm0` So denn alles geklappt hat, sollte es jetzt zumindest das Device `/dev/ttyUB0` geben.

   Mit einem Terminalprogramm sollte man jetzt direkt über die Schnittstelle kommunizieren können.

7 Literaturverzeichnis

ActivMedia ROBOTS, ActivMedia Robots M.: *Robots & Robotics Accessories*. – URL http://www.activrobots.com/. – Zugriffsdatum: 21.07.2005

Apache FOUNDATION, The Apache S.: *Apache Ant - Welcome*. – URL http://ant.apache.org/. – Zugriffsdatum: 17.07.2005

Bakker u. a. BAKKER, Jan-Derk ; MOUW, Erik ; JOOSEN, Marc ; POUWELSE, Johan: *RCUBE - Die Hardware - Labor für Künstliche Intelligenz*. – URL http://www.lart.tudelft.nl/. – Zugriffsdatum: 16.07.2005

Boersch a BOERSCH: *AKSEN - Controller für reaktive Roboter - Labor für Künstliche Intelligenz*. – URL http://ots.fh-brandenburg.de/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=20&MMN_position=29:14:17. – Zugriffsdatum: 16.07.2005

Boersch b BOERSCH: *RCUBE - der Roboterkernel - Labor für Künstliche Intelligenz*. – URL http://ots.fh-brandenburg.de/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=8&MMN_position=17:14. – Zugriffsdatum: 16.07.2005

Boersch c BOERSCH: *RCUBE - Die Hardware - Labor für Künstliche Intelligenz*. – URL http://ots.fh-brandenburg.de/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=11&MMN_position=21:14:17#bv. – Zugriffsdatum: 16.07.2005

Bosch BOSCH: *Robert Bosch GmbH - Halbleiter und Sensoren für die Automobil-Zulieferindustrie:*. – URL http://www.semiconductors.bosch.de/de/20/can/index.asp. – Zugriffsdatum: 16.07.2005

BSIG BSIG: *Bluetooth Special Interest Group | Standards & Protocols*. – URL http://www.caba.org/standard/bsig.html. – Zugriffsdatum: 22.07.2005

CiA AUTOMATION (CIA), CAN in: *Controller Area Network (CAN) - CAN in Automation (CiA)*. – URL http://www.can-cia.org/. – Zugriffsdatum: 16.07.2005

CV HOMEPAGE, Computer V.: *Computer Vision Source Code*. – URL http://www-2.cs.cmu.edu/~cil/v-source.html. – Zugriffsdatum: 17.07.2005

Digital HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.: *HP United States - Welcome to Hewlett-Packard - Find computers, laptops, digital cameras, printers, enterprise solutions, and more.:.* – URL http://www.digital.com. – Zugriffsdatum: 22.07.2005

Eclipse COMMUNITY, Eclipse: *Eclipse.org Main Page*. – URL http://www.eclipse.org/. – Zugriffsdatum: 16.07.2005

Foley u. a. 1994a FOLEY, James D. ; DAM, Andries van ; FEINER, Steven K. ; HUGHES, John F. ; PHILLIPS, Richard L.: *Grundlagen der Computergrafik*. Kap. 14.1, Addison-Wesley, 1994. – ISBN 3-89319-647-1

Foley u. a. 1994b FOLEY, James D. ; DAM, Andries van ; FEINER, Steven K. ; HUGHES, John F. ; PHILLIPS, Richard L.: *Grundlagen der Computergrafik*. Kap. 14.7, Addison-Wesley, 1994. – ISBN 3-89319-647-1

Gimp TEAM, The G.: *GIMP - The GNU Image Manipulation Program*. – URL http://www.gimp.org/. – Zugriffsdatum: 17.07.2005

Görz 2003 GÖRZ, Günther ; ROLLINGER, Claus-Rainer ; SCHNEEBERGER, Josef: Handbuch der Künstlichen Intelligenz. Oldenbourg, 2003, Kap. Bernd Neumann: Bildverstehen - ein Überblick, S. 815 – 841. – ISBN 3486272128

HAW-Hamburg HAW-HAMBURG: *Homepage des Softwarelabors am Fachbereiches Elektrotechnik/Informatik der HAW-Hamburg*. – URL http://swt.informatik.haw-hamburg.de/. – Zugriffsdatum: 16.07.2005

iRobot IROBOT: *iRobot - Robots for the Real World : Home Page*. – URL http://www.irobot.com/home.cfm. – Zugriffsdatum: 17.07.2005

ISO ISO: *ISO 11898-4:2004*. – URL http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=36306&ICS1=43. – Zugriffsdatum: 16.07.2005

Jagdmann und Stickel JAGDMANN, Dirk ; STICKEL, Michael: *SUSE Linux auf USB Festplatte*. – URL http://llg.cubic.org/docs/bootusb/suse.html. – Zugriffsdatum: 16.07.2005

jEdit COMMUNITY jEdit: *jEdit - Programmer's Text Editor*. – URL http://www.jedit.org/. – Zugriffsdatum: 16.07.2005

Jewett JEWETT, Tom: *Color tutorial*. – URL http://www.cecs.csulb.edu/~jewett/colors/. – Zugriffsdatum: 22.07.2005

JSmooth TEAM, JSmooth: *JSmooth*. – URL http://jsmooth.sourceforge.net/. – Zugriffsdatum: 17.07.2005

Kabelitz KABELITZ, H.: *CAN Bus Grundlagen*. – URL http://www.me-systeme.de/canbus.html. – Zugriffsdatum: 22.07.2005

Köckritz 2005 KÖCKRITZ, Oliver: *Visuomotorische Bewegungskoordination für mobile Roboter*. 2005

Knorr KNORR, Gerd: *video4linux*. – URL http://linux.bytesex.org/v4l2/. – Zugriffsdatum: 17.07.2005

Köthe KÖTHE, Ullrich: *VIGRA - Vision with Generic Algorithms*. – URL http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/. – Zugriffsdatum: 17.07.2005

Manger 2003 MANGER, Michel: *Design und Realisierung von 'kostengünstigen' fußballspielenden Robotern*. 2003. – URL http://users.informatik.haw-hamburg.de/~kvl/manger/studienarbeit.pdf. – Zugriffsdatum: 16.07.2005

Manger 2004 MANGER, Michel: *Design und Realisierung einer experimentellen Plattform für Roboterfußball*. 2004. – URL http://users.informatik.haw-hamburg.de/~kvl/manger/diplom.pdf. – Zugriffsdatum: 16.07.2005

Microsoft CORPORATION, Microsoft: *Microsoft Windows Family Home Page*. – URL http://www.microsoft.com/windows/default.mspx. – Zugriffsdatum: 17.07.2005

Newton INC., Newton Research L.: *Cognachrome Vision System*. – URL http://www.newtonlabs.com/cognachrome/. – Zugriffsdatum: 21.07.2005

noDNA GMBH noDNA: *noDNA - realtime interactive solutions: shop*. – URL http://www.nodna.com/shop.365.0.html?&L=1%252Fmotekmotioncapturesolutionsandsystemsformedical%252Csimulation.html. – Zugriffsdatum: 17.07.2005

Novell NOVELL, Inc: *NOVELL: SUSE LINUX*. – URL http://www.novell.com/de-de/linux/suse/. – Zugriffsdatum: 16.07.2005

Otto OTTO, George: *Color in Computer Graphics*. – URL http://viz.aset.psu.edu/gho/sem_notes/color_2d/html/primary_systems.html. – Zugriffsdatum: 22.07.2005

Philips a N.V., Koninklijke Philips E.: *Home - Royal Philips Electronics*. – URL http://www.philips.de/. – Zugriffsdatum: 16.07.2005

Philips b N.V., Koninklijke Philips E.: *PCA82C250; CAN controller interface*. – URL http://www.semiconductors.philips.com/pip/PCA82C250_N4.html. – Zugriffsdatum: 16.07.2005

Philips c N.V., Koninklijke Philips E.: *SJA1000; Stand-alone CAN controller*. – URL http://www.semiconductors.philips.com/pip/SJA1000_N1.html. – Zugriffsdatum: 16.07.2005

RWTH Aachen TECHNICAL COMPUTER SCIENCE, RWTH Aachen U. Chair of: *LTI-Lib: LTI-Lib*. – URL http://ltilib.sourceforge.net/doc/html/modules.html. – Zugriffsdatum: 17.07.2005

Sedgewick 2002 SEDGEWICK, Robert: *Algorithmen*. Addison-Wesley, 2002. – ISBN 3827370329

Sharp CORP., Sharp E.: *GP2D12 - Sharp Microelectronics Products*. – URL http://www.sharpsma.com/part.php?PartID=3900. – Zugriffsdatum: 16.07.2005

sinclair 1980 SINCLAIR: *ZX81 Basic Programming*. Kap. 4, S. 25f, sinclair, 1980

Sony 2005 CORPORATION, Sony: *Sony Global - AIBO Global Link*. 2005. – URL http://www.sony.net/Products/aibo/index.html. – Zugriffsdatum: 27.07.2005

Stollmann VERTRIEBS-GMBH, Stollmann E. und: *Stollmann ISDN and Bluetooth products and protocol stacks*. – URL http://www.stollmann.de/. – Zugriffsdatum: 17.07.2005

Storjohann 2005 STORJOHANN, Timo: *Einsatz genetischer Lernverfahren für die Programmierung eines mobilen Roboters*. 2005

Sun a SUN MICROSYSTEMS, Inc.: *Java 2 Platform, Standard Edition (J2SE)*. – URL http://java.sun.com/j2se/index.jsp. – Zugriffsdatum: 16.07.2005

Sun b SUN MICROSYSTEMS, Inc.: *Java Media Framework API (JMF)*. – URL http://java.sun.com/products/java-media/jmf/. – Zugriffsdatum: 16.07.2005

Sun Microsystems SUN MICROSYSTEMS, Inc.: *Java Technology*. – URL http://java.sun.com/. – Zugriffsdatum: 17.07.2005

Xepox INC., Xepox T.: *Siemens SAB80C515A-M18-T3 Series Datasheets. SAB80C515A-N18-T3, SAB83C515A-5N18-T3, SAB83C515A-5N18, SAB80C515A-N18, SAB80C515A-M18-T3, SAB83C515A-5M18-T3 Datasheet*. – URL http://www.chipdocs.com/datasheets/datasheet-pdf/Siemens/SAB80C515A-M18-T3.html. – Zugriffsdatum: 16.07.2005

Ziener 2005 ZIENER, Michael: *Konstruktion eines programmierbaren, omnidirektionalen Roboters*. 2005

Versicherung über Selbstständigkeit

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

Hamburg, 27. Juli 2005	
Ort, Datum	Unterschrift