Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Florian Burka

Kameragestützte Objektverfolgung in Echtzeit
im Kontext mobiler Roboter

Florian Burka

Kameragestützte Objektverfolgung in Echtzeit im Kontext mobiler Roboter

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Gunter Klemke
Zweitgutachter : Prof. Dr. rer. nat. Kai von Luck

Abgegeben am 21. März 2007

Florian Burka

Thema der Bachelorarbeit
Kameragestützte Objektverfolgung in Echtzeit im Kontext mobiler Roboter

Stichworte
Mobile Roboter, Autonome Roboter, Echtzeit-Bildverarbeitung, RoboCup, Farbsegmentierung, IEEE 802.15.4

Kurzzusammenfassung
In dieser Arbeit wurde ein System geschaffen, mit dem man farblich markierte Objekte in Echtzeit - mit einer Kamera - verfolgen kann.
Um dies zu erreichen analysiert ein Kameraserver auf einem leistungsstarken Computer die Bilder einer Kamera, sucht die richtigen Farbobjekte, bestimmt Position und Winkel dieser Objekte und sendet diese Informationen an autonome Roboter. Hierzu werden verschiedene Methoden gezeigt, Farben Farbklassen zuzuordnen und Objekte zu erkennen. Des Weiteren wird eine Architektur eines Systems zur Objekterkennung sowie dessen Realisierung erläutert.

Florian Burka

Title of the paper
Realtime object tracking of mobile robots

Keywords
autonomous robots, mobile robots, real-time image processing, RoboCup, color-segmentation, IEEE 802.15.4

Abstract
This work deals with a real-time object tracker. The object tracker was used to track mobile robots playing soccer to inform them about their heading and position.
Therefor a camera server was deployed on a high-performance desktop class computer. The system has a camera that observes the playground. The images captured by the camera are then segmented into color classes which are used to identify the robots and the ball. The algorithms to segment an image into color classes and to locate objects are described. An architecture for an object tracker is shown as well as its implementation.

Danksagung

Hiermit möchte ich mich bei meinem Betreuer Prof. Dr. Gunter Klemke für die viele Zeit und die wertvollen Inspirationen bedanken. Von dieser guten Zusammenarbeit hat die Bachelorarbeit sehr profitiert.

Danken möchte ich auch meinen Lektoren, welche mir dort weiterhalfen, wo mir alles zu selbstverständlich war.

Und danken möchte ich auch meiner Freundin und meiner Familie für die viele Geduld und Unterstützung.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Roboter sind in ihren Ursprüngen als Helfer bei der Arbeit entworfen worden. So gibt es im tschechischen das Wort Robot für den "Frontdienst" und im Russischen das Wort "Robota", welches für Arbeit steht. Früher waren Roboter meist einfache Fließbandarbeiter, heutzutage jedoch nehmen uns Roboter nicht nur die Arbeit ab, sondern sind auch in anderen Bereichen aktiv.

Kinder lernen viel durch das Spielen, warum nicht auch Roboter?

Das RoboCup Projekt[1] formuliert folgendes Ziel:
By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.
"Roboter-Fußball ist eine Herausforderung für die Robotik und die künstliche Intelligenz gleichermaßen."(Robocup)
Beim Roboter-Fußball wird das Spielfeld als Versuchslabor benutzt. So wird die Arbeit zum Spiel und die Arbeitsroboter zu Spielrobotern.
Spielende Roboter helfen nicht nur der Wissenschaft, sie schaffen es auch Faszination für Technik zu wecken und so neue motivierte Forscher zu rekrutieren.

Menschen haben Sinne - Roboter Sensoren.
Die Menschliche Wahrnehmung profitiert von den unglaublichen Fähigkeiten unseres Gehirns.
Mit diesen Fähigkeiten muss nun die Roboterforschung konkurrieren.
Einer der kompliziertesten Sinne des Menschen ist die visuelle Wahrnehmung. Diese Art der Wahrnehmung auch den Maschinen zu ermöglichen ist ein weites Forschungsfeld.

An der Hochschule für Angewandte Wissenschaften Hamburg gibt es seit Jahren viele Kurse und Projekte, in denen Roboter verwendet werden.
Und so wurde dort der CT-Bot, ein Roboter, der von dem "Heise Zeitschriften Verlag" mit dem Ziel entwickelt wurde, die Forschung auch ohne große Labore zu ermöglichen, als Grundlage für ein neues Roboter-Projekt verwendet.

Nun gilt es diese Grundlage auszubauen.

[1] http://www.robocup.org/

1.1 Motivation

Die CT-Bots sind in ihren Möglichkeiten auf wenige Sensoren beschränkt. Mit ihnen Fußball zu spielen, ist möglich, aber auch nicht sehr vielseitig, da man in seinen Möglichkeiten beschränkt ist. An der Hochschule für Angewandte Wissenschaften Hamburg wurde schon viel mit Robotern Fußball gespielt, die ein klein wenig mehr können als die CT-Bots.

Es lag also nahe, die CT-Bots mit weiteren Sensoren auszurüsten, um das Spiel komplexer und die möglichen Aufgabenstellungen interessanter zu gestalten. Eine Gruppe Studenten beschäftigte sich deswegen damit, den CT-Bot mit einer Kamera auszustatten, sodass dieser seine Umgebung genauer beurteilen kann. Eine weitere Möglichkeit, den CT-Bot mit Informationen zu versorgen, ist, eine Kamera in die Lage zu versetzen, das gesamte Spielfeld zu überblicken. Dadurch kann die Position jedes CT-Bots und des Balls genau bestimmt werden.

Ein solcher *Object Tracker* wird derzeit auch bei Roboter-Fussballturnieren verwendet. Hierfür hat jede Mannschaft ihre eigene Software, die meist eng mit der Strategieplanung und Koordination der Roboter verzahnt und in den meisten Fällen nicht frei verfügbar ist.

An der Hochschule für Angewandte Wissenschaften Hamburg wurden schon mehrere Versuche gestartet, das Problem zu beheben, aber leider führten die Ansätze nicht zu befriedigenden Lösungen.

Daher musste eine Lösung her, die auch während eines Projektes verwendet werden kann.

1.2 Zielsetzung

Die CT-Bots brauchen mehr Informationen um, besser in ihrer Umgebung agieren zu können.

Daher sollen sie diese Informationen von einer externen Instanz bekommen welche die Welt, in der die Roboter sich bewegen, komplett überblicken kann.

Diese Instanz zu realisieren ist nun das Ziel dieser Arbeit.

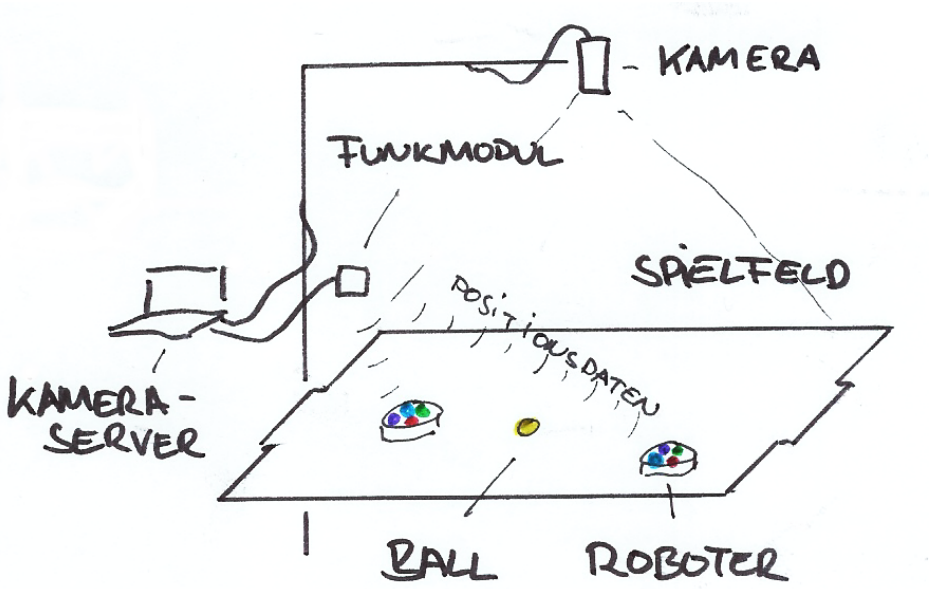

Abbildung 1.1: Der Gesamtüberblick über das System

Um dieses zu ermöglichen wird eine Kamera so aufgehängt, dass diese das gesamte Spielfeld überblicken kann; die Roboter und der Ball erhalten Farbmarkierungen. Ein Kameraserver auf einem leistungsstarken Computer soll die Bilder der Kamera analysieren, die richtigen Farbobjekte suchen, Position und Winkel dieser Objekte bestimmen und diese Informationen an die autonomen Roboter senden.

1.3 Gliederung

In dem Kapitel "Analyse"(2) werden zunächst die Anforderungen an die zu entwickelnde Lösung gesammelt. Anschließend werden verschiedene Farbmodelle (2.2) diskutiert und die zur Verfügung stehenden Technologien (2.3) an der Hochschule für Angewandte Wissenschaften Hamburg betrachtet. Das Kapitel stellt schließlich verschiedene Möglichkeiten

zur Lösung einander gegenüber (2.4) und bietet eine Übersicht an Arbeiten, die sich mit vergleichbaren Themen beschäftigen (2.5).

Der Entwurf der Lösung wird im Kapitel "Design"(3) durch die Beschreibung des Programmablaufs und ihre Struktur mit Hilfe von Klassendiagrammen dargestellt.

Das nachfolgende Kapitel "Realisierung"(4) widmet sich der Umsetzung der Anforderungen und des Designs anhand des Anwendungsbeispiels Roboter-Fußball. Es beinhaltet zunächst einen Überblick über die gewählte Programmierumgebung (4.1, 4.2), die verwendeten Bibliotheken (4.3), Vorgehensweisen (4.5) zur Realisierung von Nebenläufigkeit (4.4) und Methoden zur Auffindung und Vermeidung von Speicherlecks (4.6.2).

Abschließend stellt das Fazit (5) die erreichten Ergebnisse (5.1), aufgetretene Probleme und deren Lösungen (5.2) und einen Ausblick auf Erweiterungsmöglichkeiten (5.3) dar.

2 Analyse

In diesem Kapitel werden zunächst die genauen Anforderungen abgesteckt.

Nachfolgend werden die Grundlagen vermittelt, die für das Verständnis der Lösung notwendig sind.

Zu den wichtigen Grundlagen gehört ein Überblick über die gebräuchlichen "Farbmodelle"(2.2). Dies ist wichtig, da die Objekterkennung auf der Erkennung von Farben basiert und die gute und stabile Verarbeitung und Erkennung dieser Farben wichtig für die Gesamtfunktionalität des Systems ist.

Die Rahmenbedingungen an der Hochschule für Angewandte Wissenschaften Hamburg zeigen, in welchem Umfeld und mit welchen Mitteln gearbeitet wurde. Zur Beschreibung dieser Mittel wird auch kurz gezeigt, welche Roboter (2.3.3) an der Hochschule für Angewandte Wissenschaften Hamburg verwendet werden.

Um eine effiziente Farberkennung zu implementieren, bedarf es einer genauen Kenntnis der verschiedenen Farbmodelle (2.2). Zudem sollten die von der Kamera (2.3.2) gelieferten Bilder möglichst brillant[1] sein. Daher wird kurz gezeigt, welche gravierenden Unterschiede es bei vermeintlich ähnlichen Kameras gibt. Weiterhin werden das "Spielfeld"(2.3.1), der "Computer"(2.3.5) und die Funktechnologie (2.3.4) erklärt, welche zur Verwendung kamen.

Das Kapitel schließt mit einem Blick auf die internationale Forschung ab, der zeigt, was bisher in verwandten Arbeiten erreicht wurde.

2.1 Anforderungen

Ziel dieser Arbeit ist es, ein System für kameragestützte Objektverfolgung zu gestalten.

In diesem Fall bedeutet das konkret, dass das entwickelte System die Roboter und den Spielball mit Hilfe einer Kamera erkennt und verfolgt sowie die Information über den jeweiligen Aufenthaltsort weitergeben können muss.

[1] Brillante Bilder nutzen den Farbraum möglichst gut aus und beschränken sich nicht auf nur wenige mögliche Farbwerte.

Das zu entwickelnde System hat die Aufgabe, in einem Kurs der Hochschule für Angewandte Wissenschaften Hamburg als Kameraserver zu fungieren, um Robotern ihre absolute Position[2] mitteilen zu können.

Des Weiteren soll es folgenden Grundsätzen Genüge tun:

- Konfigurierbarkeit: Die Farbklassen sollen einfach einzustellen sein. Auch Parameter wie die verwendete Kamera, die verwendete Ausgabe für das Funkmodul und die Einstellungen für die Objekterkennung sollen gut konfigurierbar sein.
- Erweiterbarkeit: Das System soll in seinem Design so gestaltet sein, dass es mit wenig Aufwand um neue Funktionen erweitert werden kann (5.3) und dass auch komplette Komponenten einfach ausgetauscht werden können.
- Portierbarkeit: Das System soll möglichst unabhängig von dem verwendeten System programmiert werden. So sollte zum Beispiel das Betriebssystem durch ein anderes ersetzt werden können.
- Es soll die weiche Echtzeit[3] für die Weitergabe der Informationen an die Roboter eingehalten werden.

Das Programm soll folgende Anforderungen erfüllen:

- Farberkennung
  - Eine Farbfläche soll erkannt werden.
  - Die Position von einem mit einer Farbfläche markiertem Objekt soll erkannt werden.
  - Von einem durch mehreren Farbflächen markiertem Objekt soll die Position und die Ausrichtung erkannt werden.
- Anzeige und Weitergabe der Ergebnisse
  - Die Ergebnisse können an mehrere Empfänger gefunkt werden.
  - Die Ergebnisse können so ausgegeben werden, wie der Roboter sie auch empfängt (um festzustellen, ob ein Fehler beim Roboter oder bei der Erkennung liegt).

[2] Die CT-Bots (2.3.3) haben selber nur begrenzte Mittel zur Bestimmung ihrer Position und können höchstens relative Positionsveränderungen bestimmen.

[3] Echtzeit bedeutet, dass die Antwortzeiten des Systems unterhalb einer vorher festgelegten Grenze liegen (z. B. < 10 ms). Weiche Echtzeit bedeutet, dass ein Überschreiten dieser Zeit zwar nicht gewünscht ist, aber auch keine fatalen Folgen hat. Die Steuerung eines Airbags zum Beispiel hat harte Echtzeitanforderungen zu erfüllen.

  - Die Ergebnisse können zur Kontrolle auf einem Ergebnisbild angezeigt werden.
  - Man kann sich das Kamerabild zur Kontrolle anzeigen lassen.
  - Es soll eine Möglichkeit geben, das Ergebnis und die Funktionsfähigkeit der Software einfach auf dem Monitor zu überprüfen, da die Berechnungen auf den Robotern schwerer überprüfbar sind.
- Modifizierbarkeit der Ausgabe
  - Es sollen Filter in die Ausgabe eingehängt werden können, welche zum Beispiel die Position anhand der aktuellen Geschwindigkeit vorausberechnen.
- Konfigurierbarkeit und Benutzbarkeit
  - Es soll nachvollziehbar sein, wie das System die Farben den Farbklassen zuteilt. Durch eine Anzeige dieser Farbklassen soll auch die Erkennung der Roboter nachvollziehbarer sein.
  - Die einzelnen Farben können, während das Programm läuft, schnell umgestellt werden, damit Veränderungen in der Beleuchtung ausgeglichen werden können.

2.2 Farbmodelle

In der Welt der Informationstechnologie existieren verschiedene, etablierte Farbmodelle, die für unterschiedliche Bereiche und Anwendungsfälle entwickelt wurden.

Ein Farbmodell beschreibt eine Sicht, Farben zu beschreiben. Ein Farbraum ist der Raum, welcher durch ein Farbmodell beschrieben wird. Ein Farbformat beschreibt die Darstellungsform einer Farbe - mit Hilfe eines Farbmodells und innerhalb eines Farbraums - für einen Computer.

Die Wahrnehmung einer Farbe hängt jedoch von weiteren Faktoren ab. So scheint zum Beispiel ein mit nur rotem Licht beleuchtetes weisses Blatt Papier auch rot zu sein. Daher ist die Wahrnehmung von Farben auch stark von dem Licht abhängig, welches diese Beleuchtet.

Es gibt verschiedene Herangehensweisen, eine Farbe im Computer zu speichern. Zum einen kann eine Farbe aus mehreren Farben zusammengesetzt werden, welche additiv oder subtraktiv gemischt werden (2.2.1 und 2.2.2). Zum anderen kann die Farbe als Wert auf dem Farbkreis angegeben und durch weitere Angaben ergänzt werden (2.2.3). Schließlich kann die Farbe auch getrennt von ihrer Lichtintensität angegeben werden (2.2.4).

Es folgt ein Überblick über die in der Informatik gebräuchlichsten Farbmodelle, die hinsichtlich ihrer Brauchbarkeit für die automatischen Farberkennung beurteilt werden. Für die Wahl des Farbmodells ist es ebenfalls notwendig zu beurteilen, wie schwierig die Festlegung eines Farbtons oder eines Farbbereichs ist.

Eine Auswahl des verwendeten Farbmodells erfolgt erst in Kapitel 3.5.

Abbildung 2.1: Additive Farbmischung der Farben Rot, Grün und Blau

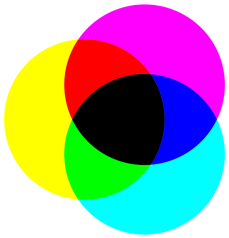

Abbildung 2.2: Subtraktive Farbmischung der Farben Türkis, Magenta und Gelb

Die additive Farbmischung entspricht der Farbmischung des Lichts von verschiedenen Scheinwerfern, hier mit den Farben Rot, Grün und Blau (Abbildung 2.1). Wenn alle Farben zu gleichen Teilen mit voller Stärke gemischt werden, ergibt sich im Idealfall Weiss.

Die subtraktive Farbmischung (Abbildung 2.2) hingegen ist mit der Verwendung von Farbpigmenten wie bei Druckern zu vergleichen. Bei der Verwendung der Farben Türkis, Magenta und Gelb in maximaler Intensität erhalten wir Schwarz.

2.2.1 RGB

Das RGB-Farbmodell verwendet die Grundfarben **R**ot, **G**rün und **B**lau, um eine Farbe zu beschreiben. Diese werden in additiver Farbmischung zusammengefügt, um die gewünschte Farbe zu erhalten.

Dieses Modell wird hauptsächlich für die Darstellung von Farben auf Computermonitoren und für die Farbbeschreibung im World-Wide-Web genutzt. Es ist auch ähnlich aufgebaut, wie das menschliche Auge, welches über Farbrezeptoren für Rot, Grün und Blau verfügt.

In diesem Farbmodell werden Grautöne, auch Schwarz und Weiss, dadurch ausgedrückt, dass jeweils gleiche Anteile von Rot, Grün und Blau angegeben werden. Weiss erhält man, indem man jeweils den maximalen Farbwert nimmt, Schwarz, indem man alle Farbwerte auf den minimalen Farbwert setzt. Bei den Farben verhält es sich ähnlich: Dunklere Farben

werden durch einen geringen Farbanteil, hellere Farben durch einen höheren Farbanteil der Grundfarben dargestellt.

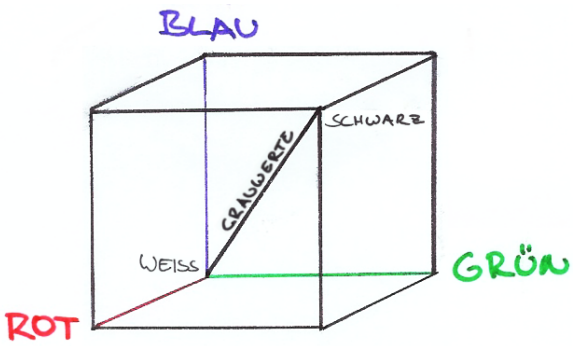

Abbildung 2.3: Der RGB-Farbraum mit den Grauwerten auf der Diagonalen zwischen Weiss und Schwarz

Durch diese Eigenschaft des RGB-Farbmodells liegen gleiche Farbtöne in etwa in einem Zylinder, dessen Hauptachse parallel zur Grauwert-Diagonalen verläuft.

Da dieses Modell das gängigste aller Farbmodelle ist, in vielen Bibliotheken verwendet wird (beispielsweise CImg (Tschumperle, 2007) und LTI-Lib (Alvarado u. a., 2007)) und am besten unterstützt wird, bietet es sich an, dieses ebenfalls zu benutzen.

Im RGB-Farbmodell einen Farbton zu beschreiben, ist nicht sehr leicht. Das liegt an dem komplexen geometrischen Gebilde, das konfiguriert werden müsste, um einen Farbton festzulegen.

2.2.2 CMYK

Im CMYK-Farbmodell wird eine Farbe durch ihre Anteile an Türkis (**C***yan*), **M**agenta[4] , Gelb (**Y***ellow*) sowie Schwarz (**K***ey*[5]) beschrieben, welche in subtraktiver Farbmischung zu der gewünschten Farbe vermischt werden.

CMYK wird bei Farbdruckern verwendet, um die Anteile an Pigmenten anzugeben, mit denen Papier bedruckt wird.

Türkis, Magenta und Gelb sind die Komplementärfarben zu Rot, Grün und Blau. Daher kann man das CMYK-Farbmodell auch als invertiertes RGB-Farbmodell betrachten.

[4] Anilinrot, ein ins Purpur übergehendes Rot mit einem leichten lila Farbton

[5] Auch **K**ontrast oder Blac**k**. Um Missverständnissen vorzubeugen, da es sonst auch als **B***lue* interpretiert werden könnte.

Programmcode 2.1: Die Umwandlung von CMYK nach RGB

```
b = (MAX_VALUE − k) ∗ (MAX_VALUE − c) / MAX_VALUE;
g = (MAX_VALUE − k) ∗ (MAX_VALUE − m) / MAX_VALUE;
r = (MAX_VALUE − k) ∗ (MAX_VALUE − y) / MAX_VALUE;
```

Der schwarze Anteil wird benutzt, um beim Drucken nach CMYK sowohl ein gesättigtes Schwarz zu ermöglichen als auch Farben abzudunkeln, da ein reines Übereinanderdrucken von Türkis, Magenta und Gelb kein absolutes Schwarz, sondern ein dunkles Ocker ergibt.

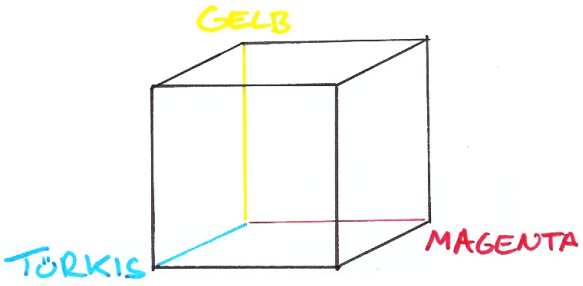

Abbildung 2.4: Der CMYK-Farbraum

Aufgrund der subtraktiven Farbmischung (Abbildung 2.2) ergibt sich Schwarz durch die Mischung aller Farben mit jeweils maximalen Farbwert, analoges gilt für Weiss.

Eine Umwandlung von CMYK in das RGB-System ist einfach und unkompliziert (siehe Programmcode 2.1).

Gleichartige Farbtöne sind genauso schwer zu beschreiben wie in dem RGB-Farbmodell.

2.2.3 HSV

Im HSV-Farbmodell wird eine Farbe durch ihren Farbton (**H***ue*), ihre Sättigung (**S***aturation*) und ihren Grauwert (**V***alue*) angegeben.

Die Farbmodelle HSL, HSB und HSI sind dem HSV-Farbmodell sehr ähnlich. In diesen Farbmodellen wird lediglich die Helligkeit unterschiedlich angegeben. So wird im HSL-Farbmodell die relative Helligkeit (**L***ightness*), im HSB-Farbmodell die absolute Helligkeit (**B***rightness*) und im HSI-Farbmodell die Lichtintensität (**I***ntensity*) angegeben.

Die verschiedenen Farben werden im HSV-Farbmodell auf einem Farbkreis angegeben (Abbildung 2.5).

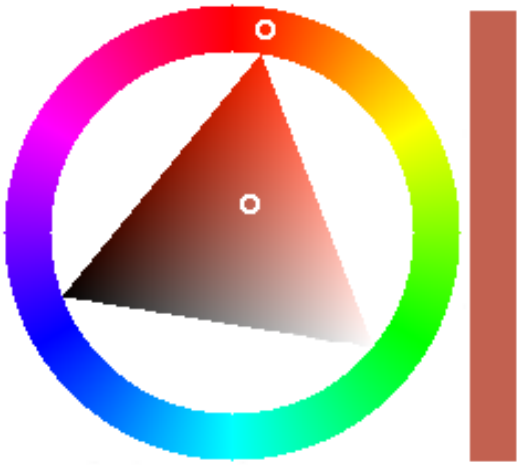

Abbildung 2.5: Der HSV Farbkreis mit den Werten H=9, S=59 und V=76 (aus (GIMP-Team, 2007))

Die Sättigung gibt an, wie stark der Anteil der Farbe und des Grauwertes an der resultierenden Farbe ist. Bei einer minimalen Sättigung und einem maximalen Grauwert ist die resultierende Farbe Schwarz, bei einem minimalen Grauwert Weiss. Wenn hingegen eine maximale Sättigung verwendet wird, hat der Grauwert keinerlei Bedeutung und es kommt nur auf den gewählten Farbwert an (Abbildung 2.6).

Es ist recht aufwändig, aus einem RGB-Farbwert den korrespondierenden HSV-Farbwert zu berechnen. Dies liegt vor allem daran, dass die Position der Farbe auf dem Farbkreis bestimmt werden muss (Programmcode 2.2).

Bei dem HSV-Farbmodell lassen sich Farbtöne und -bereiche gut beschreiben, indem man einfach einen Abschnitt auf dem Farbkreis markiert und für Sättigung und Helligkeit minimale und maximale Werte angegeben werden.

Die Nachteile des HSV-Farbmodells sind folgende:

- Es ist aufwändig, eine Farbe aus oder in das RGB-Farbmodell zu konvertieren (Programmcode 2.2).
- Nicht stark oder gar nicht gesättigte Farben - wie Grautöne - haben sowohl einen beliebigen als auch einen beliebig schwankenden Farbwert.
- Der Wertebereich der möglichen Farben ist auf einen Kreis abgebildet. Dadurch folgt auf 359° sowohl 360° als auch 0°. Wenn also eine Toleranz von $\pm$ 10 um 5° gefordert ist, sind alle Werte von 0° bis 15°, sowie von 355° bis 360°, innerhalb dieses Toleranzbereichs.

Programmcode 2.2: Die Umwandlung von RGB nach HSV

```
//red, green, blue, value und saturation sind zwischen 0 und MAX_VALUE \zB 255
//hue ist zwischen 0 und 359 Grad
void rgb2hsv( int red, int green, int blue, int * hue, int *saturation, int *
    value )
{
  int max_val, min_val;
  /*Grauwert (value) bestimmen*/
    max_val = MAX(red,green,blue);
    min_val = MIN(red,green,blue);
    *value = max_val ;
    /*Wenn es sich um einen reinen Grauwert handelt, sind wir fertig.*/
    if(max_val == min_val)
    {
      *saturation = 0;
      *hue = 0;
    }
    else /*Aber wenn nicht, muessen wir noch die Farbe und deren Saettigung
        bestimmen.*/
    {
      int delta = max_val-min_val;
      /*Zunaechst bestimmen wir die Saettigung der Farbe.*/
      *saturation = (0==max_val)?0: MAX_VALUE - delta;
      /*danach den Farbwert..*/
      if( red == max_val )
    {
            *hue = 60 * (green - blue) / delta;
            if( blue <= green )
            {
              *hue = *hue + 360;
            }
        }
        else if( green == max_val )
        {
          *hue = 60 * (blue - red) / delta + 120;
        }
        else /* if( blue == max_val ) */
        {
            *hue = 60 * (red - green) / delta + 240
      }
}
```

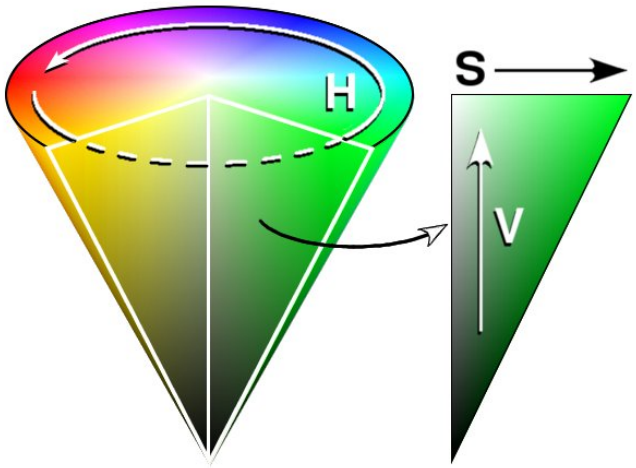

Abbildung 2.6: Der HSV-Farbraum als Kegel. Die Werte H, S und V stehen für den Farbton (**H***ue*), die Sättigung (**S***aturation*) und dem Grauwert (**V***alue*) (Pierce, 2005)

2.2.4 YUV

Das YUV-Farbmodell verwendet zur Beschreibung einer Farbe deren Lichtstärke und deren Grundfarbe. Die Grundarbe ist hierbei in zwei Komponenten aufgeteilt: U und V stehen jeweils für den Anteil an Blau und Rot. Y steht für die Helligkeit.

Dieses Farbmodell wird bei Farbfernsehern verwendet. Diese übertragen dabei die Helligkeit häufiger als die Farbinformationen, da das menschliche Auge empfindlicher auf Helligkeitsunterschiede als auf Farbunterschiede reagiert.

Das gerade angeführte Farbmodell ist den Farbmodellen YCrCb und YPbPr sehr ähnlich. Der einzige Unterschied besteht in der Skalierung der Farbachsen. Dadurch lässt sich beispielsweise YPbPr sehr einfach in YUV umwandeln (Programmcode 2.3)

Programmcode 2.3: Die Umwandlung von YPbPr nach YUV

```
u = 0.872921 * pb;
v = 1.229951 * pr;
```

Die Umwandlung in ein anderes Farbmodell ist recht einfach zu bewerkstelligen (Programmcode 2.4).

Programmcode 2.4: Die Umwandlung von RGB nach YUV (Jack, 1993)

```
Y =  (0.257 * R) + (0.504 * G) + (0.098 * B) + 16
```

$$Cr = V = \ (0.439 * R) - (0.368 * G) - (0.071 * B) + 128$$
$$Cb = U = -(0.148 * R) - (0.291 * G) + (0.439 * B) + 128$$

Es gibt verschiedene Formate, die das YUV-Farbmodell als Grundlage benutzen. Diese Formate unterscheiden sich in der Reihenfolge und Häufigkeit, mit der die einzelnen Komponenten benutzt werden.

Für unser System ist das Format UYVY oder auch YUV 4:2:2 interessant. In diesem Format wird der Y-Wert für jedes Pixel übertragen und die Werte für U und V immer abwechselnd nur für jedes zweite (Programmcode 2.5)[6].

Programmcode 2.5: Das UYVY-Format)

```
uyvy_pixel = {u,y0,v,y1};
```

Das YUV-Farbmodell ermöglicht es leicht, einen Farbton anzugeben. Dies liegt daran, dass die Farbe auf der Farbfläche (Die U und V-Komponenten des YUV-Farbmodells) festgelegt werden kann und diese nur noch durch die maximale sowie minimale Helligkeit abgegrenzt werden muss. So ist sowohl eine recht natürliche Art der Abgrenzung eines Farbtons gegeben als auch eine, die leicht beschrieben werden kann.

2.3 Rahmenbedingungen

Seit 1996 gibt es an der Hochschule für Angewandte Wissenschaften Hamburg das Projekt "Integration Kognitiver Systeme"[7]. In dem Kontext dieses Projektes werden Kurse zum Thema mobile Roboter oder Robot-Vision angeboten. Für dieses Projekt stehen auch ein eigenes Labor sowie eine Werkstatt zur Verfügung.

Im Zuge dieser Kurse wurden im Jahre 2006 mehrere CT-Bots (Heise Zeitschriften Verlag, 2006) (2.3.3) angeschafft, die auf einem Spielfeld (2.3.1) gegeneinander spielen können. Es standen mehrere Kameras zur Verfügung (2.3.2). Die Verbindung zu den Robotern wurde per Funk (2.3.4) aufgebaut.

[6] Eine ausführliche Beschreibung weiterer, auf dem YUV-Farbmodell basierender Formate ist unter http://fourcc.org/yuv.php zu finden.

[7] http://users.informatik.haw-hamburg.de/~kvl/

2.3.1 Spielfeld

Für die Roboter wurde ein Spielfeld (Abbildung 2.7) mit den Maßen 122 x 183 cm verwendet. Diese Größe ist die selbe, wie sie beim "RoboCup Junior Soccer"[8] für Spiele 2 gegen 2 vorgesehen ist

Das Spielfeld hat eine kleine Schräge vor der Wand, welche das Spielfeld begrenzt, damit der Ball leichter vom Rand zurückkommt, und einen etwa 4 cm vom Rand entfernten schwarzen Strich, damit die CT-Bots erkennen, wenn sie nahe am Rand sind.

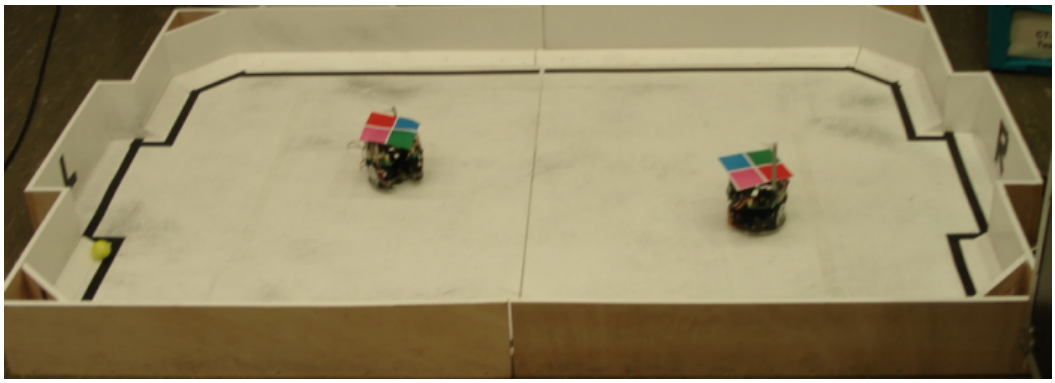

Abbildung 2.7: Das Spielfeld

2.3.2 Kamera

An der Hochschule für Angewandte Wissenschaften Hamburg stehen mehrere Arten von Kameras zur Verfügung: So gab es USB-Kameras, Webcams und Firewire-Kameras.

Im Verlauf der Entwicklung stellte sich heraus, dass die Firewire-Kameras deutlich brillantere Bilder liefern, sodass im weiteren Zuge der Entwicklung hauptsächlich Firewire-Kameras verwendet wurden.

Am wichtigsten für die Farberkennung war, dass die Kamera das vorgegebene Farbspektrum (Abbildung 2.8) möglichst gut ausnutzt und sich nicht nur auf wenige Werte in einem eingeschränkten Bereich verlässt, um einzelne Farben besser voneinander abgrenzen zu können. Hierzu wurde mit den Kameras eine Testsituation (Abbildung 2.9) gefilmt und anschließend das von der Kamera gelieferte Bild auf die Ausnutzung des Farbspektrum hin untersucht.

[8] Die "RoboCup Junior" Initiative hat das Ziel "Kindern und Jugendlichen Roboter und ihre Anwendung vorzustellen" (http://www.robocupjunior.de/).

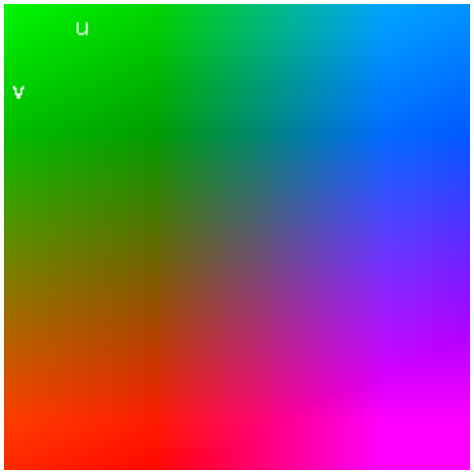

Abbildung 2.8: Das UV-Farbspektrum aus dem YUV-Farbmodell

Abbildung 2.9: Die Testsituation für die Kameras, welche alle Farben beinhaltet, die später auch erkannt werden sollen.

Die Sony DFV-V500 Kamera (Abbildung 2.10) liefert ein sehr breites Spektrum an Farben (In Abbildung 2.11 sind nur die von der Kamera gelieferten Farben eingezeichnet).

Die Imagingsource DFK 21BF04-Z (Abbildung 2.12) hingegen liefert ein deutlich weniger vielfältiges Farbspektrum (Abbildung 2.13) unter den gleichen Bedingungen.

Abbildung 2.10: Die Sony DFW-V500 Kamera

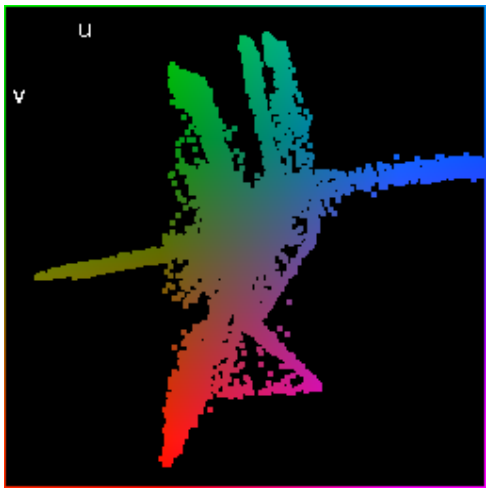

Abbildung 2.11: Das Farbspektrum der Sony DFW-V500 Kamera

Abbildung 2.12: Die Imagingsource DFK 21BF04-Z Kamera

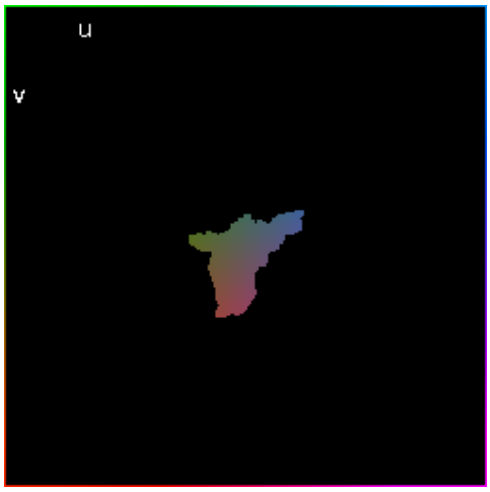

Abbildung 2.13: Das Farbspektrum der Imagingsource DFK 21BF04-Z Kamera

Die Sony-Kamera nutzt das Farbspektrum nahezu vollständig aus und liefert auch scharfe Bilder. Daher wurde diese Kamera verwendet.

2.3.3 Roboter

An der Hochschule für Angewandte Wissenschaften Hamburg werden Roboter in vielen Projekten verwendet. So gibt es unter anderem den Pioneer-Roboter (Abbildung 2.14), verschiedene Roboter auf Basis des Aksen-Boards [9] (Abbildung 2.15 und 2.16), Roboter auf reiner Lego-Basis (Abbildung 2.17) und die CT-Bots[10] (Abbildung 2.18)

Während der Entwicklung wurde mit den CT-Bots getestet, da diese von einer Gruppe Studierender zusammen mit dem erstellten Programm in einem Praktikum verwendet werden sollten.

2.3.4 Funk

Für die Funkverbindung zu den Robotern wurde der Funkstandard IEEE 802.15.4 genutzt. IEEE 802.15.4 ist ein Funkstandard, welcher mit Hinblick auf niedrigen Stromverbrauch bei niedrigen Datenraten konzipiert wurde.

Der IEEE 802.15.4 Funkstandard bietet viele Vorteile: So erlaubt er Netzwerke mit 2^{16} Einheiten, welche gleichberechtigt funken können, die Latenzzeiten sind im Bereich von Hundertstelsekunden (bei Bluetooth dauert zum Beispiel der Beitritt zu einem Netzwerk länger als 3 Sekunden, bei IEEE 802.15.4 etwa 30 Millisekunden) und der Energieverbrauch ist auch sehr gering.
Die Nachteile sind, dass der Funkstandard nur für Reichweiten von 75 Metern geschaffen wurde und dass die Verbindungsgeschwindigkeit bei maximal 250 KBit je Sekunde liegt.

ZigBee[11] ist ein Netzwerkstack[12] für IEEE 802.15.4, welcher im Hinblick auf mobile Systeme mit wenig Ressourcen entworfen wurde.

[9] Das Aksen-Board (http://www.aksen-roboter.de/) ist ein fertiger Controller für Roboter, an den Sensoren und Aktoren angeschlossen werden können.

[10] Der CT-Bot (http://www.heise.de/ct/ftp/projekte/ct-bot/ct-bot.shtml) ist ein Roboter vom Heise Zeitschriftenverlag, welcher einen preiswerten Einstieg in Roboterbau und -programmierung bieten soll.

[11] http://www.zigbee.org

[12] Der Netzwerkstack ist die Softwareschicht, welche die Vermittlung von Daten, hier über Funk, steuert. Der Netzwerkstack ist in etwa zu vergleichen mit der Post. Ein beliebiger Mensch kann einfach einen Brief bei der Post abgeben und die Post kümmert sich darum, dass dieser bei seinem Ziel ankommt.

Abbildung 2.14: Der Pioneer Roboter

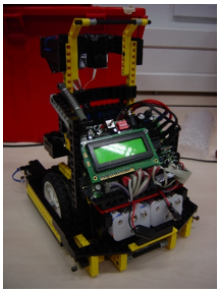

Abbildung 2.15: Ein Lego-Roboter, gesteuert durch das Aksen Board

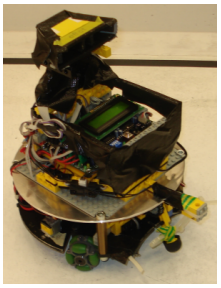

Abbildung 2.16: Ein Roboter mit omnidirektionalem Antrieb

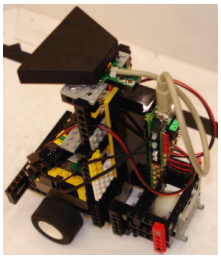

Abbildung 2.17: Ein Lego-Roboter

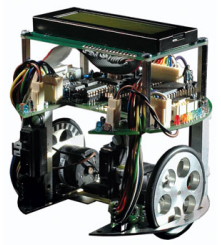

Abbildung 2.18: Ein CT-Bot

Mehr dazu bei (Fischer, 2006), der das ZigBee Protokoll speziell in Hinblick auf spontane Funknetzwerke betrachtet hat und bei (Rickens, 2005), welcher ein ZigBee Funkmodul über den CAN-Bus[13] an mobile Roboter angebunden hat.

Für dieses Projekt hat Prof. Dr. Gunter Klemke ein IEEE 802.15.4 Funkmodul sowohl für die CT-Bots als auch eines, das per USB an einen Computer angeschlossen wird, entworfen.

2.3.5 Computer

Für die Auswahl eines Computers waren zwei Faktoren entscheidend. Zum einen musste der Computer mit einem Firewire-Port ausgestattet sein, welcher auch die Stromversorgung der angeschlossenen Geräte übernimmt[14], und er musste über genug Rechenkraft verfügen, da die bisherigen Ansätze (Schmidt (2005) und Gottwald (2005)) gezeigt haben, dass die Rechenkraft bei der Bildverarbeitung schnell zu einem Engpass wird. Des Weiteren war es notwendig, kompletten Zugriff auf das System zu haben, um die Treiber für die Kamera auszutauschen.

Diese Anforderungen wurden an der Hochschule für Angewandte Wissenschaften Hamburg von einem MacBook erfüllt, welches im weiteren Verlauf unter Linux und Mac OS X zur Entwicklung und Benutzung der Software verwendet wurde. Es besaß sowohl einen Firewire-Port als auch einen Dualcore-Prozessor[15](ein "Intel Core 2 Duo" mit 2 Ghz), welcher es ermöglichte, in dem Programm auf jedem dieser Prozessorkerne einen Verarbeitungsstrang (*Thread*) laufen zu lassen (2.4.3).

2.4 Algorithmen

In diesem Abschnitt sollen die für die Erkennung von farblich markierten Objekten wichtigen Algorithmen erläutert werden.

Ziel der Objekterkennung ist es, den Ball und die mit Farbpunkten markierten Roboter zu finden, um den Robotern die Positionen mitteilen zu können (2.3.4).

Hierzu müssen zunächst die Farbflächen auf den Robotern erkannt werden, die dann wiederum Robotern zugeordnet werden.

[13] Der CAN-Bus ist ein Bus welcher mit maximal 1 Mbit je Sekunde Daten übertragen kann und bis zu 6,7 km lang sein kann. Er ist genauer bei (Rickens, 2005) erklärt.

[14] Es gibt bei Firewire einen Stecker mit 6 Polen, welcher auch Strom führt, und auch einen Stecker mit nur 4 Polen, bei welchem die angeschlossenen Geräte nicht mit Strom versorgt werden können.

[15] Ein Dualcore-Prozessor ist ein Prozessor, welcher aus zwei Prozessorkernen besteht, die gleichzeitig Prozesse verarbeiten können.

Die Zuordnung eines Pixels zu einer Farbklasse übernimmt die Farbsegmentierung (2.4.1). Daraufhin müssen die Farbklassen zu Flächen zusammengefügt werden. Mithilfe dieser Flächen findet die Objekterkennung (2.4.2) der einzelnen Roboter statt. Die Ergebnisse werden mittels der Ergebnisausgabe (3.4.6) ausgegeben.

2.4.1 Farbsegmentierung

Bei der Farbsegmentierung geht es darum, ein gegebenes Bild in Farbkategorien aufzuteilen. Dazu muss jeder Farbpunkt des gegebenen Bildes in eine Farbklasse eingeteilt werden.

Um die Farben zu klassifizieren, kann man zwischen verschiedenen Vorgehensweisen wählen:
So kann man mit Grenzwerten in jeder Dimension des Farbraums einen Würfel aufspannen, der die gewünschte Farbe beschreibt.
Ein zweiter Ansatz ist, die Farbwerte nach ihrem Abstand zu einem Referenzfarbton im Farbraum zu gruppieren und so Farbgruppen zu bilden. Dieser Ansatz wäre gut mit Kohonen-Netzen (Kohonen, 2001) realisierbar.

Zu den Anforderungen gehört, eine Klassifizierung in Echtzeit zu ermöglichen. Deshalb wird das erstgenannte Verfahren verwendet. Zudem ist es auch leichter testbar.

Da die zu suchenden Farben bekannt sind, können die Grenzen für die jeweiligen Farben vorab festgelegt werden.

Klassifizierung eines Punktes

Eine Verfahrensweise zur Feststellung der Farbklasse eines Pixels ist, diesen mit allen möglichen Farbklassen zu vergleichen, bis die ihm zugeordnete Farbklasse gefunden ist. Hierzu wären sechs Vergleiche je Farbe und Pixel notwendig (siehe Programmcode 2.6).

Programmcode 2.6: Farbklasse durch Vergleiche bestimmen

```
if(pixel.y >= threshold.y_low
  && pixel.y <= threshold.y_high
  && pixel.u >= threshold.u_low
  && pixel.u <= threshold.u_high
  && pixel.v >= threshold.v_low
  && pixel.v <= threshold.v_high)
   pixel.color_class = current_class;
```

Dieses Verfahren ist jedoch sehr aufwändig, da für jede zu klassifizierende Farbe sechs Vergleiche und ein potenzieller Sprung auszuführen sind. Eine weniger aufwändige Verfahrensweise ist es, diese Vergleiche durch drei Zugriffe auf eine Tabelle (Look-up Tabelle (Miglino u. a., 1995)) zu ersetzen, die mit booleschen Werten gefüllt ist. Als Beispiel mit einem auf acht Werte in der Farbtiefe reduzierten Kamerabild in Programmcode 2.7 gezeigt.

Programmcode 2.7: Initialisierung einer Look-up Tabelle

```
threshold.y = {0,0,0,1,1,1,0,0};
threshold.u = {0,1,1,1,1,0,0,0};
threshold.v = {0,0,1,1,1,1,0,0};
```

Dadurch wird die Frage, ob ein Pixel in einer Klasse ist, mithilfe von zwei logischen ANDs ermittelt (siehe Programmcode 2.8).

Programmcode 2.8: Bestimmung der Farbkategorie durch Nachschlagen in der Tabelle

```
while(pixel.color_class == 0 && current_class != NULL)
{
  if(current_class.threshold.y[pixel.y]
   && current_class.threshold.u[pixel.u]
   && current_class.threshold.v[pixel.v] )
      pixel.color_class = current_class.value;
  current_class = current_class.next_class;
}
```

Wenn man nun die booleschen Werte durch 32 Bit Integerwerte ersetzen und in diesen die Farbklasse unter Verwendung einer Bitmask kodiert, kann man 32 Farbklassen mit derselben Operation abdecken.

Hier als Beispiel die Tabellen mit jeweils nur zwei Farbklassen (siehe Programmcode 2.9).

Programmcode 2.9: Initialisierung einer Look-up Tabelle mit Bitmasken

```
thresholds.y = {00,01,01,11,10,10,00,00};
thresholds.u = {00,10,10,11,11,01,01,00};
thresholds.v = {00,01,01,01,10,10,00,00};
```

Dies vereinfacht das Finden der Farbklasse wie folgt:

Programmcode 2.10: Bestimmung der Farbklasse durch Nachschlagen in der Tabelle

```
pixel.color_class = threshold.y[pixel.y] && threshold.u[pixel.u] && threshold.v
    [pixel.v];
```

Die Zugehörigkeit zu einer Farbklasse kann nun festgestellt werden, indem geprüft wird, ob das jeweilige Bit gesetzt ist (siehe Programmcode 2.11).

Programmcode 2.11: Testen ob eine bestimmte Farbklasse in einer Bitmaske gesetzt ist

```
//testen ob die x-te Farbklasse gesetzt ist
if(pixel.color_class & 2^x)
  ...
```

Speichert man eine Bitmaske in einer Look-up Tabelle, so ist eine sehr gute Performance erreichbar (siehe auch Bruce u. a., 2000).

Farbflächen zusammenfügen

Die einzelnen gefundenen Farbkategorien müssen nun zu Flächen zusammengefügt werden. Hierzu bieten sich die Verfahren *Connected Components Labeling* (Meisel, 2006) oder *Region Splitting and Merging* (Gonzales u. Woods, 2002) an.

Bei *Connected Components Labeling* wird das Bild zeilenweise untersucht und hierbei die jeweilige Farbkategorie der Pixel betrachtet. Wird ein Pixel gefunden, dass in einer anderen Farbgruppe als seine Nachbarpixel ist, wird dieses einer neuen Farbfläche zugeordnet. Werden Pixel aus den selben Farbgruppen gefunden, werden die jeweiligen Farbgruppen der Pixel zusammgengefügt.

Bei *Region Splitting and Merging* wird das Bild solange in Abschnitte geteilt, bis in dem übrig bleibenden Abschnitt alle Pixel einer Farbklasse zugeordnet werden können. Darauf werden die angrenzenden Abschnitte daraufhin untersucht, zu welcher Farbklasse sie gehören. Werden zwei aneinander grenzende Abschnitte, welche der selben Farbklasse angehören gefunden, werden diese Abschnitte wieder zusammengefügt.

2.4.2 Objekterkennung

Ziel der Objekterkennung ist es, die Roboter und den Ball auf dem Spielfeld zu erkennen. Hierzu werden die Farbflächen als Eingabe verwendet.

Die Roboter sowie der Ball bekommen je eine Markierungsfarbe. Die Objekterkennung ermittelt danach aus den von der Farbsegmentierung (2.4.1) erhaltenen Farbflächen die Position und Ausrichtung der Objekte auf dem Spielfeld (2.3.1).

Der Ablauf gestaltet sich dabei im Groben wie folgt:

- Um die Roboter zu finden, werden alle Farbflächen gesucht, die in der Nähe der Markierungsfarbe sind.

- Von diesen werden die passendsten Farbflächen herausgesucht (siehe 4.5.2).
- Die gefundenen Farbflächen werden daraufhin im Uhrzeigersinn sortiert.

Als Ergebnis wird eine Liste von gefundenen Objekten weitergegeben, welche die Postionen des Balls und der Roboter enthält.

2.4.3 Nebenläufigkeit

Die weiche Echtzeit ist nur für die Weitergabe der Informationen an die Roboter relevant, da die Anzeige des Kamerabildes und der erkannten Farbflächen sowie der erkannten Objekte, nur die Funktion hat, das Ergebnis zu kontrollieren.

Um die Anzeige von der Verarbeitung zu trennen, werden diese in verschiedene Verarbeitungsstränge (*Threads*) aufgeteilt.

2.5 Verwandte Arbeiten

Object Tracker werden für viele verschiedene Systeme eingesetzt, so zum Beispiel von der Industrie, um Objekte auf dem Fließband zu erkennen und zu sortieren, oder aber in Versuchslaboren wie beim Massachusetts Institute of Technology (Abbildung 2.19).

Abbildung 2.19: Schwarm Gesundheitsmanagement in dem Aerospace Controls Labor (http://vertol.mit.edu/) am MIT

Da es sich bei dem Ziel dieser Arbeit um ein System handelt, welches für den Roboterfussball geschaffen wird, wird auch der Blick nach verwandten Arbeiten sich auf diesen Bereich konzentrieren.

Vergleichbare Systeme gibt es in der "Small Size Robot League" des "RoboCup". Dort müssen zwei Teams von kleinen Robotern[16] gegeneinander spielen. Diese Teams werden jeweils von einem Computer per Funk koordiniert, welcher die Spielsituation mit einer Kamera bestimmt (Abbildung 2.20).

[16] Die Roboter dürfen maximal 180mm Durchmesser haben und 150mm hoch sein.

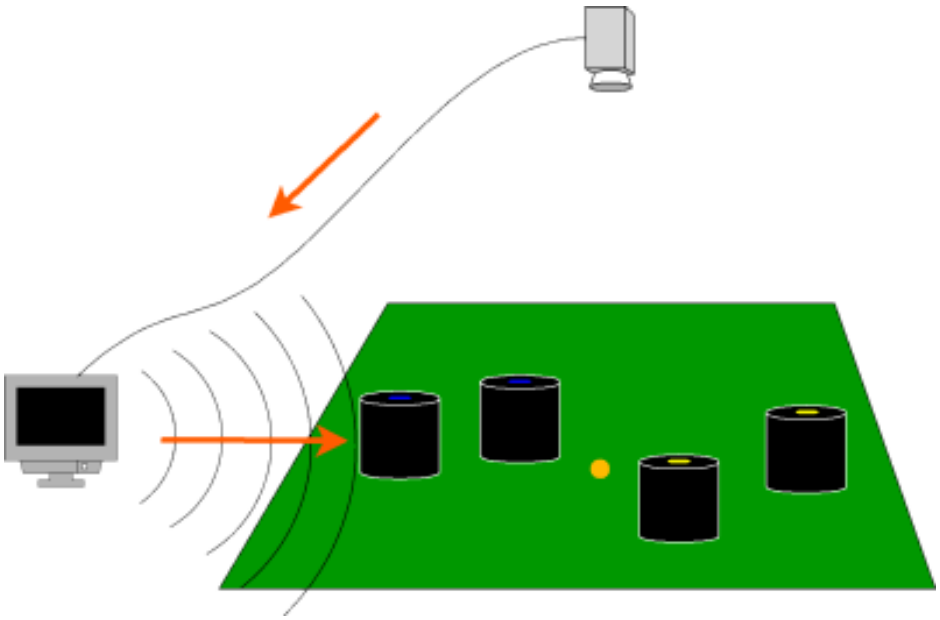

Abbildung 2.20: Die Vorgehensweise beim "RoboCup"

Die meisten der Teams benutzen einen omnidirektionalen Antrieb (Abbildung 2.21) mit 4 Rädern, die eine Bewegung in alle Richtungen und auch das Drehen während des Fahrens ermöglicht.

Von diesen Systemen ist jeweils das Bildverarbeitungssystem für diese Arbeit von Interesse und es werden zunächst die Systeme der ersten drei Teams des letzten RoboCups vorgestellt.

Leider sind von diesen Teams meist nur die Doktor- oder Diplomarbeiten verfügbar. Die jeweils eingesetzte Software wird - wenn überhaupt - nur in alten Versionen zu Verfügung gestellt.

Skuba

Das Team Skuba[17] kommt von der Kasetsart Universität in Bangkok.
Es benutzt einen Camcorder, welcher die Bilder bei 50 Hz mit einer Auflösung von 640x480 Pixeln liefert.
Das Bild wird in einem ersten Verarbeitungsschritt aus dem RGB in das HSV-Bildformat umgewandelt, um besser mit Unterschieden in der Beleuchtung umgehen zu können.

[17] http://iml.cpe.ku.ac.th/skuba/

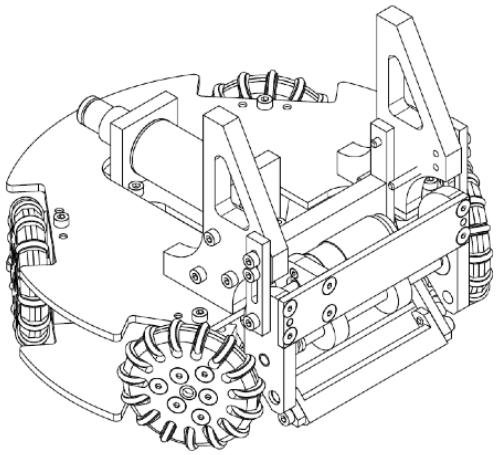

Abbildung 2.21: Der omnidirektionale Antrieb des Plasma-Z Teams 2006

Das System arbeitet mit einer Verzögerung von etwa drei bis fünf Bildern (etwa 150 Millisekunden). Diese Verzögerung soll durch eine Abschätzung über die vermutliche Position vermindert werden (Srisabye u. a., 2006).

FU-Fighters

Die FU-Fighters[18] von der freien Universität Berlin benutzen ein Bildverarbeitungssystem, welches *Region Growing* (von Hundelshausen, 2005) benutzt.

5dpo

Das Team 5dpo[19] wurde zweiter bei dem RoboCup 2006.
Es benutzt zwei Firewire-Kameras mit einer Auflösung von je 780x582 Pixeln und klassifiziert die Farben mit einem Fuzzy-System, um kontinuierliche Farbgruppen zu erhalten. Der Teambeschreibung des Teams 5dpo (Costa u. a., 2004) ist auch zu entnehmen, dass das Bildverarbeitungsystem mit einer Verzögerung von mindestens 50 Millisekunden arbeitet.

[18] http://robocup.mi.fu-berlin.de/pmwiki/pmwiki.php
[19] http://paginas.fe.up.pt/~robosoc/

CMDragons

Die CMDragons[20] von der Carnegie Mellon Universität[21] haben den RoboCup 2006 gewonnen.

Das von diesem Team benutzte Bildverarbeitungssystem benutzt die Bibliothek CMVision (Bruce u. a., 2000).

Tekkotsu

Tekkotsu[22] ist ein *OpenSource* Framework für die Entwicklung von Programmen für den AIBO[23].

Das Tekkotsu Framework verwendet ebenfalls die Bibliothek CMVision (Bruce u. a., 2000) für die Farberkennung.

CMVision

CMVision ist ein Farbsegmentierungssystem für Interaktive Roboter (Bruce u. a., 2000). Es bietet eine sehr schnelle Möglichkeit Farbflächen aus einem Bild zu extrahieren und wurde auch in dieser Arbeit (4.3.2) sowie in vielen Anderen verwendet.

An der Hochschule für Angewandte Wissenschaften Hamburg

Rainer Balzerowski hat 2002 ein Webcam basiertes Kamera-System für Lego-Mindstorms realisiert: "Realisierung eines Webcam basierten Kamera Systems für mobile Roboter"(Balzerowski, 2002).
Arne Diekmann "Verbesserung visueller Objekterkennung für mobile Roboter" (Dieckmann, 2003) entwickelte 2003 ein System, um dem Pioneer(2.3.3) eine besser Objekterkennung zu ermöglichen.
Ilia Revout "Design und Realisierung eines Frameworks für Bildverarbeitung" (Revout, 2003) entwickelte ein System für allgemeine Bildverarbeitung, welches den Einsatz mehrerer Filter ermöglichte um so vielfältige Probleme im Bereich der Bildverarbeitung zu meistern, dieses System ist jedoch mit dem Ziel entwickelt worden neue Algorithmen

[20] http://www.cs.cmu.edu/~robosoccer/small/
[21] http://www.cs.cmu.edu/
[22] http://www.cs.cmu.edu/~tekkotsu/
[23] Der AIBO ist ein Roboter-Hund von Sony dessen Produktion im März 2006 eingestellt wurde.

schnell implementieren zu können, nicht jedoch mit dem Ziel der Echtzeitfähigkeit.

Die Arbeiten, die als Vorläufer des angestrebten System angesehen werden können, stammen von Oliver Schmidt "Entwicklung eines Mehrbenutzer-Webservice am Beispiel eines Kamera-Servers für mobile Roboter" (Schmidt, 2005) und Michael Gottwald "Webcam basiertes Kamerasystem für autonome Roboter: Erste Konzeption Webcam basiertes Kamerasystem für autonome Roboter: Erste Konzeption" (Gottwald, 2005). Hierbei hat sich jedoch Oliver Schmidt auf den Aspekt eines sicheren Mehrbenutzer-Webservice konzentriert und Michael Gottwald lediglich ein Konzept für weitere Arbeiten geschaffen.

3 Design

In diesem Kapitel soll eine Übersicht über das System vermittelt werden.

Hierfür wird zunächst die Architektur des Gesamtsystems (3.1) betrachtet, um einen Überblick zu verschaffen. Darauf wird die Abfolge der Arbeitsschritte in dem Abschnitt "Programmablauf"(3.2) gezeigt. Nachdem diese Grundlagen geschaffen wurden, werden die einzelnen Komponenten genauer erläutert (3.4). Dieses Kapitel schließt ab durch die Auswahl eines geeigneten Farbmodells (3.5).

Die jeweilige Implementierung wird in dem folgenden Kapitel "Realisierung"(4) beschrieben.

3.1 Systemarchitektur

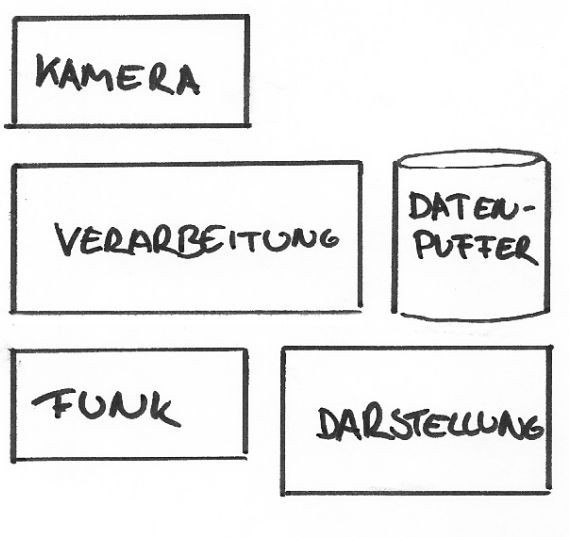

Abbildung 3.1: Die Systemarchitektur im Überblick

Das System besteht aus einer Kamera, einer Verarbeitungseinheit, einer Funkeinheit und einer Darstellungseinheit. Des Weiteren sind Datenpuffer vorhanden, welche den synchronisierten Datenaustausch[1] zwischen den Verarbeitungssträngen ermöglichen.

3.2 Programmablauf

Der Programmablauf des Gesamtsystems gestaltet sich wie in Abbildung 3.2 dargestellt.

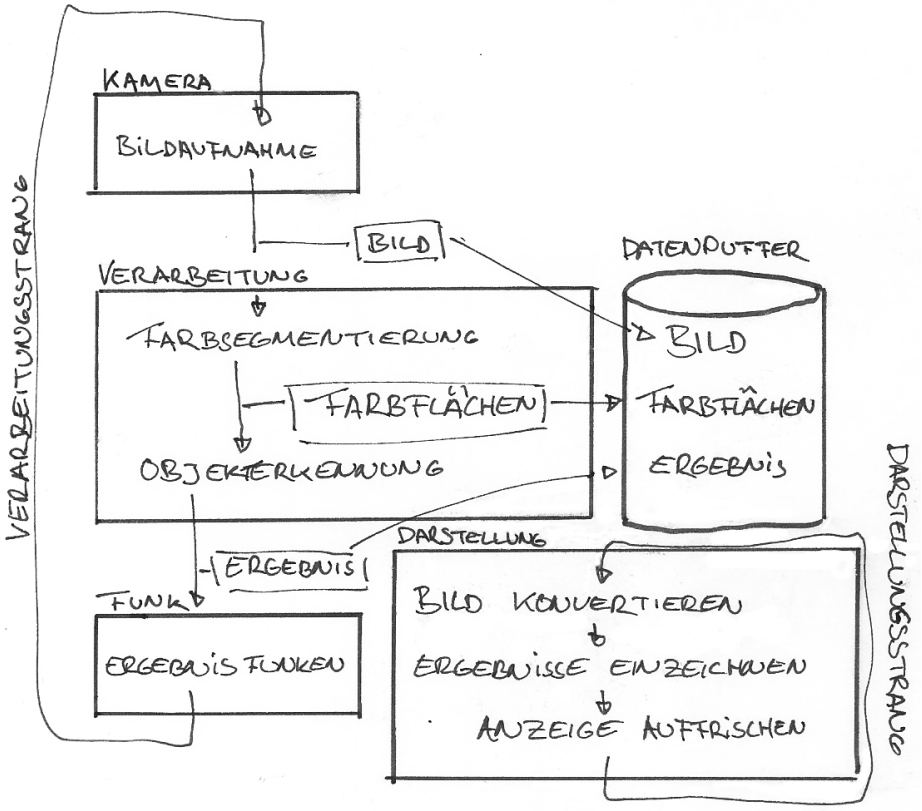

Abbildung 3.2: Der Ablauf des Programms

Es gibt zwei nebenläufige Verarbeitungsstränge (2.4.3).

Ein Verarbeitungsstrang kümmert sich um die Analyse der Bilder und die Weitergabe der gewonnen Informationen an die Konsumenten wie zum Beispiel Roboter und Darstellung. Dies ist auch der Teil, für den die Echtzeit-Anforderungen (2.1) gelten, da es in unserem

[1] Synchronisierter Datenaustausch sichert bei gleichzeitigem Zugriff von zwei Verarbeitungssträngen, dass keine Daten korrumpiert werden.

Kontext einem Fußball spielendem Roboter wenig nutzt, wenn er erfährt, wo sich der Ball vor drei Sekunden befunden hat.
Zunächst werden Bilder von der Kamera erwartet.
Das gewonnene Bild wird in seine Farbflächen zerlegt.
Die Farbflächen-Informationen werden in der Objekterkennung zur Identifizierung von Ball und Roboter verwendet.
Zuletzt werden die gewonnen Positionen und Ausrichtungen an die Roboter weitergegeben und für die Anzeige gespeichert.

Der andere Verarbeitungsstrang kümmert sich um die Anzeige der Bilder. Dafür wird das Kamerabild in das RGB-Format konvertiert, die Ergebnisse werden eingezeichnet und sowohl das Kamerabild als auch das Bild der segmentierten Farben auf dem Monitor dargestellt.

3.3 Farbkonfiguration

Die Farbkonfiguration ermöglicht es, die einzelnen Farbklassen einzustellen und anhand eines Histogramms festzustellen, welche Farben gerade von der Kamera gesehen werden (Abbildung 3.3).

Abbildung 3.3: Anwendungsfälle für die Farbkonfiguration

3.4 Klassendiagramme

In diesem Abschnitt werden die Zusammenhänge zwischen den einzelnen Komponenten des Systems dargestellt.
Dazu wird jeweils mithilfe eines UML-Diagramms (Oesterreich, 2006) eine Komponente mit deren Funktionen und Abhängigkeiten zu anderen Komponenten beschrieben.

Auf die Implementation der Komponenten wird genauer in Abschnitt 4.5 eingegangen.

3.4.1 Übersicht

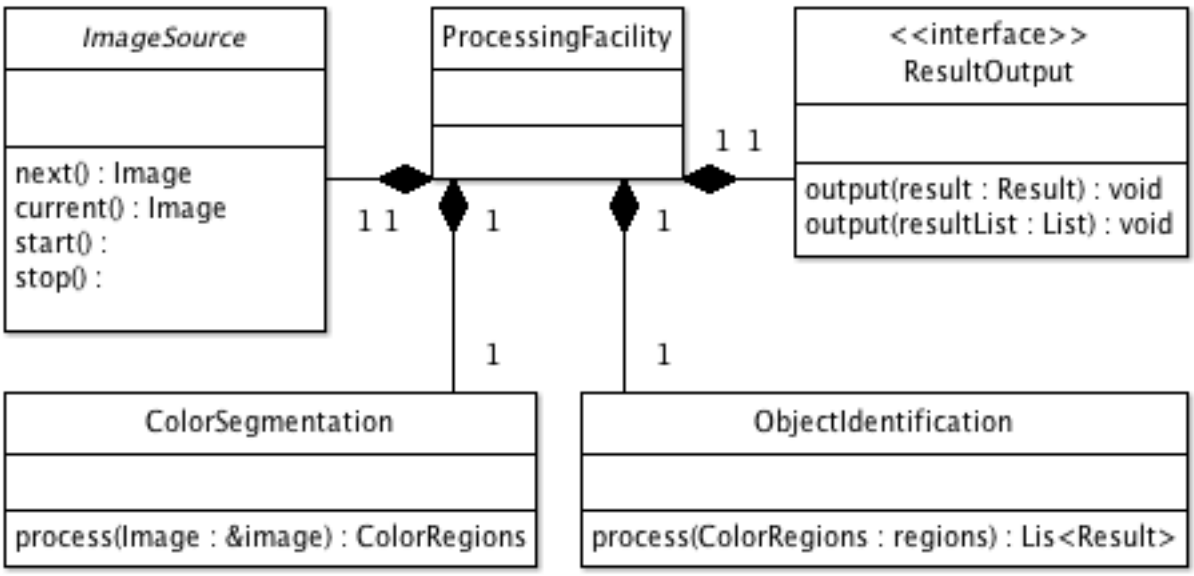

Abbildung 3.4: Klassendiagramm "Übersicht"

Die Ablaufsteuerung (*ProcessingFacility* Abbildung 3.4) ist zuständig für die richtige Abfolge der Arbeitsschritte.
Sie besteht aus einer Bildquelle (*ImageSource* 3.4.2), einem Farbsegmentierer (*ColorSegmentation* 3.4.3), einer Objekterkennung (*ObjectIdentification* 3.4.4) und der Ergebnisausgabe (*ResultOutput* 3.4.6).

Die Ablaufsteuerung ist der "Mediator"(siehe Gamma u. a., 1995, Seite 273-282) zwischen den Komponenten des Systems. Sie kapselt die Interaktionen zwischen den Teilsystemen und ermöglicht so eine lose Kopplung zwischen den Komponenten. Dies erleichtert es auch, einzelne Komponenten auszutauschen, wie es die Anforderung "Erweiterbarkeit" fordert (2.1).

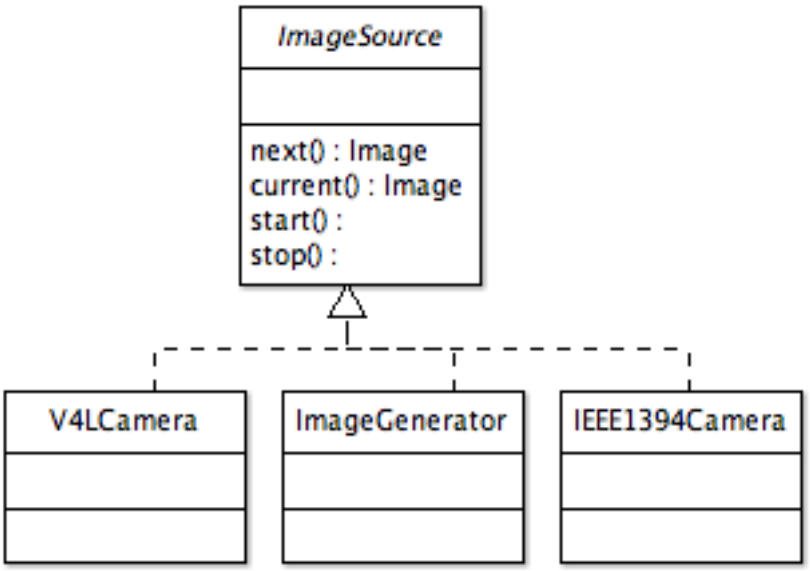

Abbildung 3.5: Klassendiagramm "Bildquelle"

3.4.2 Bildquelle

Die Bildquelle (*ImageSource* Abbildung 3.5) liefert dem Programm die zu verarbeitenden Bilder. Die Bilder können aus einer Firewire-Kamera, einer V4L-Kamera[2] oder einem Bildgenerator *ImageGenerator* gewonnen werden.

Im Folgenden werden diese Bilder "Kamerabild" genannt.

Der Bildgenerator generiert Kamerabilder, um für Testfälle (*test cases*) reproduzierbare Ausgangssituationen zu schaffen.
Die Firewire-Kamera und die V4L-Kamera sind zwei Möglichkeiten, das Programm mit weiterzuverarbeitenden Bildern zu versorgen.

Jeder Bildgenerator kann erneut das zuletzt aufgenommene Bild oder das nächste Bild liefern. Wenn das nächste Bild geliefert werden soll, dieses aber noch nicht vorliegt, dann wartet der Bildgenerator, bis er ein Bild liefern kann.
Er kann zudem noch gestartet und gestoppt werden, um zum Beispiel die Kamera zu starten oder zu stoppen.

[2] **V**ideo**4L**inux ist eine Schnittstelle, um unter Linux auf diverse Kameras wie USB-Kameras oder Fernseh-Karten zuzugreifen

3.4.3 Farbsegmentierung

Die Farbsegmentierung ist verantwortlich für die Zerlegung des Kamerabildes (3.4.2) in seine einzelnen Farbkomponenten.

3.4.4 Objekterkennung

Die Objekterkennung ermittelt mithilfe des segmentierten Bildes, an welchem Ort sich welche Objekte befinden und wie diese ausgerichtet sind.

3.4.5 Ausgabefilter

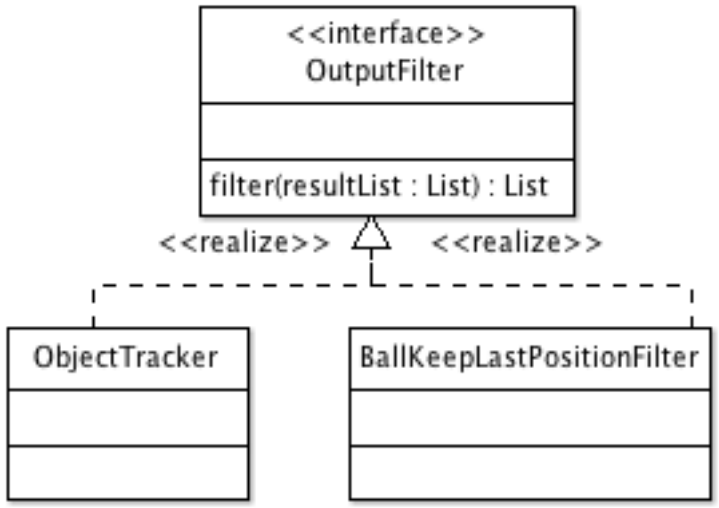

Abbildung 3.6: Klassendiagramm "Ausgabefilter"

Die Ausgabefilter beschränken oder verändern die Ausgabe, indem alle Ergebnisse zunächst durch diese geleitet werden.
Es gibt hierbei zwei Ausgabefilter (Abbildung 3.7):
Der eine Ausgabefilter merkt sich die letzte Position des Balles und fügt diese in die Liste ein, falls kein Ball gefunden wurde (*BallKeepLastPositionFilter*).

Der andere Ausgabefilter errechnet anhand der letzten Position eines Objektes die Geschwindigkeit und die sich dadurch ergebende vorraussichtliche neue Position, die dann anstelle der Originalposition in die Liste eingefügt wird (*ObjectTracker*).

3.4.6 Ergebnisausgabe

Die Ergebnisausgabe ist für die Weiterverwendung der ermittelten und gefilterten Ergebnisse zuständig. Die Ergebnisse sind die Position des Balls sowie die Positionen und Ausrichtungen der Roboter.

Die Ergebnisausgabe wird implementiert durch

- eine Liste, welche dazu verwendet wird, die Ergebnisse an alle in ihr enthaltenen Ergebnisausgaben weiterzuleiten.
  Diese Liste wurde als "Composite"(siehe Gamma u. a., 1995, Seite 163-173) implementiert, sodass sie sich wie eine einfache Ergebnisausgabe verhält.

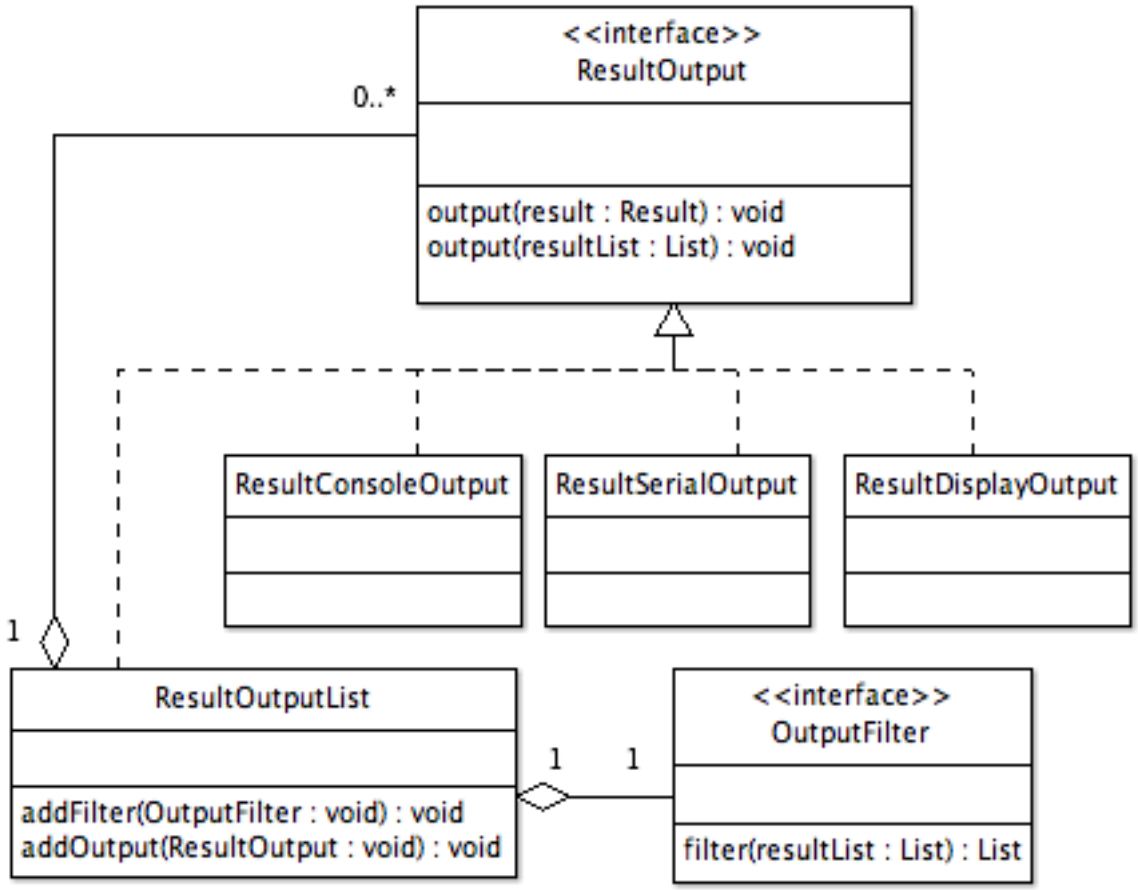

Abbildung 3.7: Klassendiagramm "Ergebnisausgabe"

- eine Ausgabe über den seriellen Port, damit das Ergebnis an die Roboter gefunkt werden kann.
- eine Anzeige auf dem Ergebnisbild der Farbsegmentierung, um die Erkennung der Objekte zu überprüfen.
- eine Ausgabe auf der Konsole, welche der auf dem seriellen Port entspricht, um eine Ausgabe zur Überprüfung zu haben, deren Rechenaufwand sehr gering ist.

3.4.7 Bild

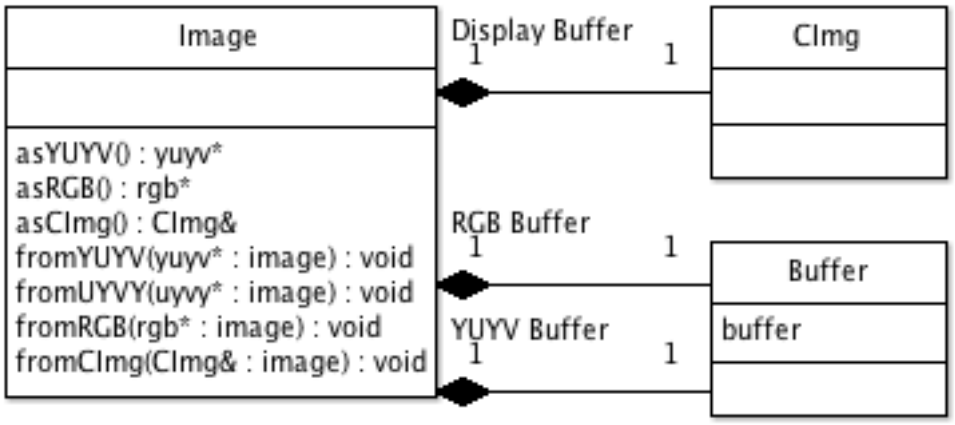

Abbildung 3.8: Klassendiagramm "Bild"

Die Klasse "Bild"(*Image* Abbildung 3.8) dient als reine Datenklasse für das Bild. Sie kann auch nötige Konvertierungen in die Formate

- YUYV (2.2.4)
- RGB (2.2.1)
- CImg (4.3.6)

vornehmen. Hierfür hat sie jeweils einen Puffer, um sich die Ergebnisse zu merken.

3.4.8 Ringpuffer

Der "Ringpuffer"(*RingBuffer* Abbildung 3.9) dient dem Austausch von Daten zwischen verschiedenen Threads.

Der Ringpuffer hat eine feste Größe. Er bietet verschiedene Möglichkeiten, ihm Elemente hinzuzufügen oder zu entnehmen. Diese Zugriffe werden durch Semaphoren geschützt (Tannenbaum, 2003).

- *Normales Hinzufügen und Entfernen* (*add*, *remove*): bei einem vollen bzw. leeren Ringpuffer wartet diese Methode, bis Platz für ein Element frei ist bzw. ein Element verfügbar ist.

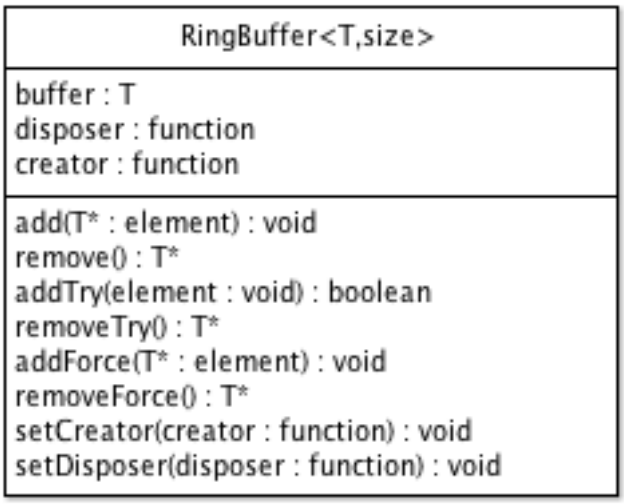

Abbildung 3.9: Klassendiagramm "Ringpuffer"

- *Probiertes Hinzufügen und Entfernen* (*tryAdd*, *tryRemove*): diese Methode, auf den Ringpuffer zuzugreifen, versucht, ein Element zu entfernen oder hinzuzufügen. Sollte dies nicht möglich sein, so wartet sie nicht, sondern gibt ein entsprechendes Ergebnis zurück.
- *Erzwungenes Hinzufügen und Entfernen* (*forceAdd*, *forceRemove*): für diese Methode können *Callbacks*[3] angegeben werden (*setCreator*, *setDisposer*), die bei vollem Ringpuffer Elemente entsorgen[4] oder bei leerem Ringbuffer Elemente bereitstellen.

3.5 Farbmodell

Im RGB-Modell ist es relativ kompliziert, eine Farbe unabhängig von der Helligkeit abzubilden (2.2.1).

Da im YUV-Farbmodell zwei Achsen für die Farbauswahl und eine Achse für die Helligkeit der Farbe benutzt werden, lässt sich hier eine Farbe leicht durch ein Farbintervall sowie eine maximale und minimale Helligkeit beschreiben.

Da zudem die in der Hochschule für Angewandte Wissenschaften Hamburg verfügbaren Firewire-Kameras alle das YUV-Format direkt unterstützen, wurde dieses verwendet.

[3] Ein *Callback* ist ein ausführbarer Programmcode, der einer Funktion als Argument übergeben wird. Diese Funktion kann dann auf diesen *Callback* zugreifen, um bestimmte Aufgaben zu erledigen.

[4] Dieses wurde z. B. verwendet, um Objekte, die in dem Ringpuffer keinen Platz mehr fanden, wieder zu verwenden.

4 Realisierung

Not to be in production is to be spending money without making money (Beck, 2000, Seite 131)

Für die Realisierung ist eine iterative Vorgehensweise mit frühzeitigem Einsatz in einem Pilotprojekt gewählt worden (*continuous integration* (Beck, 2000)). Nach der Fertigstellung des fortgeschrittenen Prototyps ist die Lösung gleich in experimenteller Umgebung in dem Roboter Projekt des Wintersemesters 2006/2007 eingesetzt worden. Das intensive Feedback durch die Nutzer vereinfachte die Fehlerfindung erheblich und machte die Feinabstimmung der Anforderungen parallel zur Realisierung erst möglich.

So kam zum Beispiel während des Roboter Projektes die Anforderung auf, dass der Ball mit seiner letzten Position auch gesendet werden soll, wenn dieser nicht mehr vom System erkannt wird. Diese Anforderung konnte schnell erfüllt werden (4.5.3).

Im Folgenden werden die gewählte Programmiersprache, die eingesetzte Entwicklungsumgebung und die Auswahlkriterien dafür beschrieben.

Um die Entwicklungszeit zu senken, baut die Lösung auf bereits existierenden Bibliotheken auf (*buy versus build* (Brooks, 1995)). Dies senkt zusätzlich das Risiko, weil die so eingebundenen Funktionen nicht mehr entwickelt und - in dem Maße, wie das im Rahmen einer Eigenentwicklung notwendig ist - getestet werden müssen.

Bevor in diesem Kapitel auf die konkrete Realisierung eingegangen wird, beschäftigt es sich mit der verwendeten Programmiersprache (4.1) und den verwendeten Programmen (4.2). Es geht auch zunächst auf die benutzten Bibliotheken (4.3) und deren Verwendung ein. Danach beschreibt es die Realisierung der einzelnen Komponenten (4.5) sowie die Optimierung der Anwendung (4.6)

4.1 Programmiersprache

Als Programmiersprache wurde C++ gewählt, da diese viele der Anforderungen (2.1) erfüllt.

So ist C++ auf vielen Systemen verfügbar und genügt somit der Portierbarkeit. Des Weiteren hat C++ keinen Garbage Collector[1], was dazu führt, dass das Laufzeitverhalten leichter zu bestimmen ist, wodurch die Anforderung der weichen Echtzeit nicht mehr durch die Garbage Collection gestört werden kann.

Leider ist dies auch mit einem erhöhtem Programmieraufwand verbunden und mit der Gefahr, dass man den angeforderten Speicher nicht wieder freigibt. Dadurch passiert es in Programmen ohne Garbage Collector leichter, dass man Speicherlecks (4.6.2) hat.

Außerdem sind im Bereich der Bildverarbeitung und der Roboter viele Bibliotheken für C++ vorhanden, was die Entwicklungszeit erheblich verkürzt.

4.2 Entwicklungsumgebung

Trac[2] wurde als Bug-Tracker für die Projektplanung und Subversion[3] als Versionskontrollsystem benutzt (Fogel, 2005).

Unter Linux wurden für die Entwicklung KDevelop[4], Emacs[5], Valgrind[6] und KCachegrind [7] benutzt.

Unter Mac OS X kam als IDE XCode[8] zum Einsatz und MallokDebug zum Finden von Speicherlecks.

4.3 Bibliotheken

In diesem Kapitel wird eine Übersicht über die verwendeten Bibliotheken nach folgenden Kriterien gegeben.

- Funktionsumfang?
- Wo wurde die Bibliothek entwickelt?
- Wofür wird sie noch verwendet?

[1] Garbage Collection ist ein Teil der Speicherverwaltung, welcher sich um die Freigabe von nicht mehr verwendetem (Arbeits-)Speicher kümmert.
[2] http://trac.edgewall.org/
[3] http://subversion.tigris.org/
[4] http://www.kdevelop.org/
[5] http://www.gnu.org/software/emacs/
[6] http://valgrind.org/
[7] http://kcachegrind.sourceforge.net/
[8] http://developer.apple.com/tools/xcode/

- Wie wurde die Bibliothek in dieser Bachelorarbeit verwendet?
- Was für Probleme gab es bei der Verwendung?
- Unter welcher Lizenz steht die Bibliothek?

4.3.1 Libdc1394

Libdc1394 ist eine Bibliothek für den Zugriff auf Firewire-Kameras, die den IIDC/DCAM Standard[9] erfüllen.

Die Bibliothek ist in Version 1.2 nur unter Linux lauffähig, ab Version 2 auch unter Mac OS X.

Zu Beginn der Entwicklung war "Libdc1394" als Releasekandidat[10] verfügbar und bis Ende 2006 sollte die finale Version erscheinen. Jedoch ist sie bis Mitte März 2007 noch nicht fertig gestellt worden.

Die Entwicklung basiert deshalb auf dem oben erwähnten Releasekandidaten.

4.3.2 CMVision

CMVision wurde entwickelt, um Robotern das Farbsehen in Echtzeit zu ermöglichen.

Die Bibliothek wurde an der Carnegie Mellon University in Pittsburg, Pennsylvania in dem *CORAL Groups Color Machine Vision Project* entwickelt.

Die Bibliothek wird derzeit in mehreren Projekten verwendet, welche sich mit Robotern und Farbsehen beschäftigen. So zum Beispiel in Player[11], einem Rahmenwerk zum Entwickeln und Testen von Software für Roboter und Sensoren und in Tekkotsu[12], einem Entwicklungs-Rahmenwerk für den Sony AIBO Roboter.

Bei der Verwendung der Bibliothek gab es keine größeren Probleme, lediglich kleine Anpassungen waren nötig. Allerdings brachte die Konfiguration der Farbgrenzwerte (3.3) Zusatzaufwand mit sich, der sich jedoch zügig bewältigen ließ.

Die Bibliothek ist unter der GPL (Free Software Foundation, 2007a) veröffentlicht.

[9] Ein von der 1394 Trade Association (http://www.1394ta.org/) spezifiziertes Protokoll. Nicht zu verwechseln mit Firewire *Camcordern* oder DV Kameras

[10] Ein Releasekandidat ist ein Programm in dem Zustand, wie es später auch letztlich werden soll. Es werden in dieser Phase möglichst nur noch Fehler behoben und keine neuen Funktionen zu dem Programm hinzugefügt.

[11] http://playerstage.sourceforge.net/

[12] http://www.cs.cmu.edu/~tekkotsu/

4.3.3 GEOS

GEOS (Geometry Engine - Open Source) http://geos.refractions.net/ ist eine Bibliothek, welche grundlegende geometrische Algorithmen für im Raum verteilte Objekte zur Verfügung stellt. Die Bibliothek ist eine Portierung der Java Topology Suit[13] .

GEOS wird unter anderem in der Datenbank PostgreSQL[14] als Bibliothek für die Erweiterung PostGIS[15] genutzt, um *geographische Objekte* in Datenbanken speichern und abfragen zu können.

In dieser Arbeit wurde die Bibliothek während der Objekterkennung (2.4.2) verwendet.

Die Bibliothek ließ sich gut und ohne Probleme einbinden. Man musste nur dafür sorge tragen, dass sämtliche Objekte, die von Ihr verwendet wurden, auch wieder gelöscht werden.

Die Bibliothek ist unter der GPL (Free Software Foundation, 2007a) veröffentlicht.

4.3.4 CPPUnit

"CPPUnit"[16] ist eine C++ Portierung des JUnit[17] Rahmenwerks für Unit Tests.

Die Bibliothek ist unter der GPL (Free Software Foundation, 2007a) veröffentlicht.

4.3.5 ConfigFile

"ConfigFile"[18] ist eine C++ Klasse von Rick Wagner, um Konfigurationsdateien zu lesen.

In dieser Arbeit wurde die Klasse verwendet, um eine Konfigurationsdatei zu lesen.

Die Bibliothek ist unter der MIT Lizenz (Massachusetts Institute of Technology) freigegeben.

[13] http://www.jump-project.org/project.php?PID=JTS\&SID=OVER
[14] http://www.postgresql.org/
[15] http://www.postgis.org/
[16] http://sourceforge.net/projects/cppunit/
[17] http://www.junit.org/
[18] http://www-personal.umich.edu/~wagnerr/ConfigFile.html

4.3.6 CImg

"CImg"[19] ist eine Bibliothek für Bildbearbeitung und Bilddarstellung.

Die Bibliothek wurde in dem Odyssee Labor (http://www-sop.inria.fr/odyssee/) an dem Inria Sophia-Antipolis Institut[20] in Frankreich entwickelt und wird dort für Vorlesungen aus dem Bereich Bildverarbeitung benutzt.

In dieser Arbeit wurde die CImg-Bibliothek verwendet, um sowohl das Kamerabild als auch das Ergebnisbild darzustellen. Des Weiteren wurde "CImg"verwendet, um ein Histogramm des Kamerabildes darzustellen und mithilfe von diesem die Anwendung zu konfigurieren.

Der Vorteil bei der Verwendung dieser Bibliothek lag darin, dass sie unter Mac OS X, Linux und Windows arbeitet und nur eine einzelne *Header-Datei*[21] erfordert.

CImg speichert die Bilddaten intern nicht als RGB-Tupel sondern zunächst die komplette rote, die grüne und dann die blaue Ebene, also in einem planaren-Format.

Dies führt dazu, dass zwischen den verschiedenen Speicherformen konvertiert werden muss.

Die Bibliothek ist unter der CeCILL-C Lizenz (lizenz:cecill, 2007) freigegeben, welche ähnlich der LGPL (Free Software Foundation, 2007b) ist.

4.4 Nebenläufigkeit

Da die Anzeige der Bilder etwa zehn mal so lange dauert[22] wie die reine Verarbeitung und nur für die visuelle Kontrolle der Ergebnisse zuständig ist, wurden die Anzeige und die Verarbeitung getrennt.

Dadurch wurde es nötig, eine Möglichkeit zu schaffen, Daten zwischen den Verarbeitungssträngen auszutauschen. Dafür wurde ein Ringpuffer mit fester Größe programmiert, der die Möglichkeit bietet, Objekte, die überlaufen würden, weiterzugeben, sodass die Objekte weiter verwendet werden können. Dieser Ringpuffer wurde im weiteren Verlauf auch verwendet, um die Ergebnisse aus der Objekterkennung mehrfach zu verwenden.

[19] | http://cimg.sourceforge.net/

[20] http://www-sop.inria.fr/

[21] Eine Header-Datei ist eine Textdatei, die Deklarationen in C/C++ enthält, welche während der Programmerstellung eingebunden werden.

[22] Bei einer Auflösung von 640x480 Pixeln braucht die Anzeige bei 15 FPS 100% CPU, die Verarbeitung bei 30 FPS etwa 10-20% CPU. Die Verarbeitung könnte also mit etwa 150-300 FPS auf einer CPU laufen und ist somit mindestens zehn mal so schnell.

4.5 Komponenten

In diesem Abschnitt werden die Realisierungen der einzelnen Komponenten des Systems beschrieben.

4.5.1 Farbsegmentierung

Die Farbsegmentierung verwendete die Kamera mit einer Auflösung von 640x480 Pixeln. Sie konnte durch Verwendung der Bibliothek CMVision (Beschreibung siehe 4.3.2) realisiert werden (Abbildung 4.5.1 und 4.5.1 (Bruce u. a., 2000)). Diese Bibliothek benutzt für die Farbsegmentierung eine Look-up Tabelle (Miglino u. a., 1995) und für das berechnen der Flächen einen *union based tree* mit *path compression*

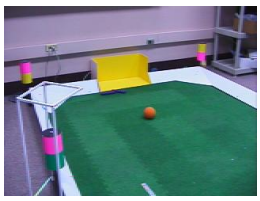

Abbildung 4.1: Videobild nach der Aufnahme

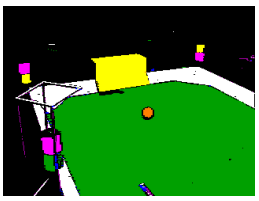

Abbildung 4.2: Bild nach der Farbklassifizierung

4.5.2 Objekterkennung

Die Aufgabe der Objekterkennung ist, möglichst viele Objekte auf dem Kamerabild zu erkennen und deren Position und Ausrichtung zu bestimmen.

Die Objekterkennung erhält hierzu als Eingabe die Farbflächen, welche in der Farbsegmentierung (4.5.1) ermittelt wurden.

Das Auffinden von Objekten beschränkt sich in unserem Anwendungsfall auf zwei konkrete Objekte:

- Einen Ball, der durch eine bestimmte Farbe markiert ist (hier Gelb).
- Einen Roboter, der mit vier Farbpunkten markiert ist. Von den Farbpunkten ist ein Farbpunkt in einer Farbe, nach welcher die Objekterkennung sucht (hier Grün). Die drei weiteren Farbpunkte werden genutzt, um die Ausrichtung des Roboters zu bestimmen (Grün ist immer vorne) und den Roboter anhand der Farbreihenfolge eindeutig zu identifizieren (Abbildung 4.5.2).

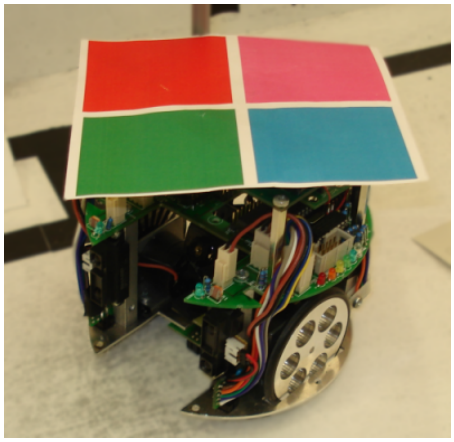

Abbildung 4.3: Ein mit den Farben Grün, Rot, Pink und Türkis markierter Roboter, welcher im System als "grpt" identifiziert wird.

Abbildung 4.4: Ein mit Gelb markierter Ball

Filterung der Farbflecken

Um die Anzahl der potentiell möglichen Farbflecken einzuschränken, werden diese gefiltert. Dazu müssen die Farbflecken folgende Kriterien erfüllen:

- Das Verhältnis der Seitenlängen eines um die Farbflächen gezeichneten Rechtecks darf nicht größer als 2:1 oder kleiner als 1:2 sein.
- Das Objekt muss mindestens einen bestimmten Prozentsatz der Fläche ausfüllen, die ein das Objekt einfassendes Rechteck beschreibt.
- Das Objekt muss eine Mindestfläche ausfüllen.
- Das Objekt darf nicht größer als eine Maximalfläche sein.

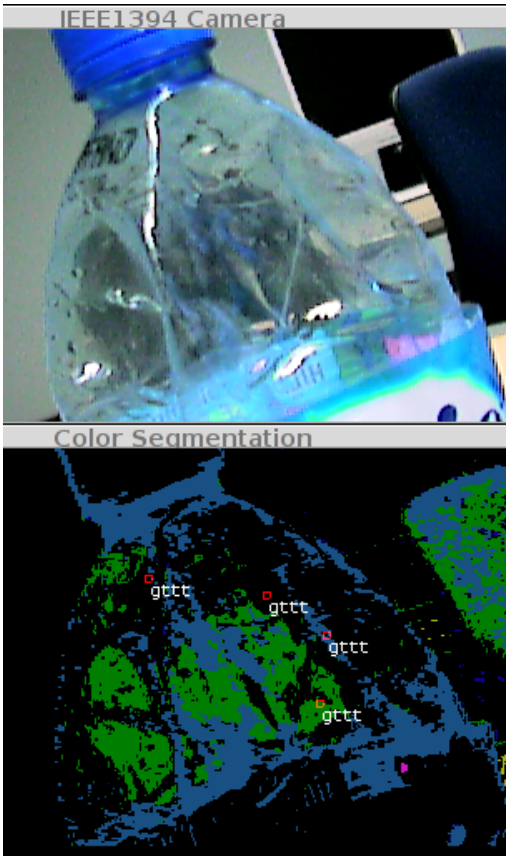

Abbildung 4.5: Fälschlich erkannte "Roboter", ohne Filterung

Abbildung 4.6: Fälschlich markierte Farbflächen, ohne Filterung

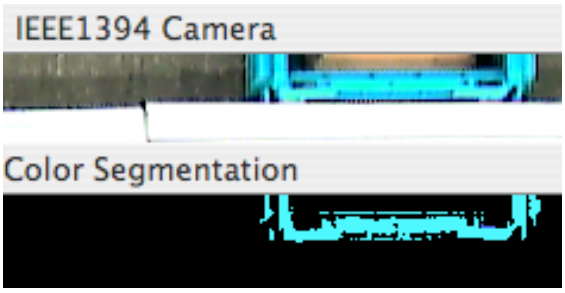

Abbildung 4.7: Eine Kiste, welche dank der Filter nicht in Betracht gezogen wird

Finden eines Balles

Zum Finden des Balles müssen nun einfach alle Farbflecken mit der Markierungsfarbe gefiltert werden und der erste gültige Fund wird als Ball angenommen[23].

Finden eines Roboters

Um das Auffinden eines Roboters schneller zu gestalten wird ein Quadtree benutzt. Der Quadtree als Datenstruktur bietet eine effiziente Möglichkeit, nah zueinander liegende Farbflächen zu finden, ohne jede Farbfläche mit jeder anderen zu vergleichen. Hierfür teilt der Quadtree das Bild in vier Teile und wiederholt das für die neuen Teilbilder, bis die Anzahl der Farbflächen in einem Teilbild einen bestimmten Wert unterschreitet. Dadurch lässt sich die Nachbarschaft von Farbflächen durch die Hierarchie der Teilbilder leicht herausfinden.

Als Vorbereitung zum Auffinden werden alle verbleibenden Farbflächen in einen Quadtree (4.3.3) eingefügt. Darauf werden die Farbflecken aus dem Quadtree gesucht, die sich in der Nähe eines Markierungsflecks finden.

Falls mehr als drei Flächen gefunden werden, werden diese so sortiert, dass die Farbflächen, die am besten passen, weiterverwendet werden. Am besten passend sind hierbei die Farbflächen, die am meisten Fläche aufweisen und am nächsten an der Markierungsfläche sind.

Programmcode 4.1: Wertungskriterium für gefundene Farbflächen

```
areavalue = pointsize - 2 * distance to marker-point;
```

Dann werden die gefundenen Flächen im Uhrzeigersinn um ihren Mittelpunkt sortiert.

Als Bezeichner für den gefundenen Roboter werden die Anfangsbuchstaben der Farbgruppen von den sortierten Farbflecken verwendet. Für **G**rün **R**ot **B**lau **R**ot steht zum Beispiel "grbr". Der Roboter in Abbildung 4.5.2 ist zum Beispiel im System mit "rgpt" bezeichnet. Dadurch ist es leicht möglich, einen neuen Roboter hinzuzufügen, ohne etwas konfigurieren zu müssen. Der Roboter muss lediglich seine Bezeichnung kennen.

Daraufhin werden die gefundenen Punkte im Uhrzeigersinn um ihren Mittelpunkt sortiert und als Ergebnis weitergeben.

[23] Bei zwei Bällen auf dem Spielfeld führt dies dazu, dass immer ein beliebiger Ball gefunden wird. Da aber bei richtigen Spielen nur mit einem Ball gespielt wurde und dieser auch zuverlässig erkannt wurde, ist dieses Verhalten akzeptabel.

Abbildung 4.8: Ein mit den Farben Rot, Grün, Pink und Türkis markierter Roboter

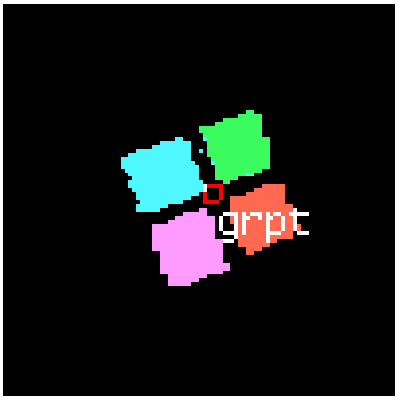

Abbildung 4.9: Das Ergebnisbild zu Abbildung 4.5.2

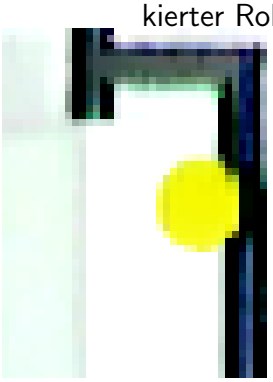

Abbildung 4.10: Ein durch die Farbe Gelb markierter Ball

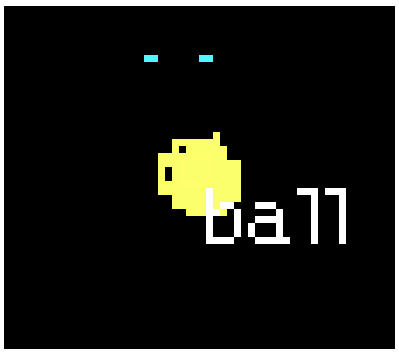

Abbildung 4.11: Das Ergebnisbild mit dem Ball zu Abbildung 4.5.2

4.5.3 Ausgabefilter

Ballpositionsmerker

Diese erst spät aufgekommene Anforderung wird realisiert, indem sich ein "Ausgabefilter" die letzte Ballposition merkt. Wird in einer beliebigen zu filternden Ergebnisliste kein Ball gefunden, wird die letzte bekannte Ballposition als aktuelle Ballposition angegeben. Damit die Roboter dies auch berücksichtigen können, wird gezählt, wie viele Bilder in Folge kein Ball gefunden wurde, und dem Roboter mitgeteilt.

Objektverfolgung

Eigentlich war es aufgrund der Verzögerung bei anderen Systemen geplant die Positionen der Objekte vorauszuberechnen. Aufgrund der Qualität der ausgewählten Algorithmen hat hat es sich jedoch aus unnötig erwiesen.

Da aber noch eine Verzögerung von etwa 22 Millisekunden vorliegt, ließe sich mit der folgenden Vorgehensweise eine noch genauere Abbildung der tatsächlichen Positionen erreichen.

Da zu verfolgende Objekte sich durch die Verzögerung bei der Bildaufnahme, der Bildübertragung und der Bildverarbeitung schon weiterbewegt haben können, kann die Objektverfolgung, dies mit einfachen Mitteln auszugleichen versuchen. Hierfür berechnet die Objektverfolgung die vermutlich aktuellen Positionen der zu verfolgenden Objekte.
Dies tut sie, indem sie sich die jeweils letzte Position der Objekte merkt und anhand ihrer neuen Position und der Verzögerung durch die Aufnahme, die Verarbeitung und die Weiterleitung die vermutliche Position berechnet.
Die Verzögerung, die zu berücksichtigen ist, kann aus folgender Tabelle entnommen werden:

Aufnahmeverzögerung	3 ms
Übertragungsverzögerung von der Kamera zu dem Computer	12 ms
Verarbeitungsverzögerung	5 ms
Übertragungsverzögerung via IEEE 802.15.4	2 ms
Gesamtverzögerung	22 ms

Die Verzögerungen für die Aufnahme ist dem Technischen Handbuch zu der Sony DFW-V500 und DFW-VL500 Kamera (Sony Corporation, 2001) entnommen. Für andere Kameras gelten hier andere Werte und diese müssten dem jeweiligen Handbuch entnommen werden.
Die Übertragungszeit geht davon aus, dass nur diese Kamera den Firewire-Bus benutzt und dass dieser mit 400 Mbps[24] läuft. Da ein Bild $640x480x2 = 614.400\ Byte$ groß ist [25] und der Bus 50 MByte je Sekunde transportiert, braucht er $\frac{614.400\ \text{Byte}}{50\ \text{MByte je Sekunde}} \approx 12$ ms
Die Verarbeitungszeit wurde auf der Grundlage geschätzt, dass der Verarbeitungsthread mit etwa 15% CPU-Last läuft und 30 Bilder pro Sekunde verarbeitet.
Von einer Sekunde benötigt er also 150 ms für 30 Bilder, also 5 ms je Bild.
Die Übertragungsverzögerung für IEEE 802.15.4 ist (Fischer, 2006) entnommen.
Da die Zeit zwischen den einzelnen Bildern bei $\frac{1}{30}$ Sekunde liegt, ist sie mit etwa 33 ms etwas größer als die 22 ms der Gesamtverzögerung.

[24] Mbps bedeutet **M***ega***b***it* **p***er* **s***econd*.
[25] x-Auflösung, y-Auflösung, ein Byte Helligkeit und ein Byte mit einer der beiden Farbinformationen (U oder V, siehe 2.2.4).

Somit muss einfach (alte Position − neue Position)∗ 22/33 = wahrscheinliche Position berechnet werden, um die Position zu erhalten, die das Objekt bei unveränderter Geschwindigkeit und Richtung nach der Verzögerungszeit hat. Für den Winkel muss genauso (alter Winkel − neuer Winkel)∗ 22/33 = wahrscheinlicher Winkel berechnet werden.

4.5.4 Objekt-Wiederverwendung

Die Ergebnisse und Ergebnislisten werden, wenn sie nicht mehr benötigt werden, in einem Ringpuffer zurück gespeichert, um deren Wiederverwendung zu ermöglichen.

4.5.5 Bild

Das Bild (3.4.7) ist für die Speicherung von Bilddaten und deren Konvertierung in ein gewünschtes Format zuständig.

Für die Speicherung hat es jeweils für die Repräsentationsarten YUV (2.2.4), RGB (2.2.1) und CImg (4.3.6) einen Puffer.

Das Bild kann seinen Inhalt in diese Repräsentationsarten konvertiert weitergeben. Die Puffer hierfür werden bei Bedarf träge initialisiert (*lazy initialization*((Gamma u. a., 1995) Seite 112). Sobald dem Bild neue Bilddaten zugewiesen werden, werden sämtliche konvertierten Puffer als ungültig markiert.

Über die nebenläufige Verwendung aus mehreren Threads mehr unter 3.2.

4.6 Optimierung der Anwendung

4.6.1 Profiling

Mithilfe von Profiling[26] wurde das Programm an kritischen Stellen optimiert.

An Abbildung 4.12 erkennt man zum Beispiel, dass scheinbar fast die Hälfte der 6,17%, der Ausführungszeit für die Objekterkennung, nämlich 3,07%, für Abfragen an die Konfiguration verwendet werden. Hier lassen sich die in der Konfiguration gespeicherten Variablen einfach in der Objekterkennung speichern und dadurch ist die Objekterkennung doppelt so schnell. Man muss allerdings dafür Sorge tragen, dass bei einer Änderung der Konfiguration

[26] Profiling ist eine Methode, um ein Profil über das Laufzeitverhalten eines Programms zu erstellen und zu ermitteln, wo im Programm wie viel der Ausführungszeit verbraucht wird.

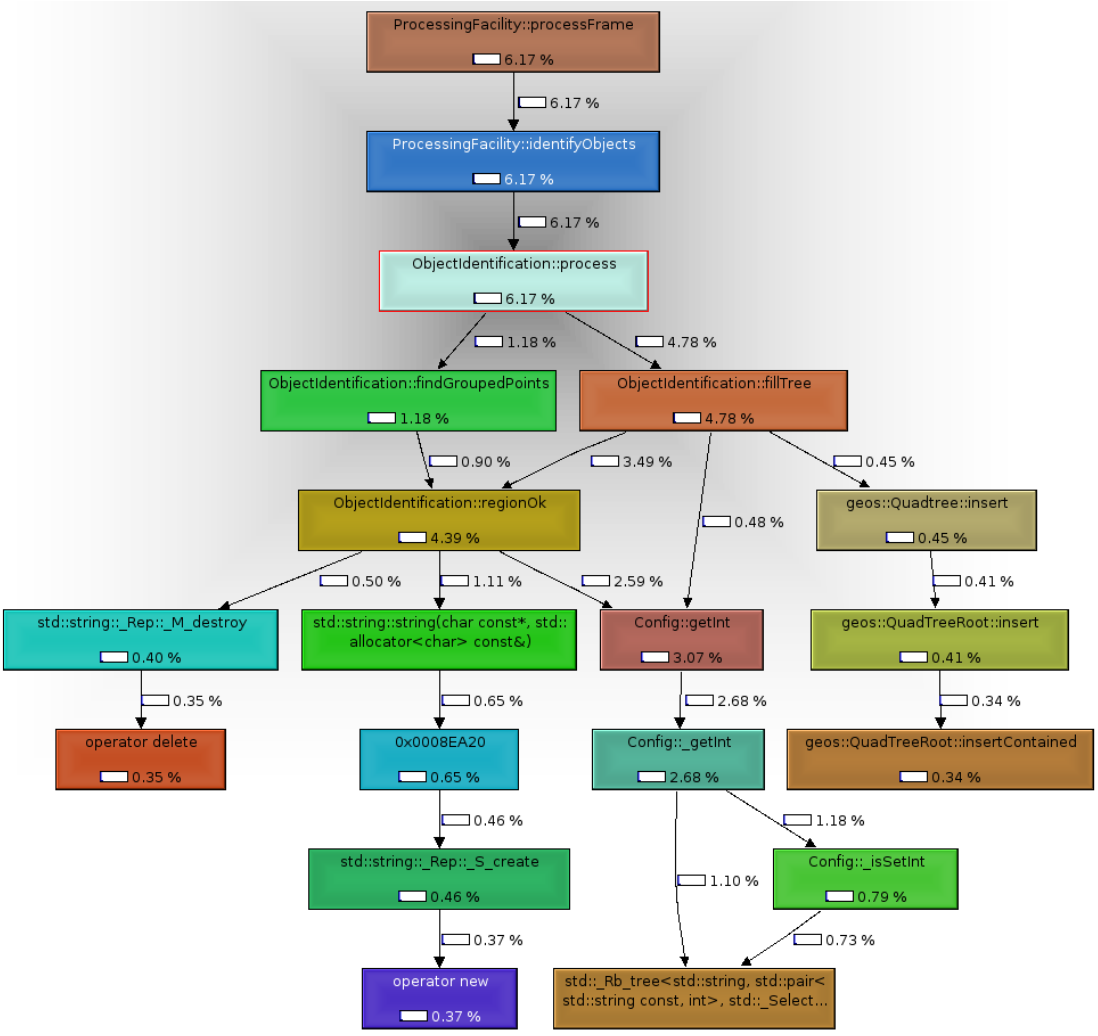

Abbildung 4.12: Ausführungszeiten der einzelnen Teile der Objekterkennung

diese Änderungen weitergegeben werden. Dies ist mit dem Observer-Pattern (Gamma u. a., 1995) leicht realisierbar.

Es hat sich auch gezeigt, dass durch ein Ersetzen der CImg-Bibliothek weitere Optimierungen möglich sind. Zum einen verbraucht die Bibliothek selbst viel CPU-Zeit zum Anzeigen der Bilder, zum anderen sind Konvertierungen nötig um das Bild anzuzeigen (Abbildung 4.13).
Es gibt auch Möglichkeiten, das Bild ohne Konvertierungen direkt anzuzueigen, jedoch sind diese von Betriebssystem zu Betriebssystem verschieden und hätten so den Wartungsaufwand erhöht. Da die Anzeige auch keinen Echtzeitanforderungen unterliegt, wurde hier auf weitere Optimierungen verzichtet.

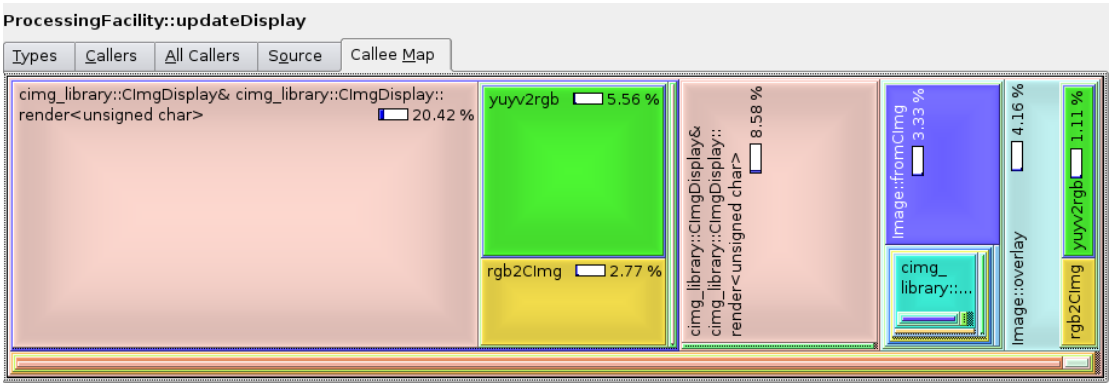

Abbildung 4.13: CPU-Auslastungsvergleich von der Anzeige (rosa) und der Konvertierung für die Anzeige (Rest)

4.6.2 Vermeidung von Speicherlecks

Um Speicherlecks zu beheben, wurde MallokDebug benutzt. MallokDebug ist Teil der XCode IDE und beobachtet Speicheranforderungen und -freigaben sowie verwendete Speicherbereiche. Dadurch ist MallokDebug in der Lage, nicht mehr verwendete Speicherbereiche zu finden sowie dem Entwickler dadurch zu helfen, dass es aufzeigt, wo dieser Speicher angefordert wurde.

In der entwickelten Objektverfolgung gab es einige Stellen, an denen der Speicher nicht wieder freigegeben wurde. Dadurch, dass im Verlauf der Entwicklung zudem noch die Nebenläufigkeit der Anzeige und der Verarbeitung eingeführt wurde und Objekte zwischen diesen beiden Verarbeitungssträngen ausgetauscht werden mussten, war es nötig, eine Verwaltung dieser Objekte einzuführen.

Um die Ergebnisse bedenkenlos zwischen den Verarbeitungssträngen auszutauschen, wurde der Ringbuffer verwendet. Dieser wurde des Weiteren auch verwendet, um die Objekte, sobald sie nicht mehr benötigt wurden, zu einem Objektpool hinzuzufügen, aus dem Anforderungen für neue Objekte befriedigt wurden.

5 Fazit

In diesem Kapitel werden zunächst die Ergebnisse anhand der Anforderungen gezeigt, dann werden ein paar der aufgetretenen Probleme erläutert und abschließend ein Ausblick auf Erweiterungsmöglichkeiten gegeben.

5.1 Ergebnisse

Die Ergebnisse beziehen sich auf die Anforderungen, welche allesamt erfüllt werden konnten.

Genauigkeit

Die Genauigkeit des Systems hängt hauptsächlich von der Auflösung der Kamera ab.

Bei der verwendeten Auflösung von 640x480 Pixeln entspricht ein Pixel einer Entfernung von etwa 2,7 mm auf dem Spielfeld. Da es an den Rändern der zu erkennenden Objekte meist einen Rand von einem Pixel gab, welcher nicht richtig erkannt wurde, liegt die Genauigkeit bei etwa 4 mm.

Die Genauigkeit des Winkels liegt bei etwa 5°.

Effizienz

Das System hat auf dem verwendeten MacBook bei einer Auflösung von 640x480 Pixeln und 30 FPS zwischen 10% und 20% einer CPU belastet. Damit liegt die Verarbeitungszeit je Bild bei etwa drei bis sieben Millisekunden.
Somit wurde die Anforderung der weichen Echtzeit erfüllt.
Da bei 30 Bildern die Sekunde alle 33 Millisekunden ein Bild vorliegt, lassen sich noch weitere Berechnungen anschließen, ohne das Echtzeitverhalten zu stören.

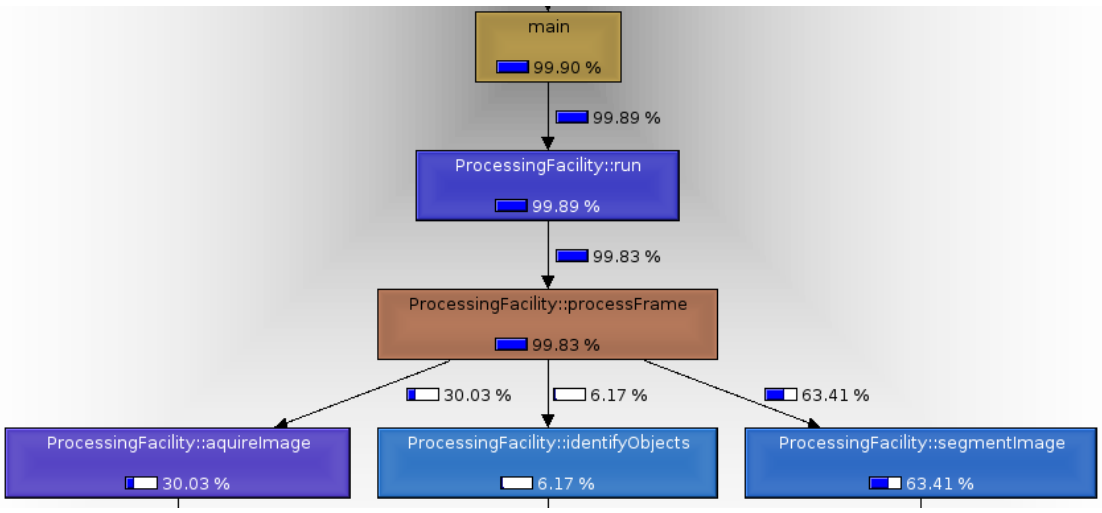

Abbildung 5.1: CPU-Zeiten bei der Verarbeitung

Benutzbarkeit

Das System war nach dem Aufbau der Kamera und des Spielfeldes innerhalb weniger Minuten benutzbar. Es musste nur die Kamera ausgerichtet werden und gegebenenfalls kleine Anpassungen an der Farbklassen vorgenommen werden (Abbildung 5.2).

Diese Benutzbarkeit wurde besonders dadurch verbessert, dass das System während der Entwicklung ständig benutzt wurde (4).

Stabilität

Das System hat die Roboter zuverlässig erkannt (Abbildung 5.3). Auch Änderungen in der Beleuchtung beeinflussen das System nicht sehr stark und gegebenenfalls lassen sich die Farbklassen schnell anpassen.

Portierbarkeit

Das System funktioniert unter Mac OS X und Linux. Um das System unter Windows einsatzfähig zu machen, wäre es nur notwendig, eine Bildquelle, zum Beispiel auf der Basis von "Video for Windows", zu programmieren.

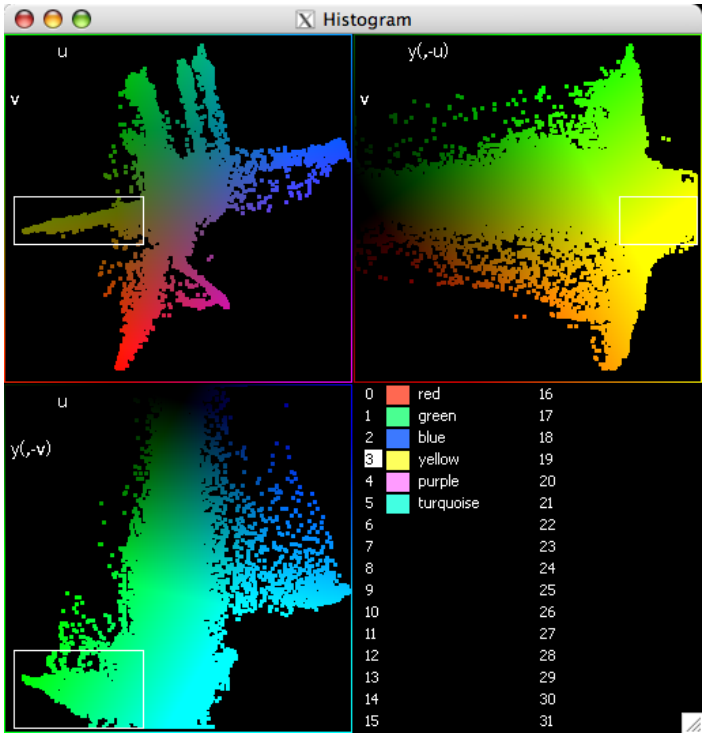

Abbildung 5.2: Anpassung der Farbklassen

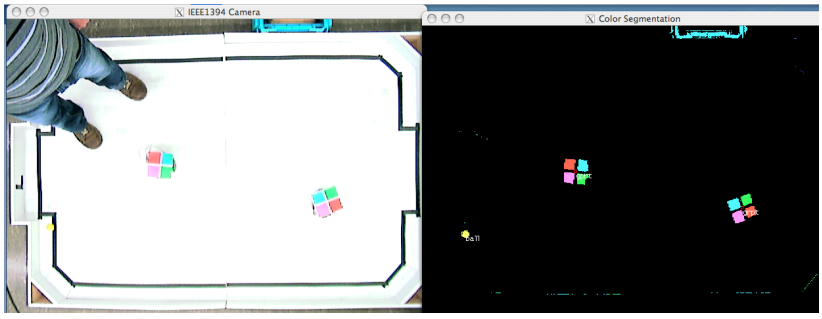

Abbildung 5.3: Die Farberkennung lässt sich nicht so leicht stören.

5.2 Aufgetretene Probleme und ihre Lösungen

Langsame Darstellung

Die Anforderung der weichen Echtzeit gilt nur für die Weitergabe der Informationen an die Roboter.
Da aber die Darstellung auf dem Bildschirm deutlich mehr CPU-Zeit brauchte als die Verarbeitung, stellte sich im Verlauf der Entwicklung und des Testens heraus, dass es notwendig ist, die Darstellung von der Verarbeitung zu trennen. Deshalb wurde die Darstellung in einen separaten Verarbeitungsstrang gelegt (Siehe hierzu auch 4.6).
Dies bringt auch den weiteren Vorteil, dass die Entwicklungs- und Testcomputer besser ausgelastet sind, da hierfür Doppelkern-CPUs verwendet wurden.

Durch die parallele Verarbeitung kam es zu einer unterschiedlichen Auslastung beider CPU-Kerne. Der Kern, der mit der Verarbeitung beschäftigt war, arbeitete mit der Kamerageschwindigkeit von 30 FPS[1] bei einer Auslastung zwischen 10 und 15%. Der Kern, der die Anzeige übernommen hatte, war mit 15 FPS voll ausgelastet. Bei der Anzeige sind definitiv noch weitere Optimierungen möglich[2], die jedoch nicht umgesetzt wurden, da sie nur der Kontrolle des Bildes und der Farbanzeige dienen.

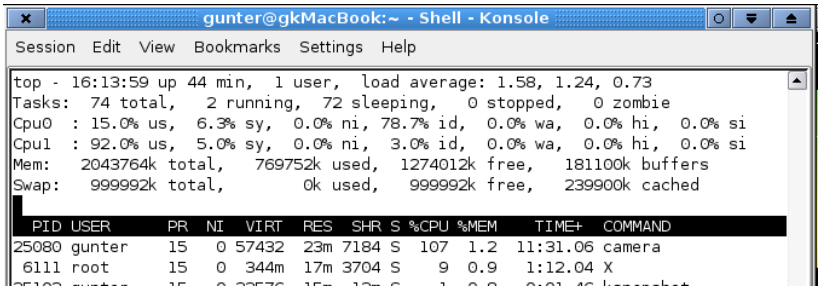

Abbildung 5.4: Auslastung der CPU-Kerne von 15% und 92% bei laufendem Programm

Portierbarkeit

Die Anforderung, Portierbarkeit (2.1) zu erreichen, verursachte folgende Probleme:
Die Bibliothek Libdc1394 (4.3.6) unterstützt in Version 2.0 auch Mac OS X. Da sich die

[1] **F***rames* **p***er* **s***econd*, die Anzahl der Bilder, die die Kamera je Sekunde liefert.
[2] Zum Beispiel ließe sich die Anzeige mit einer Bibliothek realisieren, welche es ermöglicht, ein Bild in dem YUV-Format direkt anzeigen zu lassen.

Fertigstellung der Bibliothek seitens des Herstellers bisher um ca. drei Monate verzögerte und während dieser Phase mehrere Änderungen an den Schnittstellen vorgenommen wurden, entstand zusätzlicher Aufwand für entsprechende Anpassungen.

Die Bibliothek CImg verursachte ebenfalls Probleme. Da CImg die Daten intern auf andere Weise speichert (4.3.6) als sie aus anderen Bibliotheken vorliegen, wurden Konvertierungen zwischen den Formaten notwendig. Weiterhin benötigt die Bibliothek relativ zur Konvertierung viel Zeit zum Anzeigen der Bilder (Abbildung 4.13).

5.3 Ausblick

Es gibt noch viele Bereiche, in denen die Objektverfolgung ausbaubar ist.

So könnte:

- das Histogramm nur für ausgewählte Bildbereiche dargestellt werden, sodass man beispielsweise auf dem Kamerabild nur den Ball auswählt, damit man die Grenzen des Farbtons leichter bestimmen kann.
- nicht nur ein minimaler und maximaler Grenzwert eines Farbtons bestimmt werden, sondern dieser durch eine Fläche zwischen der U- und der V-Achse (bei dem YUV-Farbmodell (2.2.4)) angegeben werden. Es kann auch direkt ein Farbvolumen angegeben werden, wodurch dann ebenfalls das RGB-Farbmodell (2.2.1) als Basis für die Farbtonauswahl herangezogen werden kann.
- die Koordinaten relativ zur Spielfläche ausgegeben werden und nicht relativ zum Kamerabild[3]. Dies bietet die weitere Möglichkeit, die Kamera so einzustellen, dass diese nur den gewünschten Bereich aufnimmt (*regions of interrest*), wodurch eine schnellere Reaktionszeit erreichbar ist[4].
- man die Grenzwerte für die Farberkennung automatisch an die vorherrschenden Lichtverhältnisse anpassen[5].
- die Farbsegmentierung durch *Region Growing* (Gonzales u. Woods, 2002) ersetzt werden und auch nur die Regionen, in denen der Roboter erwartet wird, von der

[3] Dies ist in dieser Arbeit nicht getan worden, da die Benutzer des Systems die Koordinaten relativ zum Kamerabild von Anfang an benutzten und eine Umstellung hier zu Problemen geführt hätte.

[4] Wenn auch nur eine minimal schnellere, da wir ohnehin fast das ganze Bild brauchen. Wir könnten bei unserer Konfiguration je 10% weniger Bildfläche, welche übertragen werden müssten, mindestens 1 ms sparen(4.5.3).

[5] Der automatische Weissabgleich führt zu einer Verbesserung der Farberkennung, aber es ließe sich sicherlich noch mehr erreichen.

Kamera abgefragt werden. Eine Implementation von *Region Growing* wird in (von Hundelshausen, 2005) beschrieben.

- das System um einen Schiedsrichter erweitert werden, der die Tore automatisch zählt und die Roboter über den Start und das Ende des Spiels informiert, sodass ein automatisches Spiel möglich wird.
- die Bibliothek zur Bildanzeige durch eine performantere ersetzt werden.
- die Bildquelle auch mit "Video for Windows" realisiert werden, wodurch das Programm auch unter Windows benutzbar wäre.
- eine verbesserte Konfiguration der Farbwerte programmiert werden, die zum Beispiel auch das Abspeichern von Farbprofilen für verschiedene Kameras ermöglicht. Bisher muss dies direkt in der Konfigurationsdatei geändert werden.

5.4 Resümee

Die Anforderungen an die Anwendung wurden alle in vollem Umfang erfüllt. Die Software ist gut erweiterbar und bietet der Hochschule für Angewandte Wissenschaften Hamburg eine Grundlage für weitere Entwicklungen. Das System wurde bereits während eines Projektes mit einem Abschlusswettbewerb getestet und hat sich als zuverlässig erwiesen.

Literaturverzeichnis

[Alvarado u. a. 2007] Alvarado, Pablo ; Doerfler, Peter ; Canzler, Ulrich: *LTI-Lib*. http://ltilib.sourceforge.net/. Version: 2007, Abruf: 18.03.2007

[Balzerowski 2002] Balzerowski, Rainer: *Realisierung eines Webcam basierten Kamera Systems für mobile Roboter*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2002. http://users.informatik.haw-hamburg.de/~lego/Projekte/Balzerowski/diplomarbeit-www.pdf, Abruf: 18.03.2007

[Beck 2000] Beck, Kent: *extreme Programming explained*. Addison-Wesley, 2000. – ISBN 0–201–61641–6

[Brooks 1995] Brooks, Frederick P.: *The Mythical Man-Month*. Addison-Wesley, 1995. – 197–199 S. – ISBN 0–201–83595–9

[Bruce u. a. 2000] Bruce, James ; Balch, Tucker ; Veloso, Manuela: *Fast and Inexpensive Color Image Segmentation for Interactive Robots*, Carnegie Mellon University, Paper, 2000. http://www.cs.cmu.edu/~jbruce/cmvision/papers/CMVision-IROS2000.pdf, Abruf: 18.03.2007

[Costa u. a. 2004] Costa, Paulo ; Moreira, Antonio ; Marques, Paulo ; Sousa, Armando ; Costa, Pedro ; Gaio, Susana: *5dpo Robotic Soccer Team for Year 2004*, Universidade do Porto, Team Description, 2004. http://iml.cpe.ku.ac.th/skuba/archive/SkubaTDP2007.pdf, Abruf: 18.03.2007

[Dieckmann 2003] Dieckmann, Arne: *Verbesserung visueller Objekterkennung für mobile Roboter*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2003. http://users.informatik.haw-hamburg.de/~kvl/dieckmann/diplom.pdf, Abruf: 18.03.2007

[Fischer 2006] Fischer, Christian: *Entwicklung von ZigBee-Modulen für spontane Funknetzwerke*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2006. http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/fischer.pdf, Abruf: 18.03.2007

[Fogel 2005] Fogel, Karl: *Producing Open Source Software*. OReilly, 2005. – ISBN 0–596–00759–0

[Free Software Foundation 2007a] Free Software Foundation: *GPL Lizenz*. http://www.gnu.de/documents/gpl.de.html. Version: 2007, Abruf: 18.03.2007

[Free Software Foundation 2007b] Free Software Foundation: *LGPL Lizenz*. http://www.gnu.de/documents/lgpl.de.html. Version: 2007, Abruf: 18.03.2007

[Gamma u. a. 1995] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns*. Addison Wesley, 1995. – ISBN 0–201–63361–2

[GIMP-Team 2007] GIMP-Team: *Gimp*. http://www.gimp.org/. Version: 2007, Abruf: 18.03.2007

[Gonzales u. Woods 2002] Gonzales, Rafael C. ; Woods, Richard E.: *Digital Image Processing*. Prentice-Hall, 2002. – 612–617 S. – ISBN 0–201–18075–8

[Gottwald 2005] Gottwald, Michael: *Webcam basiertes Kamerasystem für autonome Roboter : Erste Konzeption*, Hochschule für Angewandte Wissenschaften Hamburg, Studienarbeit, 2005. http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/studien/gottwald.pdf, Abruf: 18.03.2007

[Heise Zeitschriften Verlag 2006] Heise Zeitschriften Verlag: *CT-Bots*. http://www.heise.de/ct/ftp/projekte/ct-bot/. Version: 2006, Abruf: 18.03.2007

[von Hundelshausen 2005] Hundelshausen, Dr. F.: Ortung eines Roboters durch Verfolgen homogener Farbregionen. In: *it-InformationTechnology* (2005), April, Nr. 47(5), 258–265. http://page.mi.fu-berlin.de/~hundelsh/publications/Y2005_RobotLocalizationThroughRegionTracking/RobotLocalization.pdf

[Jack 1993] Jack, Keith: *Video Demystified*. LIh Technology Pub, 1993. – ISBN 1–878707–09–4

[Kohonen 2001] Kohonen, Teuvo: *Self-Organizing Maps*. Springer, 2001. – ISBN 3–540–67921–9

[lizenz:cecill 2007] *CeCILL-C Lizenz*. http://www.cecill.info/licences/Licence_CeCILL-C_V1-en.txt. Version: 2007, Abruf: 18.03.2007

[Massachusetts Institute of Technology] Massachusetts Institute of Technology: *MIT Lizenz*. http://www.opensource.org/licenses/mit-license.php, Abruf: 18.03.2007

[Meisel 2006] Meisel, Andreas: *Merkmalsextraktion*. https://www.informatik.haw-hamburg.de/cms/uploads/media/RV07_01.pdf. Version: 2006, Abruf: 18.03.2007

[Miglino u. a. 1995] Miglino, Orazio ; Lund, Henrik H. ; Nolfi, Stefano: Evolving Mobile Robots in Simulated and Real Environments. In: *Artificial Life* 2 (1995), Nr. 4, S. 417–434

[Oesterreich 2006] Oesterreich, Bernd: *Analyse und Design mit UML 2.1*. Oldenbourg Wissenschaftsverlag GmbH, 2006. – ISBN 3–486–57926–6

[Pierce 2005] Pierce, Eric: *HSV Cone*. http://commons.wikimedia.org/wiki/Image:HSV_cone.jpg. Version: 2005, Abruf: 18.03.2007

[Revout 2003] Revout, Ilia: *Design und Realisierung eines Frameworks für Bildverarbeitung*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2003. http://users.informatik.haw-hamburg.de/~kvl/revout/diplomarbeit.pdf, Abruf: 18.03.2007

[Rickens 2005] Rickens, Helge: *Ein Zigbee-Funkmodul zur Kommunikation unter autonomen Robotern*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2005. http://users.informatik.haw-hamburg.de/~kvl/rickens/Diplomarbeit_Rickens.pdf, Abruf: 18.03.2007

[Robocup] *Robocup*. http://www.robocup.de, Abruf: 18.03.2007

[Schmidt 2005] Schmidt, Oliver: *Entwicklung und Aufbau eines Kamera-Service basiert auf ".net"-Technologie*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2005. http://users.informatik.haw-hamburg.de/~kvl/schmidt/diplom.pdf, Abruf: 18.03.2007

[Sony Corporation 2001] Sony Corporation: *Technical Manual for Sony DFW-V500 and DFW-VL500*. 2001

[Srisabye u. a. 2006] Srisabye, Jirat ; Parkpien, Napat ; Kongniratsiakul, Poom ; Hoonsuwan, Phachachon ; Bowarnkitiwong, Saran ; Archawananthakul, Marut ; Dumnernkittikul, Ratchai ; Chongkaonar, Santi ; Ratanaparadorn, Anuchit ; Keawpromman, Chayaporn ; Limnirunkul, Varut ; Tipsuwan, Yodyium: *Skuba 2007 Team Description*, Kasetsart University, Team Description, 2006. http://iml.cpe.ku.ac.th/skuba/archive/SkubaTDP2007.pdf, Abruf: 18.03.2007

[Tannenbaum 2003] Tannenbaum, Andrew S.: *Moderne Betriebssysteme*. Pearson Studium, 2003. – ISBN 3–8273–7019–1

[Tschumperle 2007] Tschumperle, David: *CImg*. http://cimg.sourceforge.net/. Version: 2007, Abruf: 18.03.2007

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, 21. März 2007
Ort, Datum

Unterschrift