Entwicklung eines Farbkalibrierungswerkzeugs unter Verwendung von Kameraparametern

UNIKASSEL
VERSITÄT

BACHELORARBEIT

dem Fachbereich Elektrotechnik/Informatik
der Universität Kassel

vorgelegt von

Philipp Sperber

Matrikelnummer: 29211090

aus

Kassel

Kassel, Oktober 2013

Erstgutachter:	Prof. Dr. Kurt Geihs
Zweitgutachter:	Prof. Dr. Gerd Stumme
Betreuer:	Andreas Witsch, M.Sc.

Plagiatserklärung

Hiermit erkläre ich, dass ich meine Bachelorarbeit mit dem Thema:

Entwicklung eines Farbkalibrierungswerkzeugs unter Verwendung von Kameraparametern

selbständig verfasst sowie alle wesentlichen Quellen und Hilfsmittel angegeben habe.

Kassel, den 16.10.2013

Philipp Sperber

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Kurzfassung

Viele autonome mobile Roboter verwenden Kameras, um Objekte in ihrem Umfeld wahrzunehmen. Die Analyse basiert häufig auf der farblichen Segmentierung der Kamerabilder. Grundlage dafür bilden korrekt kalibrierte Kameraparameter sowie die vollständige Identifikation der Farben des Zielobjektes.

Im Rahmen der vorliegenden Arbeit wurde ein Softwarewerkzeug entwickelt, welches die Identifikation gesuchter Farbwerte und die Anpassung der Kameraparameter eines Roboters zur Laufzeit des Bildverarbeitungsprozesses erlaubt. Um die Kalibrierung zu unterstützen, wird mit Hilfe von Erwartungswertmaximierung die Menge der Farbwerte vervollständigt. Zur Evaluation wurden Kamerabilder auf einem Roboterfußballfeld aufgezeichnet und mit einem methodischen Vorgehen die Farbwerte eines Fußballs extrahiert. Dabei hat sich gezeigt, dass die Anzahl der falsch erkannten Farbwerte geringer als bei der Vergleichsmethode ist.

Kapitel 2

Einleitung

In einer Zeit, in der voll automatisierte Roboter in der Lage sind, eigenständig komplexe Aufgaben zu erledigen, spielt die Wahrnehmung ihrer Umgebung für eine wichtige Rolle. Die Roboter erfassen ihr Umfeld mit Hilfe von Kameras und müssen die aufgenommenen Bilder selbstständig und zuverlässig verarbeiten. Zu diesem Zweck existieren bereits diverse Automatismen, beispielsweise Verfahren zur Objekterkennung.

Das Fachgebiet *Verteile Systeme* der Universität Kassel erforscht diesen Bereich der Informatik im Umfeld des *RoboCup Wettbewerbs* [1]. Für diesen werden Roboter entwickelt, die eigenständig Fußball spielen können. Unter dem Namen *Carpe Noctem* [2] nimmt die Universität Kassel jährlich in der *Middle Size*-Klasse teil. Bei diese Middle Size Variante haben die Roboter eine maximale Seitenlänge von 50 cm und spielen in Teams von bis zu fünf Robotern mit einem Fifa Fußball. Die Bildverarbeitung dient dabei dem Auffinden des Balles, dem Erkennen der Gegner beziehungsweise der Mitspieler und bildet somit die Grundlage zur Steuerung des Roboters. Die Ballerkennung findet anhand von Farbwerten in einem aufgenommenen Bild statt. Eine entsprechende Umsetzungstabelle enthält für alle Farbwerte die Information, ob diese Relevanz haben. Damit bildet diese die Grundlage für die farbliche Segmentierung der Bilder. Vor jedem RoboCup-Spiel muss die Umsetzungstabelle an den aktuell verwendeten Ball angepasst werden. Die Spiele sollen in Zukunft unter freiem Himmel stattfinden; somit ist es möglich, dass sich die Lichtverhältnisse zwischen zwei Halbzeiten, zum Beispiel durch Wolken, verändern. Die Kalibrierung der Umsetzungstabelle muss demnach einfach und detailliert möglich sein.

Die Motivation der vorliegenden Arbeit liegt in der Entwicklung eines Softwarewerkzeugs, welches die Kalibrierung der Umsetzungstabelle ermöglicht und mit dessen Hilfe die Kameraparameter der Roboter zur Laufzeit manipuliert werden können. Da, wie bereits erwähnt, die korrekte Kameraeinstellung für eine fehlerfreie Bilderkennung unablässig ist, soll das Werkzeug nicht nur aktuelle Kamerabilder von den

Roboter übertragen, sondern auch die Kameraeinstellungen verändern können. Die aktuellen Kamerabilder sollen darüber hinaus für die Kalibrierung der Umsetzungstabelle verwendet werden.

Die vorliegende Arbeit beschäftigt sich mit der Realisierung der beschriebenen geforderten Funktionalitäten. Hierfür musste in erster Linie eine Möglichkeit geschaffen werden, die Umsetzungstabelle in einer Datei speichern zu können. Die entsprechenden Farbwerte können somit zum einen direkt in die Umsetzungstabelle eingetragen, zum anderen aber auch indirekt, aus den aktuell Bildern abgeleitet und dort übernommen werden.

Der entsprechende Aufbau der Arbeit gestaltet sich folgendermaßen: In Kapitel 3 werden die Probleme der Vorgehensweise bei der Bilderkennung aufgezeigt und die daraus resultierende Aufgabenstellung abgeleitet. Kapitel 4 beinhaltet Grundlagen, wie die relevanten Farbräume und das aktuelle Bilderkennungsverfahren. In Kapitel 5 wird das entstandene Werkzeug vorgestellt und die entsprechenden Implementierungsdetails beleuchtet. Eine Bewertung der Arbeit findet anschließend in Kapitel 6 statt. Im Anschluss werden in Kapitel 7 einige verwandte Arbeiten aufgeführt. Abschließend weist Kapitel 8 eine Zusammenfassung und einen Ausblick auf.

Kapitel 3

Problemstellung

Die Bilderkennung der aufgezeichneten Bilder ist eine fundamentale Softwarekomponente des Roboterframeworks. Diese bietet sowohl die Grundlage für die Ball- und Hinderniserkennung als auch für die Positionsbestimmung des eigenen Roboters. Eine fehlerfreie Bildverarbeitung ist essenziell, um eine erfolgreiche Reaktion auf die Umgebung zu garantieren. In diesem Kapitel werden die Funktionsweise des Bilderkennungsverfahrens und die daraus resultierende Aufgabenstellung beschrieben.

3.1. Bilderkennung

Bei der Bildverarbeitung werden anhand der Farbwerte die Bilder nach „Region of Interest“[1] (ROI) durchsucht. Die Informationen, welche Farben eine ROI identifizieren, werden üblicherweise in einer Umsetzungstabelle hinterlegt. Eine detaillierte und fehlerfreie Umsetzungstabelle bildet somit die Grundlage für eine einwandfreie Bilderkennung.

In Listing A.1 ist zu sehen, dass die Initialisierung dieser Umsetzungstabelle bisher direkt in dem Programmcode des Roboterframeworks stattfindet. Diese Vorgehensweise hat einige wesentliche Nachteile. Die Kalibrierung der Umsetzungstabelle erfordert einen hohen Zeitaufwand, weil das Programm nach jeder Anpassung erneut kompiliert werden muss. Im Anschluss muss der Bildverarbeitungsprozess auf dem Testroboter komplett neu gestartet werden. Ein zusätzliches Problem stellt die Tatsache dar, dass die Einarbeitung in den Initialisierungsprozess ein langwieriger Vorgang ist. Das hat zur Folge, dass in der Regel nur wenige Personen in der Lage sind, die Kalibrierung vorzunehmen. Der hohe Zeitaufwand und die Komplexität führen zu mangelnder Dynamik. Eine umfangreiche Änderung an der Umsetzungstabelle im Laufe

[1] Englisch für „interessante Flächen“

eines Turniers ist auf Grund des hohen zeitlichen Aufwands schwierig. Das macht eine spontane Reaktion auf die Umwelt, zum Beispiel auf wechselnde Lichtverhältnisse, nahezu unmöglich.

Ein weiteres Problem ist, dass die Möglichkeiten bei der Generierung der Umsetzungstabelle für eine optimale Parametrisierung nicht ausreicht. Dies hat zur Folge, dass die Umsetzungstabelle nicht bestmöglich kalibriert werden und somit ein erhöhtes Rauschen auftreten kann. Als Rauschen bezeichnet man fälschlicherweise klassifizierte Punkte. Das kann zwei Gründe haben, zum einen, dass die Umsetzungstabelle die falschen Farbwerte enthält oder zum anderen, dass die Ballfarbe auch an anderen Stellen des Bildes auftritt als an der Position des Balls. Die erste Variante des Rauschen ist mit einer besser kalibrierten Umsetzungstabelle zu vermeiden. Die zweite Art wird bei dem *Carpe Noctem*-Framework in einem späteren Verlauf des Ballerkennungsverfahrens durch Abgleichen von Formen eliminiert und ist somit nicht Gegenstand dieser Arbeit.

Vor Beginn dieser Arbeit war es außerdem noch nicht möglich, aktuelle Bilder der Roboterkameras zur Laufzeit zu laden, um diese dauerhaft zu speichern. Diese Funktionalität ist einerseits nötig, um eine Kalibrierung zu einem späteren Zeitpunkt ohne aktiven Roboter zu ermöglichen, andererseits um eine Analyse eines fehlerhaften Verhaltens des Roboters zu ermöglichen.

3.2. Aufgabenstellung

Die Aufgabe dieser vorliegenden Arbeit ist die Entwicklung eines Kalibrierungswerkzeugs für die Anpassung einer Umsetzungstabelle. Das Werkzeug soll dem Benutzer eine einfache Möglichkeit bieten, die Umsetzungstabelle schnell, komfortabel anzupassen und testen zu können. Folgende Funktionalitäten sollen bereitgestellt werden:

Anzeige von Kamerabildern Die Kamerabilder liegen in einem YUV422 kodierten Format vor. Werden die Bilder in einer Datei gespeichert, so werden die Werte mit Leerzeichen getrennt in die Datei geschrieben. Das hat zur Folge, dass man diese mit keinem herkömmlichen Bildanzeigeprogramm öffnen kann. Dies soll jedoch mit dem Werkzeug ermöglicht werden. Sowohl lokal gespeicherte Bilder als auch Momentaufnahmen der Roboterkameras sollen darstellt werden können.

Bildübertragung Bei der Kalibrierung der Umsetzungstabelle ist die Arbeit mit aktuellen durch die Roboter erfassten Bilder am effektivsten. Aus diesem Grund muss

eine Möglichkeit geschaffen werden, aktuelle Bilder von den im selben Netzwerk befindlichen Robotern übertragen zu können.

Umsetzungstabelle bearbeiten Die Umsetzungstabelle kann gezielt verändert werden, dazu stehen zwei Möglichkeiten zur Verfügung:

- Die Farbwerte direkt in der Umsetzungstabelle bearbeiten.
- Die Farbwerte aus den geladenen Bildern zu der Umsetzungstabelle hinzufügen.

Nach dem Hinzufügen von Farbwerten aus den aktuellen Bildern sind in der Regel nicht alle relevanten Ballfarben erfasst. In der Umsetzungstabelle treten somit Lücken auf. Diese Lücken müssen mit einem automatisierten Verfahren aufgefüllt werden können.

Kameraparameter modifizieren Die Änderungen an den Kameraparametern werden bisher nur beim Start des Bilderkennungsprozesses übernommen. Das zieht den Kamerakalibrierungsprozess zeitlich erheblich in die Länge. Aus diesem Grund soll mit dem Werkzeug eine Manipulierung der Kameraparameter über das Netzwerk ermöglicht werden.

Kapitel 4

Grundlagen

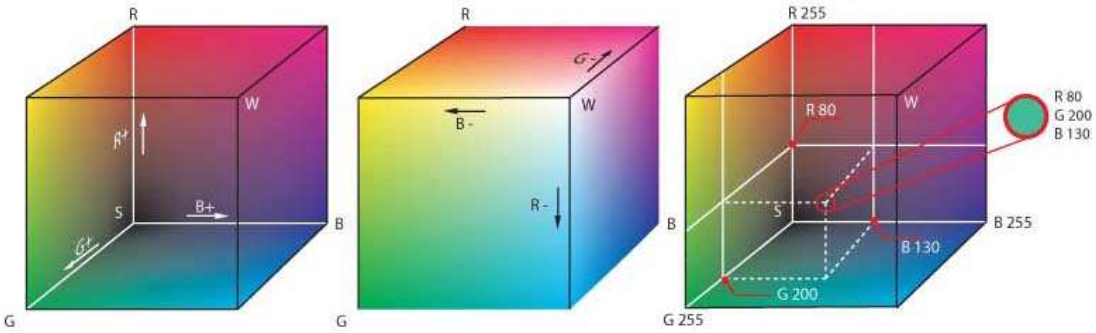
In diesem Kapitel werden die für die vorliegende Arbeit benötigten Grundlagen erläutert. Diese beinhalten die relevanten Farbräume, die Clusteranalyse mit der Erwartungswertmaximierung und das Roboter Operation System. Zusätzlich wird das Bilderkennungsverfahren des *Carpe Noctem*-Frameworks beschrieben.

4.1. Farbräume

Ein Farbraum ist die Menge aller darstellbaren Farben. In einem Farbraum werden alle möglichen Farben durch eindeutige Parameter definiert. Die Farbinformationen werden nicht nur über die Farbe, zum Beispiel Grün, sondern auch über die Helligkeit und den Kontrast dargestellt.

4.1.1. RGB

Der RGB-Farbraum stellt die Farben dar, wie sie in der additiven Farbmischung entstehen. Bei der additiven Farbmischung werden alle Farben durch das Mischen der drei Grundfarben Rot, Grün und Blau erzeugt. Gelb lässt sich beispielsweise durch die Kombination von Rot und Grün darstellen. Häufig werden die Farbanteile in Werten von 0 bis 255 angegeben; diese Darstellungsart nennt sich *RGB888*, da jeder Anteil 8 Bit benötigt. Abbildung 4.1 zeigt, dass sich der RGB-Farbraum als ein Quadrat in einem dreidimensionalen Koordinatensystem darstellen lässt. Auf den Koordinatenachsen werden dabei die Farbanteile von Rot, Grün und Blau aufgetragen.



Quelle: `http://commons.wikimedia.org/wiki/File:RGB_farbwuerfel.jpg` (Abgerufen am: 04.09.2013)

Abbildung 4.1.: Der RGB-Farbraum lässt sich als Quadrat in einem dreidimensionalen Koordinatensystem darstellen.

4.1.2. YUV

Bei dem YUV-Farbraum werden die Farb- und Helligkeitsinformationen voneinander getrennt. Der Y-Wert gibt die Helligkeit (Luminanz) und die U- und V-Werte die Farbwerte (Chrominanz) an. Auf Grund der Informationstrennung ist der YUV-Farbraum für eine farbliche Segmentierung geeignet. In Abbildung 4.2 ist zu sehen, dass sich die gleiche Farben bei unterschiedlichen Luminanzwerten an der selben Stelle befindet. Das ermöglicht eine Farbanalyse, unabhängig von der Helligkeit.

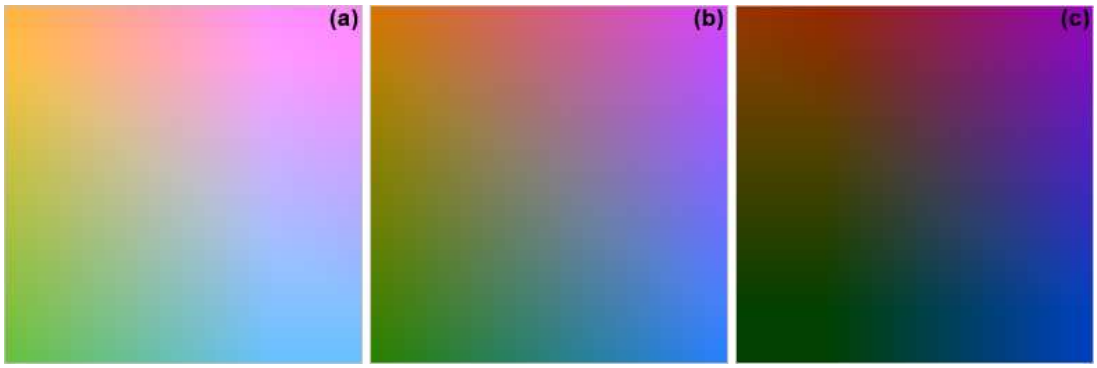


Abbildung 4.2.: Der YUV-Farbraum für verschiedene Luminanzen, *(a)* Y=64, *(b)* Y=128 und *(c)* Y=192.

4.1.2.1. YUV422

YUV422 ist eine Kodierungsart des YUV-Farbraums. Dabei wird ein Bild nicht mit allen Informationen, also pro Punkt einen Y-, U- und V-Wert, gespeichert. Stattdessen werden einige Informationen ausgelassen, weil diese durch das menschliche Auge ohne Qualitätsverlust aufgefüllt werden können. Diese Vorgehensweise nennt sich Farbunterabtastung [3]. Bei einem YUV422 kodierten Bild werden die Luminanzwerte für

jeden Punkt gespeichert, die Chrominanzwerte hingegen nur für jeden zweiten,

$$U_0Y_0V_0Y_1U_2Y_2V_2Y_3U_4Y_4V_4Y_5\cdots. \tag{4.1}$$

Für die Rekonstruktion in ein vollständiges YUV-Bild werden die Chrominanzwerte doppelt verwendet. Ein Beispiel für das in Formel (4.1) gezeigte Bild ist in Tabelle 4.1 zu sehen [4].

Pixelnummer	Pixelwerte
0	$U_0Y_0V_0$
1	$U_0Y_1V_0$
2	$U_2Y_2V_2$
3	$U_2Y_3V_2$
4	$U_4Y_4V_4$
...	...

Tabelle 4.1.: Die Rekonstruktion von YUV aus YUV422

4.2. Ballerkennungsverfahren

Die Ballerkennung spielt bei dem RoboCup eine elementare Rolle. Die Position des Balles ist eine der grundlegenden Informationen, welche die Roboter verwenden, um Entscheidungen über ihre Verhaltensweisen zu treffen. In diesem Abschnitt wird das Ballerkennungsverfahren des *Carpe Noctem*-Frameworks erläutert. Dabei konzentrieren sich die Erklärungen auf die für diese Arbeit relevanten Teilbereiche.

Die Roboter verwenden zwei unterschiedliche Kameratypen, das Modell *Flea 2 FL2G-13S2C* von *Point Grey* und das Modell *DFK 21AF0* von *The Imaging Source*. Die Kameras sind auf einen Wölbspiegel gerichtet. Dies hat den Vorteil, dass man mit einem relativ kleinen Bild viele Informationen erfassen kann. Beide Kameramodelle liefern YUV422 kodierte Bilder mit einer Auflösung von 640×480 Pixel. Die Bilder werden mit Filtern verarbeitet. Ein Filter ist eine C++-Klasse, welche ein ihr übergebenes Bild verwendet, um bestimmte Ergebnisbilder zu erzeugen. Der Filter erzeugt drei Bilder aus einem YUV422 kodierten Kamerabild, ein Graubild, einen ROI-Kanal und ein Kontrastbild [5]. Die Bilder haben eine Größe von 400×400 Pixel. Der ROI-Kanal enthält für jeden Punkt des 400×400 großen Ausschnitts die Information, ob dieser Relevanz aufweist. Diese Entscheidung findet anhand der Farbwerte, also der Chrominanzwerte, statt. Die interessanten Farbwerte werden in einer Umsetzungstabelle hinterlegt. Abbildung 4.3 zeigt die bisher verwendete Umsetzungstabelle. Die Bestimmung der Ballposition sollte den Roboter nicht mehr als 15 *ms* kosten, damit noch genügend Zeit

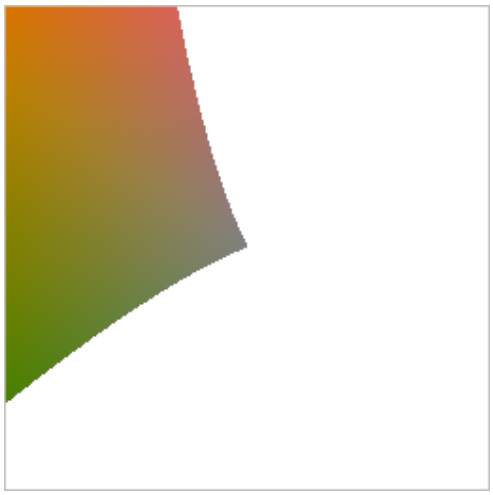


Abbildung 4.3.: Für die farbliche Segmentierung wird bisher diese Umsetzungstabelle verwendet.

für weitere Berechnungen, wie zum Beispiel der Positionsbestimmung des Roboters, bleibt, bevor das nächste Bild geladen wird [5]. Aus diesem Grund ist es sinnvoll, die interessanten Farbwerte nicht bei jedem Bild neu zu bestimmen, sondern in der Umsetzungstabelle zu hinterlegen [6]. Außerdem bietet es die Möglichkeit, die Umsetzungstabelle unabhängig von dem Bildverarbeitungsprozess zu kalibrieren.

4.3. Robot Operating System

Das *Robot Operating System* (ROS) [7] ist ein Framework für den Einsatz auf Robotern. Die bereitgestellten Bibliotheken sind für Linux, Windows und Mac OS X verfügbar. Derzeit existieren zwei stabile Versionen *ROS Fuerte Turtle* (erschienen am 23.04.2012), welche die Universität Kassel verwendet, und *ROS Groovy Galapagos* (erschienen am 31.12.2012). ROS beinhaltet Bibliotheken unter anderem für Hardwareabstraktion, Gerätetreiber und Nachrichtenübertragung [8]. Für diese Arbeit ist lediglich die Nachrichtenübertragung relevant, deshalb werden ausschließlich die benötigten Grundlagen für dieses Teilgebiet näher erklärt.

4.3.1. Node

Unter einem Node[1] versteht man Prozesse, die eine bestimmte Aufgabe erfüllen. Ein Node kann beispielsweise für die Antriebssteuerung zuständig sein. Um die Aufgaben erfüllen zu können, muss dieser mit anderen Nodes gemeinsam arbeiten. Der Antrieb-Node muss beispielsweise mit einem Node kommunizieren, der mit Hilfe von Lasern

[1] Englisch für „Knoten“

eine Positionsbestimmung vornimmt. Dieser Node wiederum erhält Bildinformationen von einem weiteren Node, der eine Kamerainformation extrahiert.

4.3.2. Master

Der ROS Master[2] bildet die Grundlage des Robot Operating Systems. Er stellt die Basisfunktionen zur Verfügung. Zu diesen zählen die Verbindung mit den anderen Nodes und die Verwaltung von Topics,[3] welche das zentrale Element für die Nachrichtenübertragung zwischen den Robotern darstellen. Jeder Roboter muss zu genau einem Master verbunden sein. Dieser wird in der Regel lokal auf dem Roboter gestartet. Das entsprechende Programm dafür ist *roscore*.

4.3.3. Nachrichtenübertragung

Für den Informationsaustausch zwischen einzelner Komponenten der Roboter und der Roboter untereinander existiert in ROS eine eigene Übermittelungsart. Dabei handelt es sich um ein Publish[4]-Subscribe[5] Modell [9]. Diese Übertragungsart findet ohne eine Kopplung zwischen den Teilnehmern statt. Die Informationen werden mit Hilfe von Nachrichten versendet. Es gibt zwei Arten von Teilnehmern, Publisher und Subscriber. Die Nachrichten werden in ROS in Topics versendet. Ein Subscriber meldet sich bei einem Node unter einem bestimmten Topic an. Ab diesem Zeitpunkt erhält der Subscriber alle Nachrichten, die unter der Topic versendet werden. Um Nachrichten zu versenden muss der Prozess die Rolle eines Publishers einnehmen. Dafür muss dieser sich ebenfalls bei einem Node als solcher anmelden. Danach kann er unter dem entsprechenden Topic Nachrichten versenden. Ein Prozess kann gleichzeitig beliebig häufig Publisher oder Subscriber sein.

Bei einer Nachricht handelt es sich um eine Datenstruktur. Diese hat eine festgelegte Struktur, um einen Informationsaustausch fehlerfrei zu ermöglichen. Primitive Nachrichtentypen sind in dem ROS Framework bereits enthalten. Diese können direkt verwendet werden, um einzelne Daten, wie zum Beispiel Integer-, Floatzahlen oder Strings, zu übertragen.

In ROS ist es zusätzlich möglich, komplexe Datenstrukturen mit Hilfe einer Nachricht zu übertragen. Diese werden in *msg*-Dateien definiert. Listing 4.1 zeigt eine Nachricht, welche die Telefonnummer einer Person enthält. Die Nachricht enthält drei Felder, *id*, *name* und *phonedigits*. Alle Felder besitzen unterschiedliche Datentypen. Das

[2]Englisch für „Meister“
[3]Englisch für „Themen“
[4]Englisch für „herausgeben“
[5]Englisch für „abonnieren“

Feld *id* ist eine positive, maximal 32 Bit große Zahl; *name* enthält eine Zeichenfolge und das Feld *phonedigits* ein Array mit positiven, maximal 8 Bit großen Zahlen.

```
uint32      id
string      name
uint8[]     phonedigits
```

Listing 4.1: Eine ROS-Nachricht mit 3 Feldern zur Übertragung von Telefonnummern

Aus den definierten Nachrichten werden mit Hilfe des CMake-Befehls *rosbuild_genmsg()* Quelldateien erzeugt. Diese können dann in den entsprechenden Programmen verwendet werden.

4.4. Clusteranalyse

In der heutigen Zeit sind die Kosten für den Speicherplatz im Verhältnis zu der Größe der Datenmenge, die gespeichert werden soll, sehr günstig. Das hat zur Folge, dass viele Daten gesammelt werden. Diese Daten sollen möglichst in einen logischen Zusammenhang gebracht werden. In [10] ist das Ziel des Clusteringverfahrens folgendermaßen definiert: „Daten [werden] (semi-)automatisch so in Kategorien, Klassen oder Gruppen (*Cluster*) einzuteilen, daß Objekte im gleichen Cluster möglichst ähnlich und Objekte aus verschiedenen Clustern möglichst unähnlich zueinander sind.“ Als ein mögliches Beispiel für die Clusteranalyse nennen Ester und Sander die Erstellung von thematischen Karten aus Satellitenbildern [10]. Die unterschiedlichen Oberflächenteile der Erde, zum Beispiel Wasser, Wiesen oder Eis haben kennzeichnende Reflexions- und Emissionsverhalten bei verschiedenen Teilen des elektromagnetischen Spektrums. Aus diesem Grund werden mehrere Satellitenbilder von der zu betrachtenden Landschaft mit verschiedenen elektromagnetischen Wellenlängen angefertigt. Die aus diesen Bildern gewonnenen Informationen können nun verwendet werden, um ähnliche Gebiete zusammenzufassen. Diese werden in den thematischen Karten, wie in Abbildung 4.4 zu sehen ist, mit den gleichen Farben markiert.

Um Daten automatisch verarbeiten zu können, müssen diese in einer einheitlichen Form vorliegen. Jedem Datenobjekt werden die beobachteten Merkmale zugeordnet. Ein Datenobjekt wird demnach durch eine Liste von Merkmalen charakterisiert,

$$x = (x_1, x_2, \ldots, x_d)\,. \tag{4.2}$$

Diese Darstellungsart bietet die Möglichkeit, Objekte anhand von Distanzmetriken zu vergleichen.

Quelle: `http://www.fis.uni-bonn.de/recherchetools/infobox/einsteiger/faszination-fernerkundung/bilder-und-karten` (Abgerufen am: 13.09.2013)

Abbildung 4.4.: Aus einem Satellitenbild wird unter Verwendung von Clustering eine thematische Karte, *(a)* zeigt das Satellitenbild *(b)* die thematische Karte.

4.4.1. Distanzmetriken

Die Ähnlichkeit von Datenobjekten wird durch ihre Distanz zueinander angegeben. Je kleiner die Distanz zwischen zwei Objekten ist, desto ähnlicher sind diese zueinander. Es gibt für unterschiedliche Merkmalstypen, zum Beispiel numerisch oder kategorisch, verschiedene Metriken zur Berechnung der Distanz zweier Objekte zueinander. Es müssen laut [10] immer folgende Bedingungen erfüllt sein:

1. $\text{dist}(o_1, o_2) = d \in \mathbb{R}^{\geq 0}$
2. $\text{dist}(o_1, o_2) = 0$ genau dann wenn $o_1 = o_2$
3. $\text{dist}(o_1, o_2) = \text{dist}(o_2, o_1)$ Symmetrie
4. $\text{dist}(o_1, o_3) \leq \text{dist}(o_1, o_2) + \text{dist}(o_2, o_2)$ Dreiecksungleichung

Für nummerische Merkmale gibt es 4 gängige Distanz-Metriken, die Euklidische, die Manhattan-, die Maximums- und die Allgemeine L_p-Distanz.

Bei Wahrscheinlichkeitsverteilungen, die bei der Erwartungswertmaximierung verwendet werden, wird häufig die Mahalanobis-Distanz,

$$\text{dist(x, y)}=\sqrt{(x-y)^{\mathrm{T}}\Sigma^{-1}(x-y)}\,, \tag{4.3}$$

für die Bestimmung ähnlicher Objekte verwendet [11]. In dieser wird die Kovarianzmatrix Σ der entsprechenden Verteilung genutzt. Diese gibt einen Zusammenhang

zwischen den Merkmalen der Wahrscheinlichkeitsverteilung an.

4.4.2. Erwartungswertmaximierung

Die *Erwartungswertmaximierung* (EM-Algorithmus) ist ein weit verbreiteter Algorithmus für die Clusteranalyse. Die erfassten Datenobjekte werden als Punkte in einem euklidischen Vektorraum repräsentiert. Die Menge an Merkmalen bestimmt die Anzahl der Dimensionen d des Vektorraums. Bei der Erwartungswertmaximierung wird angenommen, dass die Cluster in Form von Gaußschen Normalverteilungen auftreten. Das hat zur Folge, dass ein Punkt in mehreren unterschiedlichen Clustern gleichzeitig liegen kann. Ein Cluster in der Clustermenge wird nicht wie üblich nur durch einen repräsentativen Punkt charakterisiert, sondern durch 2 Komponenten. Der *Mittelwert aller Punkte im Cluster* μ_i bestimmt die Lage und die $d \times d$ *Kovarianzmatrix* Σ_C definiert die Form des Clusters. Die Kovarianzmatrix enthält auf ihrer Diagonalen die Varianzen der einzelnen Merkmale. In den restlichen Feldern befinden sich die Kovarianzen zwischen den Merkmalen [12]. Das Ziel bei der Erwartungswertmaximierung ist es, die Gaußverteilungen so weit anzupassen, dass die einzelnen Datenobjekte zu einer möglichst großen Wahrscheinlichkeit in einem der Cluster enthalten sind. Der EM-Algorithmus läuft in zwei Schritten ab, dem Erwartungsschritt und dem Maximierungsschritt. Diese werden abwechselnd wiederholt, bis die Änderung des Ergebnisses einen bestimmten Schwellwert unterschreitet.

In dem Erwartungsschritt werden für alle zu betrachteten Datenobjekte die Wahrscheinlichkeiten berechnet, sodass diese einem Cluster zugeordnet werden. Um die Wahrscheinlichkeit zu berechnen, dass ein Datenobjekt x bei nur einem Cluster C in der Datenmenge vorkommt, wird die Formel (4.4) verwendet. In dieser wird die Mahalanobis-Distanz als Abstandsmaß genutzt,

$$P(x|C) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_C|}} e^{\frac{1}{2}(x-\mu_C)^T (\Sigma_C)^{-1} (x-\mu_C)} . \tag{4.4}$$

Mit Hilfe dieser Formel lässt sich die Wahrscheinlichkeit, dass sich ein Datenobjekt x in der Datenmenge befindet, für k Cluster durch folgende Formel bestimmen

$$P(x) = \sum_{i=1}^{k} W_i P(x|C_i) . \tag{4.5}$$

Aus den Formeln 4.4 und 4.5 leitet sich die Wahrscheinlichkeit ab, dass ein Punkt aus

der Datenmenge einem bestimmten Cluster C_i zugeordnet werden kann,

$$P(C_i|x) = W_i \frac{P(x|C_i)}{P(x)}\,. \tag{4.6}$$

In dem Maximierungsschritt werden mittels der im Erwartungsschritt berechneten Werte die Parameter der Gaußverteilungen angepasst. Dabei wird der Erwartungswert der Wahrscheinlichkeit $P(x)$ maximiert. Für alle Cluster C_i werden die Parameter W_i, μ_i und Σ_i berechnet. W_i gibt den Anteil der Datenobjekte an, die zu dem Cluster C_i gehören,

$$W_i = \frac{1}{n} \sum_{x \in D} P(C_i|x)\,. \tag{4.7}$$

Demnach gibt W_i eine Gewichtung der Cluster an. Für die Berechnung von μ_i werden alle beobachteten Datenobjekte D nach der Wahrscheinlichkeit, dass sie in dem Cluster C_i auftreten, anteilig gewichtet,

$$\mu_i = \frac{\sum_{x \in D} x \cdot P(C_i|x)}{\sum_{x \in D} P(C_i|x)}\,. \tag{4.8}$$

Die Kovarianzmatrix Σ_C gibt die Ausprägung von den einzelnen Merkmalen der Objekte im Cluster an. Diese wird mit Hilfe des Mittelwertvektors μ_i berechnet,

$$\Sigma_i = \frac{\sum_{x \in D} P(C_i|x)(x - \mu_i)(x - \mu_i)^T}{\sum_{x \in D} P(C_i|x)}\,. \tag{4.9}$$

Für die Bewertung des Clusterings wird der Erwartungswert $E(M)$ bestimmt. Je größer dieser Wert ist, desto mehr Datenobjekte werden mindestens einem Cluster zugeordnet.

$$E(M) = \sum_{x \in D} \log(P(x))\,. \tag{4.10}$$

Die beiden Schritte des Algorithmus werden so lange wiederholt, bis die Änderung des Erwartungswertes einen bestimmten Schwellenwert ε unterschreitet.

Kapitel 5

Implementierung

Im Laufe dieser Arbeit ist ein Softwarewerkzeug mit dem Namen *ColorCalibrator* (CoCa) entstanden. Die relevanten Teile des *Carpe Noctem*-Framework sind in C++ geschrieben. Um diese nutzen zu können, wurde für die Entwicklung des CoCa ebenfalls C++ verwendet. Zum Bau der grafischen Oberfläche wurde die Bibliothek *Qt* [13] verwendet.

In diesem Kapitel wird zunächst die Integration in das *Carpe Noctem*-Framework erläutert. Anschließend wird die Funktionsweise des CoCa, deren Implementierungsdetails erklärt und die Anwendung des EM-Algorithmus dargelegt.

5.1. Integration in das Framework

Der CoCa ist an einigen Punkten mit dem *Carpe Noctem*-Framework verwoben. Es können zwei Funktionalitäten aus dem Framework direkt verwendet werden. Die YUV422 kodierten Bilder müssen für die Darstellung in eine RGB-Kodierung umgewandelt werden. In dem *Carpe Noctem*-Framework ist bereits eine Funktion vorhanden, die dies leistet. Der Segmentierungsfilter wird verwendet, um die Umsetzungstabelle auf das aktuell dargestellte Bild anzuwenden. Diese Funktion ist für die Kalibrierung der Umsetzungstabelle notwendig, da der Anwender direkt eine Rückmeldung auf die Änderungen an der Umsetzungstabelle bekommt.

Um die Kommunikation zwischen dem CoCa und den Robotern zu ermöglichen, mussten Ergänzungen an dem Framework vorgenommen werden. Es wurden zwei Komponenten implementiert; die eine dient der Anpassung der Kameraparameter und eine andere versendet die angeforderten Kamerabilder an den CoCa.

5.1.1. Übertragung von Kameraparametern

ROS bietet keine Möglichkeit, eine ROS Nachricht an alle Roboter gleichzeitig zu versenden. Diese Funktion wird allerdings bei dem Austausch der ermittelten Daten während eines RoboCup-Spiels häufig benötigt. Ein Roboter müsste also eine Verbindung zu allen ROS Mastern einzeln aufbauen, um diesen eine Nachricht zu versenden. Abbildung 5.1 zeigt, dass in dem *Carpe Noctem*-Framework eine alternative Lösung verwendet wird. Alle ROS Nachrichten werden durch einen Proxy verwaltet. Dieser ver-

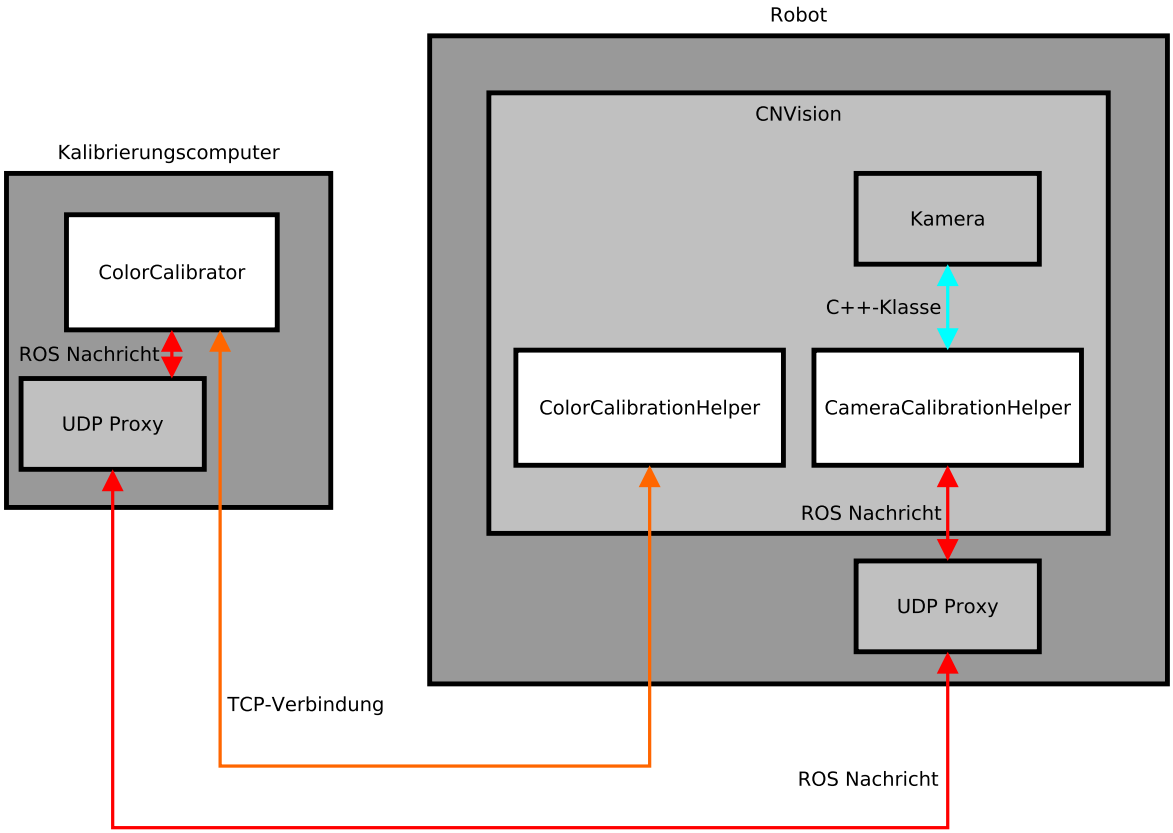


Abbildung 5.1.: Der CoCa kommuniziert über zwei Kanäle mit dem *Carpe Noctem*-Framework. Die Bildübertragung zwischen dem CoCa und dem ColorCalibrationHelper wird über eine direkte TCP-Verbindung realisiert. Kameraparameter werden in einer ROS Nachrichte über zwei UDP Proxys gesendet, der diese an den CameraCalibrationHelper weiterleitet. Der CameraCalibrationHelper greift auf die Funktionen der Kamera mit der entsprechenden C++-Klasse zu.

wirft alle Nachrichten, die vorher nicht in einer Datei als relevant deklariert worden sind. Die relevanten Nachrichten werden per UDP Broadcast an alle erreichbaren Roboter versendet.

Die Kameraparameter werden durch den *CameraCalibrationHelper* mit Hilfe einer ROS Nachricht versendet und empfangen. Bei dem Versenden der Kameraparameter

von dem CoCa zu den Robotern ist die zu versendende Menge an Daten einerseits abschätzbar und andererseits relativ gering. Alle Kameraparameter lassen sich deshalb mit einer ROS Nachricht versenden. Es werden zwei Nachrichten für die Übertragung verwendet. Eine dient dazu, das Versenden der aktuellen Kameraeinstellungen von ausgewählten Robotern anzufordern; sie wird durch den CoCa versendet. Listing 5.1 zeigt, dass diese Nachricht lediglich ein Array enthält. In diesem werden die IDs[1] von allen Robotern hinterlegt, deren Konfigurationen benötigt werden. Mit Hilfe der ID können alle Roboter eindeutig identifiziert werden. Wenn ein Roboter eine solche Nachricht empfängt und seine ID in dem Array enthalten ist, dann versendet er seine aktuellen Kameraeinstellungen in der dafür vorgesehenen Nachricht.

```
1 int32[]    receiverID
```

Listing 5.1: Die ROS Nachricht zum Beantragen von Kameraeinstellungen

Sowohl für das Versenden der aktuellen, als auch neuer Konfigurationen einer Kamera wird die gleiche Nachricht verwendet. In Listing 5.2 ist zu sehen, dass die Nachricht zwei Felder für IDs enthält, die *senderID* und die *receiverID*. Die *senderID* wird verwendet, wenn ein Roboter seine aktuellen Einstellungen an den CoCa überträgt. Der Roboter füllt die Felder mit den entsprechenden Werten und setzt die *receiverID* auf -1, was einem ungültigen Wert entspricht. Wenn die Kameraparameter bei einem Roboter verändert werden sollen, verschickt der CoCa eine entsprechende Nachricht. Dabei wird die *receiverID* auf die ID des betreffenden Roboters gesetzt und wiederum die *senderID* auf -1.

```
1  int32    senderID
   int32    receiverID
3  bool     useBrightness
   int32    brightness
5  int32    exposure
   bool     autoWhiteBalance
7  int32    whiteBalance1
   int32    whiteBalance2
9  int32    hue
   int32    saturation
11 bool     enabledGamma
   int32    gamma
13 bool     autoShutter
   int32    shutter
15 bool     autoGain
   int32    gain
```

Listing 5.2: Die ROS Nachricht zum Versenden von Kameraparametern

[1] Abkürzung für identifier, Englisch für „Kennung“

5.1.2. Bildübertragung

Jegliche Kommunikation zwischen den Robotern findet mit Hilfe der ROS Nachrichtenübertragung statt. Aus diesem Grund wurde zunächst ebenfalls versucht, die Bilder unter Verwendung von ROS zu versenden. In dem *Carpe Noctem*-Framework ist der *ColorCalibrationHelper* für die Übermittelung von Bildern zuständig. Listing B.1 zeigt den ersten Entwurf für eine entsprechende ROS Nachricht. Nach den ersten Testläufen hat sich allerdings herausgestellt, dass die Pakete, sobald sie die maximale UDP-Paketgröße von 65.535 Byte überschritten haben, durch den Proxy verworfen werden. Auch das Aufteilen der Bildinformationen in passende Pakete mit einer maximalen Größe von 65.535 Byte erzielte keinen Erfolg. Wegen bei der Übertragung verloren gegangener Pakete konnte nahezu kein Bild rekombiniert werden. Stattdessen werden die Bilder außerhalb von ROS mit einer TCP-Verbindung versendet.

Abbildung 5.1 zeigt, dass der CoCa bei der Bildübertragung direkt mit dem *ColorCalibrationHelper* kommuniziert. Dafür startet der *ColorCalibrationHelper* bei dem Beginn des Bildverarbeitungsprozesses einen Thread. Dieser läuft so lang, bis die Bildverarbeitung beendet wird, parallel zu dieser ab. In dem Thread wird ein Socket geöffnet, welcher ununterbrochen darauf wartet, dass der CoCa eine TCP-Verbindung mit einem bestimmten Port aufbaut. Sobald dies geschehen ist, wird das nächste aufgenommene Bild über diese Verbindung übertragen. Anschließend wird die Verbindung wieder geschlossen. Der Socket bleibt während des gesamten Vorgangs geöffnet und könnte demnach parallel weitere Anfragen entgegen nehmen.

5.2. Das Werkzeug

Im Folgenden werden die einzelnen Funktionen des CoCa und die Arbeitsweise erläutert. In Abbildung D.1 ist die vollständige Benutzeroberfläche zu sehen. Die Aufteilung der Abschnitte orientiert sich dabei an der in Abschnitt 3.2 gestellten Zielsetzung.

5.2.1. Anzeige von Kamerabildern

Eine grundlegende Funktion des CoCa ist die Anzeige von Bildern. Diese können sowohl aus einer lokal gespeicherten Datei geladen als auch direkt von einem bestimmten Roboter übertragen werden. Die Kameras der Roboter stellen YUV422 kodierte Bilder zur Verfügung. Für die Darstellung der Bilder müssen diese in RGB888 umgewandelt werden. Dies ist notwendig, da die verwendete *Qt*-Klasse *QImage* lediglich mit dem RGB-Farbraum arbeiten kann. Für die Umwandlung wird ein in dem *Carpe Noctem*-Framework bereits vorhandener Filter verwendet.

5.2.2. Aktuelle Bilder laden

Die Möglichkeit, aktuelle Bilder von einem oder mehreren Robotern übertragen zu können, ist essenziell, um die Umsetzungstabelle anpassen oder neu initialisieren zu können. Es gibt zwei wichtige Anwendungsfälle für das Laden von aktuellen Bildern. Bei fehlerhaftem Verhalten eines Roboters können diese betrachtet und für eine spätere Analyse gespeichert werden. Des Weiteren ist es sinnvoll für die Kalibrierung der Umsetzungstabelle Farbwerte aus den Bildern der Roboter zu verwenden, da exakt diese Farben in dem späteren Ballerkennungsverfahren gesucht werden. Ein mögliches Initialisierungsszenario könnte sein: Ein Roboter fährt in alle 4 Ecken des Spielfelds. An jeder Ecke werden Bilder an das Werkzeug gesendet. Dabei wäre es sinnvoll, entweder mehrere Bälle an unterschiedliche Positionen zu positionieren oder, sollte nur ein Ball zur Verfügung stehen, mehrere Bilder mit diversen Ballpositionen anzufertigen. Generell gilt, je mehr Bilder zur Kalibrierung zur Verfügung stehen, desto besser.

Mit dem CoCa ist es möglich Bilder sowohl von einzelnen als auch von allen Robotern gleichzeitig zu übertragen. Abbildung 5.2 zeigt den für die Bildübertragung relevanten Teil des CoCa. Bei der Benutzung des *selected*-Knopfes werden Bilder von den selektierten Robotern geladen. Mit dem *all*-Knopf überträgt der *CoCa* von allen in der Liste befindlichen Robotern ein Kamerabild. Die Übertragung der Bilder findet mit dem in Abschnitt 5.1.2 erläuterten Verfahren statt.

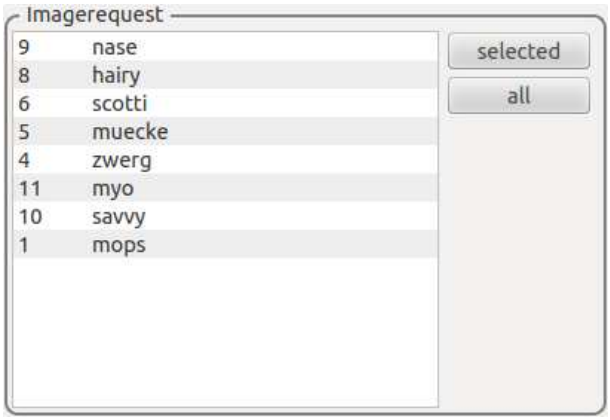


Abbildung 5.2.: Von den in der Liste ausgewählten Robotern können Bilder übertragen werden.

5.2.3. Umsetzungstabelle bearbeiten

Für die Bearbeitung der Umsetzungstabelle stehen zwei Möglichkeiten zur Verfügung: Erstens ist es möglich, Farbwerte direkt zu der Umsetzungstabelle hinzuzufügen oder zu löschen und zweitens lassen sich Regionen in einem geladenen Bild markieren, um

die enthaltenen Farben in die Tabelle zu schreiben beziehungsweise zu entfernen.

Abbildung 5.3 (Lookuptable) zeigt, dass in dem CoCa immer die aktuelle Umsetzungstabelle dargestellt wird. Mit der Maus kann bei gedrückter linker Maustaste ein Rechteck gezogen werden. Die markieren Farbwerte werden, wie in Abbildung 5.4 zu sehen, in die Umsetzungstabelle geschrieben. Mit den Knöpfen kann die Tabelle geleert oder auf die Standardwerte zurückgesetzt werden. Abbildung 5.3 (YUV) zeigt eine weitere Möglichkeit, Farbwerte direkt zu der Umsetzungstabelle hinzufügen. Zu

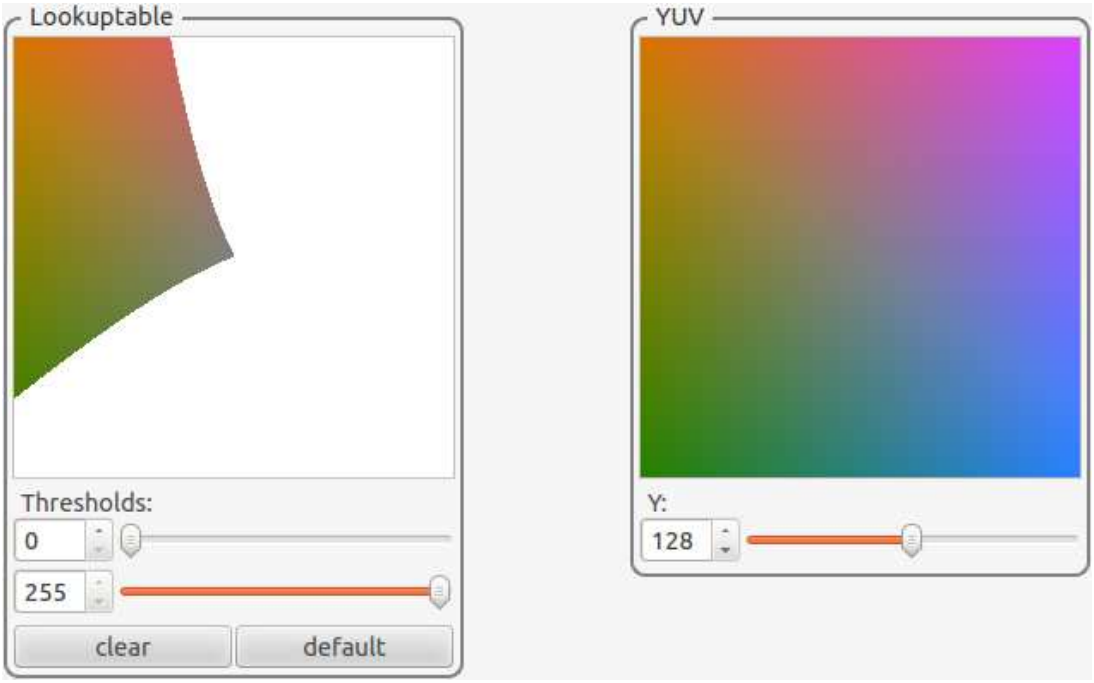


Abbildung 5.3.: Visuelle Darstellung der Umsetzungstabelle und des YUV-Farbraums. (Lookuptable) Farbwerte können ausgewählt und hinzugefügt werden. (YUV) YUV-Farbraum wird mit einstellbarer Luminanz dargestellt. Farbwerte können ausgewählt und zur Umsetzungstabelle hinzugefügt werden.

diesem Zweck werden diese direkt in dem Farbraum markiert. Der Y-Wert dient lediglich zur Anzeige der dargestellten Farben.wurde Für die Umsetzungstabelle werden lediglich die U- und V-Werte verwendet. Während des Rechteckziehens werden, bei gedrückter rechter Maustaste, die markierten Farbwerte aus der Umsetzungstabelle entfernt. Wird beim Hinzufügen von Farben die STRG-Taste gedrückt, werden die eingegrenzten Punkte interpoliert. Dabei werden die nicht erfassten Farbwerte in den Zwischenräumen zu der Umsetzungstabelle hinzugefügt.

Unter der Verwendung von Kamerabildern lassen sich die einzutragenden Farbwerte einfacher und detaillierter bestimmen. Wie bei dem direkten Bearbeiten der Umsetzungstabelle können mit der Maus Rechtecke über die Bilder gezogen werden. In Abbildung 5.5 werden die in dem grünen Rechteck enthaltenen Farben zu der Umsetzungstabelle hinzugefügt.

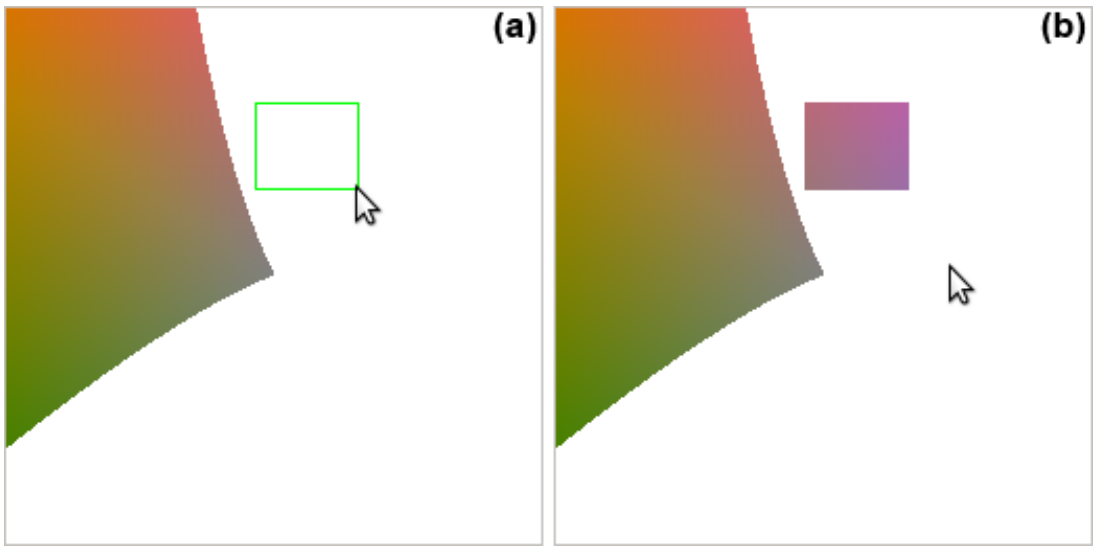


Abbildung 5.4.: Farbwert können mit dem Mauszeiger ausgewählt und zur Umsetzungstabelle hinzugefügt werden. *(a)* Auswahl eines Farbbereiches durch Ziehen eines Rechtecks mit dem Mauszeiger. *(b)* Ausgewählter Farbbereich ist hinzugefügt.

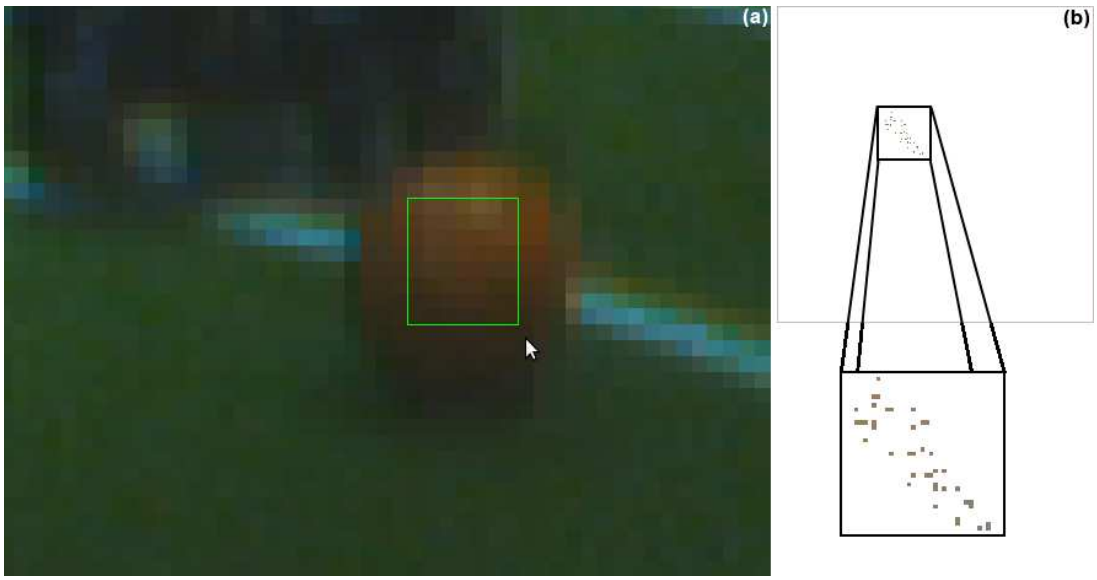


Abbildung 5.5.: Die mit dem Mauszeiger im Bild markierten Farbwerte werden zu der Umsetzungstabelle hinzugefügt. *(a)* Auswahl eines Bildbereiches zum Hinzufügen in die Umsetzungstabelle. *(b)* Resultierende Umsetzungstabelle durch die Auswahl im Bild.

5.2.4. Umsetzungstabelle speichern/laden

In der Menüzeile unter dem Punkt *File* befindet sich die Möglichkeit, die Umsetzungstabelle in eine Datei zu speichern beziehungsweise zu laden. Diese Funktion ist für zwei Anwendungsfälle sinnvoll. Zum einen kann die Umsetzungstabelle für eine spätere Bearbeitung gespeichert werden und zum anderen lässt sich die gespeicherte Tabelle an die Roboter übertragen, damit diese sie verwenden können. Die Umsetzungstabelle wird in einer Textdatei gespeichert. Dabei werden die Werte mit Leerzeichen getrennt und nach jeder Zeile wird ein Zeilenumbruch hinzugefügt. Diese Speicherart hat den Vorteil, dass die gespeicherte Textdatei in MATLAB als Matrix geladen und bearbeitet werden kann. Die Bearbeitung der gespeicherten Tabelle kann auch mit jedem herkömmlichen Texteditor durchgeführt werden.

5.2.5. Kameraparameter modifizieren

Eine gut eingestellte Kamera an den Robotern ist wichtig, um störungsfreie Bilder aufnehmen zu können. Zu den Kameraeinstellungen zählen beispielsweise die Helligkeit oder der Weißabgleich. Letzterer ist besonders wichtig für ein einwandfreies Bild, da dieser einen großen Einfluss auf die angezeigten Farben aufweist. Vor der Entwicklung des CoCa war es lediglich möglich, die Parameter in einer Konfigurationsdatei zu hinterlegen. Diese werden beim Starten der Bilderkennung eingelesen. Abbildung 5.6 zeigt alle anpassbaren Werte. Mit Hilfe des *sendSettings*-Knopfes werden die Einstellungen an den ausgewählten Robotern mit Hilfe von ROS übertragen. Der entsprechende Roboter wird über der Bildanzeige in einem Auswahlmenü eingestellt.

Abbildung 5.6.: Durch Schieberegler verstellbare Kameraparameter des ColorCalibrators.

5.2.5.1. Konfigurationsdateien

Es gibt eine Vielzahl von Informationen und Konfigurationen, die für alle Roboter gleich sind. Zu diesen zählen zum Beispiel die Größe des Spielfeldes und die Positionen der Tore. Die Kameraeinstellungen müssen allerdings für jeden Roboter einzeln

konfiguriert werden. Die unterschiedlichen Werte werden in Konfigurationsdateien organisiert. Diese Dateien sind Textdateien mit einer festgelegten Formatierung. In Listing 5.3 ist zu erkennen, dass es Kategorien gibt, in denen die Werte zusammengefasst sind. Diese Kategorien werden mit eckigen Klammern gekennzeichnet.

```
[Globals]
  OwnTeamColor = cyan
  OwnGoalColor = yellow
  [Dimensions]
    DiameterBall = 220.0
    DiameterRobot = 500.0
  [!Dimensions]
[!Globals]
```

Listing 5.3: Eine Beispiel-Konfigurationsdatei

Die Konfigurationsdateien werden in dem CarpeNoctem-Git-Repository gespeichert. Anhand der Ordnerstruktur können die Einstellungen für die Roboter spezifiziert werden. Es gibt für jede Datei eine Standarddatei; diese liegt in dem Wurzelverzeichnis. Wenn die Konfigurationen von dem Standard abweichen sollen, dann wird eine Konfigurationsdatei mit demselben Namen in einem Unterordner hinterlegt. In dieser können alternative Werte eingestellt werden. Der Name des Ordners ist gleich dem Namen des betreffenden Roboters, zum Beispiel „savvy“. Dieses Vorgehen wird beispielsweise bei den Kameraparametern angewandt, weil alle Kameras an den Robotern unterschiedliche Feinabstimmungen benötigen.

In dem *Carpe Noctem*-Framework gibt es Klassen, die diese Konfigurationsdateien in der verwendeten Programmiersprache in entsprechende Datenstrukturen umwandeln. Die eingetragenen Werte können so in primitive Datentypen übertragen werden. Der Wert *DiameterBall* in Listing 5.3 ist beispielsweise sinnvoll als *double*-Zahl zu verarbeiten.

Das Werkzeug ist mit Hilfe der im Framework vorhandenen Bibliothek ebenfalls in der Lage, die Konfigurationsdateien einzulesen. Für die Minimal- und Maximalwerte der Kameraparameter wurde eine eigene Konfigurationsdatei mit dem Namen „CameraVendor“ angelegt. Diese kann in Listing C.1 eingesehen werden.

5.3. Interpolation

In diesem Abschnitt wird die konkrete Umsetzung des EM-Algorithmus und ihr Nutzen beschrieben. Die Möglichkeit der Interpolation der in der Umsetzungstabelle enthaltenen Farbwerte ist eine sinnvolle Funktionalität, weil immer nur exakt die markierten Ballfarben zu der Umsetzungstabelle hinzugefügt werden. Abbildung 5.5 zeigt,

dass zwischen diesen Farbwerten nicht erfasste Räume entstehen. Diese ausgelassenen Farbwerte treten allerdings in der Regel auch auf, wenn sich die Position des Balles oder des Roboters ändert. Deshalb ist es sinnvoll, diese Zwischenräume aufzufüllen. Dies soll automatisiert erfolgen.

Die verwendeten Bälle sind meist einfarbig. Dies hat zur Folge, dass die entsprechenden Farben in dem YUV-Farbraum relativ nah beieinander liegen (siehe Abbildung 5.5). Aus diesem Grund wird der EM-Algorithmus genutzt, um die Zwischenräume in den Farbwerten aufzufüllen. Der EM-Algorithmus hat zwei Vorteile; zum einen haben die entstandenen Cluster eine beliebig elliptische Form, zum anderen kann ein Objekt zu beliebig vielen Clustern gleichzeitig zugeordnet werden. Diese beiden Eigenschaften können verwendet werden, um beliebige Strukturen abzubilden. Dabei wird eine relativ hohe Anzahl von Clustern gewählt.

In Abbildung 5.7 sind die durch den Benutzer definierbaren Parameter für das Auffüllen der Punkte zu sehen:

- Mit *Minimum prior* wird die Wahrscheinlichkeit P_{min} eingestellt, die ein Farbwert mindestens haben muss, damit er einem Cluster zugeordnet werden kann. Dieser Wert wird verwendet, nachdem der EM-Algorithmus angewendet wurde.

- Der Benutzer kann die Anzahl der Cluster k über das Feld *Clustercount* festlegen.

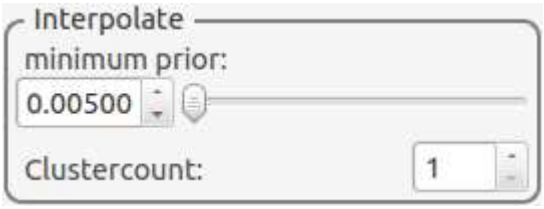


Abbildung 5.7.: Mindest Wahrscheinlichkeit und Clusteranzahl können durch Eingabe bestimmt werden.

Es werden lediglich die mit dem Rechteck markierten Punkte für die Clusteranalyse herangezogen. Aus diesem Grund werden zu Beginn alle in dem Rechteck enthaltenen Punkte erfasst. Dabei wird der Wertebereich von $[255; 255]$ auf $[25,5; 25,5]$ geändert, indem alle Werte durch 10 geteilt werden. Dies ist nötig, weil die Zwischenwerte im Laufe der Anwendung des EM-Algorithmus sonst außerhalb des *double*-Wertebereichs liegen.

Bei dem EM-Algorithmus hat die Initialisierung der Cluster einen großen Einfluss auf das Endergebnis. Abbildung 5.8 zeigt, dass sich die Ergebnisse der Clusteranalyse, bei zufällig initialisierten Clustern erheblich unterscheiden. Deshalb wird für die Interpolation eine zuverlässige Initialisierungsmethode verwendet.

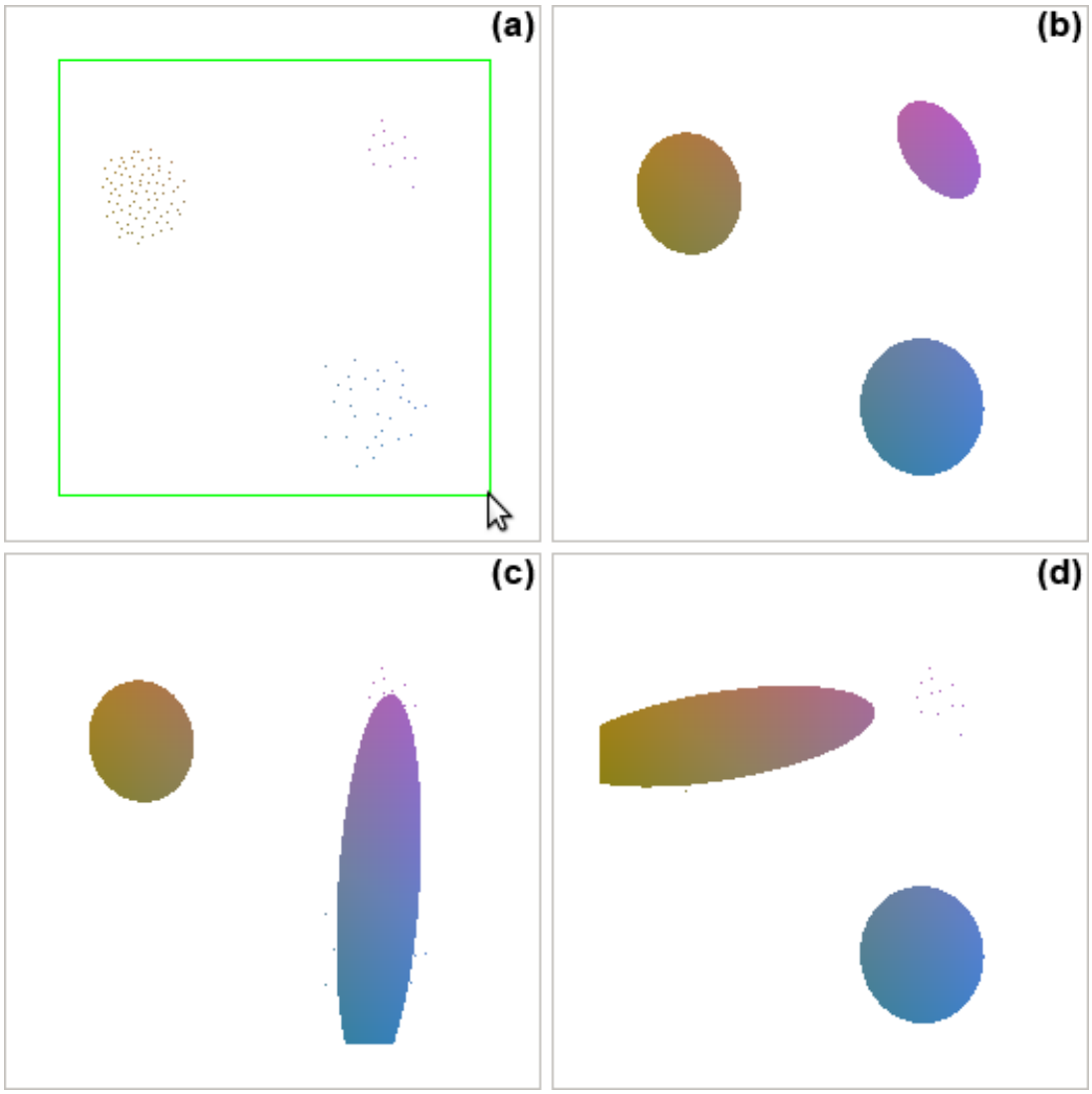


Abbildung 5.8.: EM-Algorithmus mit zwei zufälligen Initialisierungen. *(a)* Die Ausgangswerte können bestmöglich *(b)* das gewünschte Ergebnis zeigen. *(c)* und *(d)* illustrieren zwei mögliche Ergebnisse mit $P_{min} = 0.006$ und $k = 3$.

5.3.1. Initialisierung

Um möglichst alle Punkte mit den Clustern erfassen zu können, werden die initialen Clusterzentren μ_i in Punkte aus den gegebenen Daten D gelegt. Außerdem werden die repräsentierenden Punkte gewählt, die möglichst weit voneinander entfernt sind. Um die Laufzeit zu verringern, wird eine maximale Anzahl von Punkten, die für die Initialisierung verwendet werden, definiert. Diese beträgt 25 zufällige Punkte pro Cluster. Zu Beginn werden die Distanzen zwischen allen verwendeten Punkten D_i bestimmt. Dabei wird als Distanzmaß die Euklidische Distanz,

$$\text{dist(x, y)}=\sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}\,, \tag{5.1}$$

verwendet. Anschließend werden, wenn mindestens zwei Cluster gefordert sind, die beiden am weitesten voneinander entfernten Punkte gesucht. Diese Punkte werden als Clusterzentren für die ersten beiden Cluster verwendet. Anschließend werden jeweils für alle restlichen Cluster die übrigen Punkte durchlaufen. Es wird derjenige Punkt verwendet, der von den bisherigen Clusterzentren $Z = (\mu_1, \mu_2, ...)$ am weitesten entfernt ist. Jedes Mal, wenn die verwendeten Punkte durchgelaufen wurden, wird der Punkt x mit dem größtem mittleren Abstand,

$$\text{dist}_{\text{Mittel}}(x) = \frac{1}{|Z|} \sum_{\mu \in Z} \text{dist}(x, \mu)\,, \tag{5.2}$$

und mit der geringsten mittleren Abweichung,

$$\text{dist}_{\text{Abweichung}}(x) = \frac{1}{|Z|} \sum_{\mu \in Z} |\text{dist}_{\text{Mittel}}(x) - \text{dist}(x, \mu)|\,, \tag{5.3}$$

gesucht. Da dieser Punkt von den bisherigen Clusterzentren am weitesten entfernt ist, wird er für den aktuellen Cluster C_i als Clusterzentrum μ_i verwendet. Abbildung 5.9 zeigt, dass die Clusterzentren mit der gezeigten Initialisierungsmethode in alle Ballungsgebiete verteilt wurden. Der EM-Algorithmus liefert daraufhin das gewünschte Ergebnis.

5.3.2. EM-Algorithmus Anwendung

Der EM-Algorithmus wird auf die Farbwerte in der Umsetzungstabelle angewendet, um diese zu interpolieren. Die Datenobjekte haben somit zwei beobachtete Merkmale, den U- und den V-Wert. Die Dimension d des Vektorraums, welche von U und V aufgespannt wird, beträgt somit 2. Für die Bewertung der Cluster wird, wie in Ab-

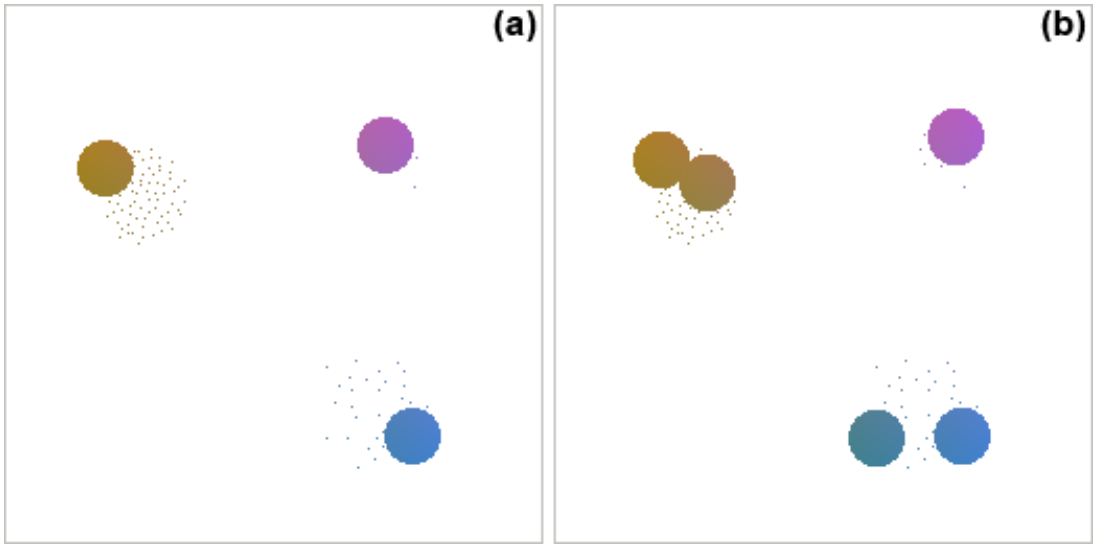


Abbildung 5.9.: Zwei Clusterinitialisierung vor der Anwendung des EM-Algorithmus mit $P_{min} = 0.05$, *(a)* $k = 3$ und *(b)* $k = 5$.

schnitt 4.4.2 erläutert, der Erwartungswert $E(M)$ verwendet. (siehe Formel 4.10) Das Abbruchkriterium für den Algorithmus ist die Veränderung des Erwartungswertes,

$$\text{diff(M, M')} = |E(M) - E(M')| \,. \tag{5.4}$$

M' repräsentiert das bisherige Modell und M die neu berechneten Cluster. Liegt die Differenz und somit die Veränderung zwischen den beiden Modellen unter einem bestimmtem Schwellwert ε, dann bricht der Algorithmus ab. Als Wert für ε hat sich im Laufe der Testphase 10^{-5} als ausreichend herausgestellt.

5.3.3. Prognose

Nachdem mit Hilfe des EM-Algorithmus die Cluster geformt wurden, müssen die darin enthaltenen Punkte in die Umsetzungstabelle hinzugefügt werden. Zu diesem Zweck werden alle Punkte, welche mit dem Rechteck markiert wurden, iteriert. Für die Farbwerte, die in der ursprünglichen Umsetzungstabelle noch nicht eingetragen waren, wird geprüft, ob diese zu einem Cluster zugeordnet und somit zu der Umsetzungstabelle hinzugefügt werden können. Um dies zu entscheiden, wird die Formel 4.4 verwendet. Wenn die Wahrscheinlichkeit über der von dem Benutzer definierten Wahrscheinlichkeit P_{min} liegt, wird der entsprechende Punkt x in die Umsetzungstabelle hinzugefügt. Mit der Wahrscheinlichkeit P_{min} kann der Benutzer demnach die Größe der entstehenden Cluster bestimmen. Abbildung 5.8 *b* zeigt den erfolgreich angewendeten EM-Algorithmus.

Kapitel 6

Evaluation

Bei der Evaluation des CoCa werden zwei Ziele verfolgt. Zum einen wird die Effizienz des CoCa bei der Kalibrierung gegenüber der herkömmlichen Methode ermittelt und zum anderen die Güte der Interpolation durch den EM-Algorithmus gezeigt. Zu diesem Zweck wird eine Kalibrierung von Umsetzungstabellen vorgenommen. Dazu wird die Umgebung eines RoboCup-Turniers nachgestellt. Mit einem Roboter werden auf einem RoboCup-Spielfeld Kalibrierungsbilder aufgenommen, anhand derer Umsetzungstabellen erzeugt werden. Die Tabellen werden im Anschluss anhand von Testbildern, welche mit dem selben Roboter aufgenommen werden, evaluiert. Da die ursprüngliche Umsetzungstabelle lediglich bei gelben und orangefarbenen Bällen funktioniert, hat der Testball eine gelbe Farbe. Es werden insgesamt acht Umsetzungstabellen getestet; einerseits die ursprüngliche Umsetzungstabelle und andererseits sieben mit dem CoCa erstellten Tabellen.

6.1. Umsetzungstabellen

In diesem Abschnitt werden alle evaluierten Umsetzungstabellen, welche mit dem CoCa erstellt sind, erklärt. Abbildung 6.1 zeigt, dass die Umsetzungstabellen aufeinander aufbauen. Die Basis für alle Tabellen bildet die *Grundlagen*-Tabelle. Diese wird erzeugt, indem in allen Kalibrierungsbildern der Ball markiert wird und dessen Farbwerte zu der Umsetzungstabelle hinzugefügt werden. In Abbildung 6.3 (a) ist zu sehen, dass die *Grundlagen*-Tabelle einige grüne Farbwerte enthält.

Das in Abbildung 6.2 gezeigte Kalibrierungsbild zeigt zwei Gründe für dieses Ergebnis: Erstens befindet sich grüne Schrift auf dem Ball und zweitens reflektiert sich der Spielfeldboden an der Unterseite des Balles. Aus diesem Grund sind weitere Anpassungen an der Umsetzungstabelle notwendig.

Für die Optimierungsversuche einer Umsetzungstabelle gibt es zwei wesentliche

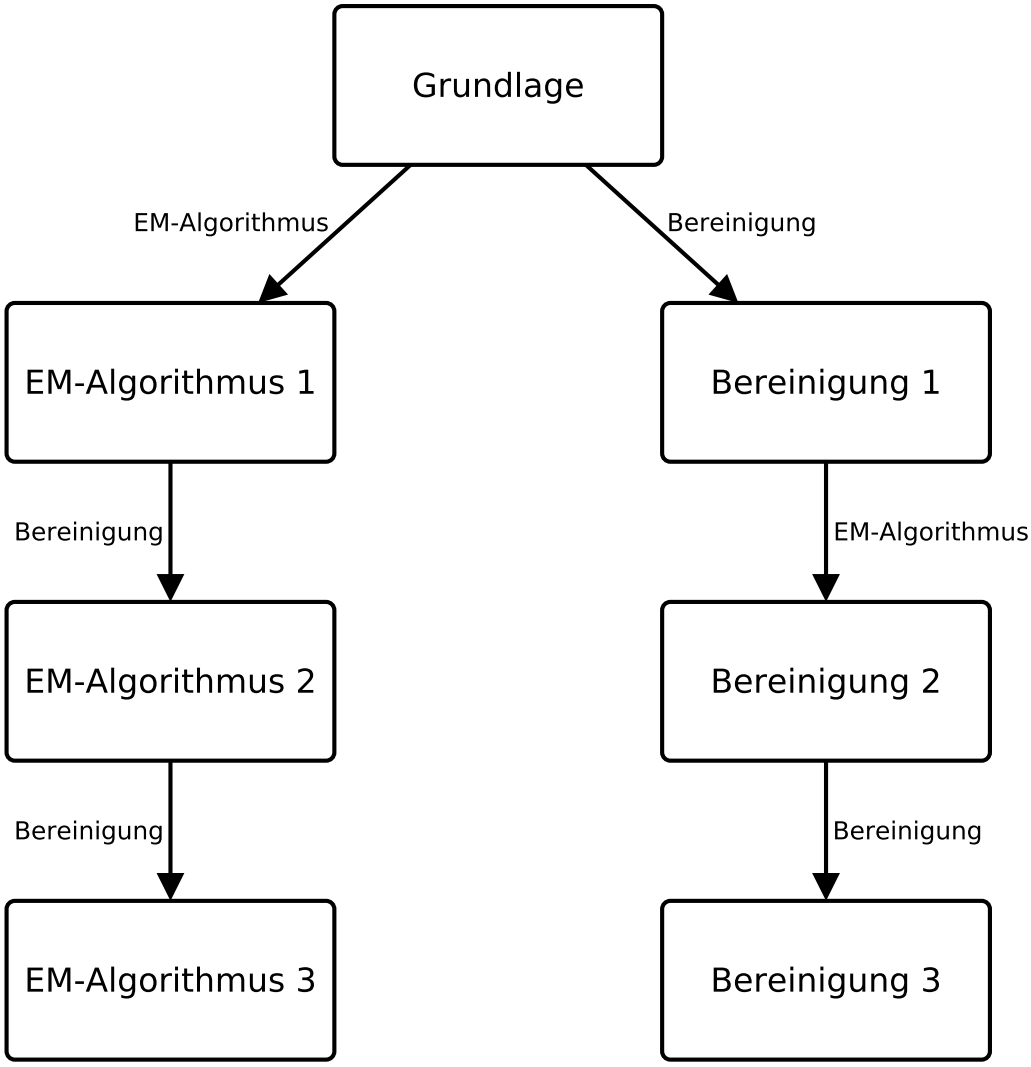


Abbildung 6.1.: Auf die *Grundlagen*-Tabelle werden zwei Vorgehensweisen angewendet, ein Zweig basierend auf dem EM-Algorithmus, einer beginnend mit einem Bereinigungsschritt. An den Pfeilen sind die verwendeten Schritte angegeben.

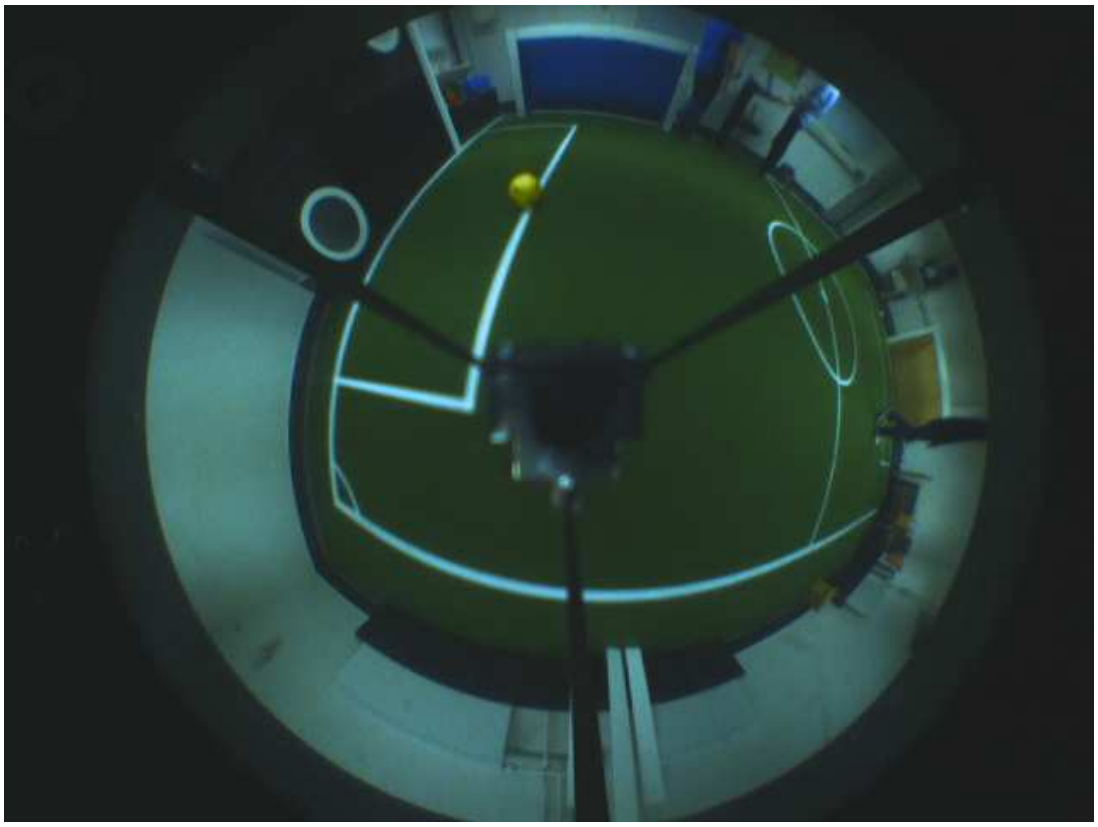


Abbildung 6.2.: Bei den Kalibrierungsbildern liegt nur der gesuchte Ball auf dem Spielfeld.

Vorgehensweisen. Die erste Maßnahme ist das *Bereinigen* der Umsetzungstabelle, um falsch erkannte Punkte zu vermeiden. Für die Bereinigung gibt es zwei Möglichkeiten, entweder werden bei allen Kalibrierungsbildern die Farbwerte markiert und aus der Umsetzungstabelle gelöscht, welche fälschlicherweise als Ball erkannt werden, oder die Farbwerte werden direkt aus der Tabelle entfernt. Bei der zweiten Optimierungsmethode handelt es sich um die Anwendung des *EM-Algorithmus*, dabei werden weitere Farbwerte zu der Umsetzungstabelle hinzugefügt, um somit Ballfarben, die in den Kalibrierungsbildern nicht enthalten sind, zu erfassen.

Neben der *Grundlagen*-Tabelle gibt es jeweils drei Umsetzungstabellen mit dem Namen *EM-Algorithmus* und *Bereinigen*. Diese Tabellen bauen aufeinander auf; der Name richtet sich danach, welche Optimierungsmethode auf die *Grundlagen*-Tabelle angewendet wird.

Die *EM-Algorithmus*-Tabellen entwickeln sich daraus, den EM-Algorithmus direkt auf die Farbwerte in der *Grundlagen*-Tabelle anzuwenden. Abbildung 6.3 zeigt sowohl die *Grundlagen*-Tabelle als auch die drei *EM-Algorithmus*-Tabellen. Die *EM-Algorithmus 1*-Tabelle ist das Ergebnis des angewendeten EM-Algorithmus auf die *Grundlagen*-Tabelle. Es fällt auf, dass diese Umsetzungstabelle ein großes Spektrum an Farbwerten abdeckt. Aus diesem Grund wird im nächsten Schritt eine Bereinigung der Tabelle

durchgeführt. Diese findet anhand der Kalibrierungsbilder statt. In allen Bildern werden die Stellen, welche durch die Umsetzungstabelle fälschlicherweise als Ballfarbe klassifiziert werden, markiert und gelöscht. Die entsprechende Umsetzungstabelle ist mit *EM-Algorithmus 2*-Tabelle benannt. In Abbildung 6.3 (c) ist zu sehen, dass bei der Bereinigung einige Farbwerte übrig geblieben sind. Diese werden in *EM-Algorithmus 3*-Tabelle direkt aus der Umsetzungstabelle entfernt.

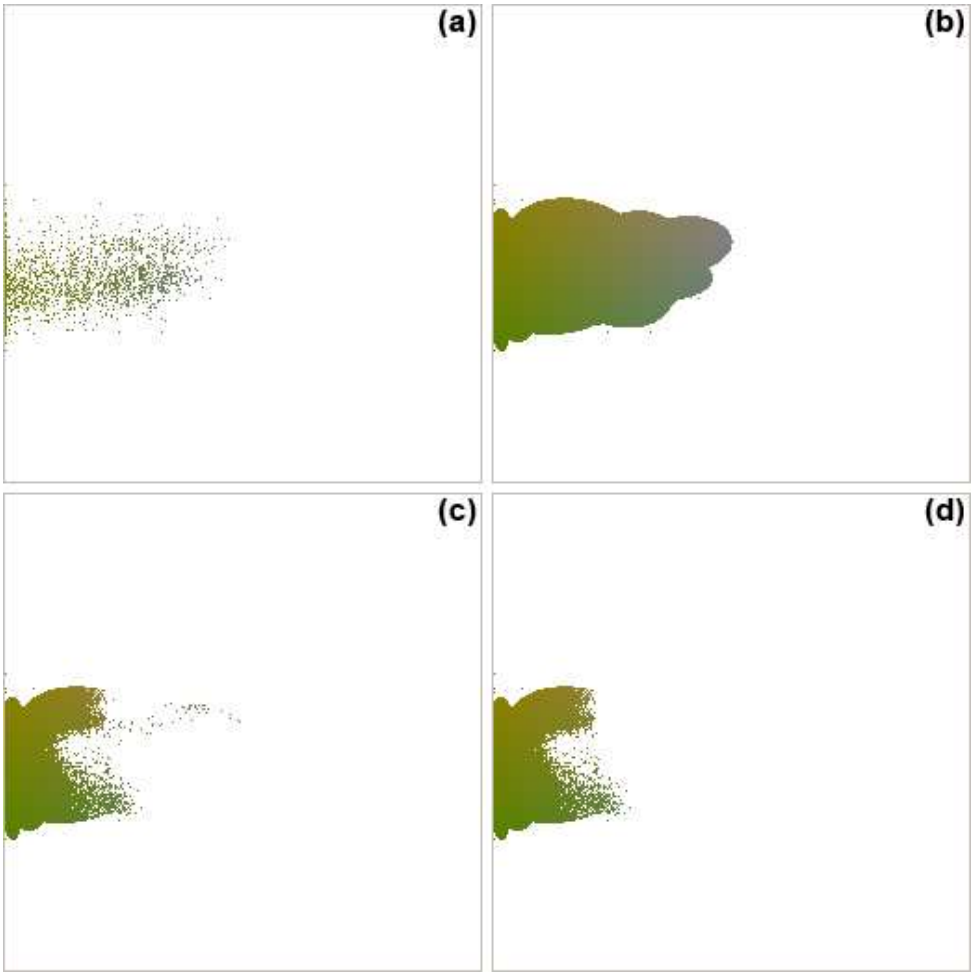


Abbildung 6.3.: Umsetzungstabellen aufbauend auf EM-Algorithmus. *(a)* Zeigt die *Grundlagen*-Tabelle, *(b)* die *EM-Algorithmus 1*-Tabelle nach Anwendung des EM-Algorithmus, *(c) EM-Algorithmus 2*-Tabelle nach einer Bereinigung und *(d)* nach *EM-Algorithmus 3*-Tabelle nach einem weiteren Bereinigungsschritt.

Die *Bereinigung*-Tabellen sehen vor, die Daten vor der Anwendung des EM-Algorithmus zu bereinigen. Analog zu den *EM-Algorithmus*-Tabellen zeigt Abbildung 6.4 zum einen die *Grundlagen*-Tabelle und zum anderen die drei daraus resultierenden *Bereinigung*-Tabellen. Nachdem die *Grundlagen*-Tabelle erstellt ist, wird diese direkt bereinigt. Zu diesem Zweck werden erneut die Kalibrierungsbilder durchlaufen und alle fälschlicherweise als Ballfarbe erkannten Farbwerte aus der Umsetzungstabelle ge-

löscht. Die daraus entstandene Tabelle ist die *Bereinigung 1*-Tabelle. Auf diese wird der EM-Algorithmus angewendet, wodurch sich *Bereinigung 2*-Tabelle ergibt. Zuletzt wird die Umsetzungstabelle genauso bereinigt, wie bereits die *Bereinigung 1*-Tabelle. Die entsprechende Umsetzungstabelle lautet *Bereinigung 3*-Tabelle.

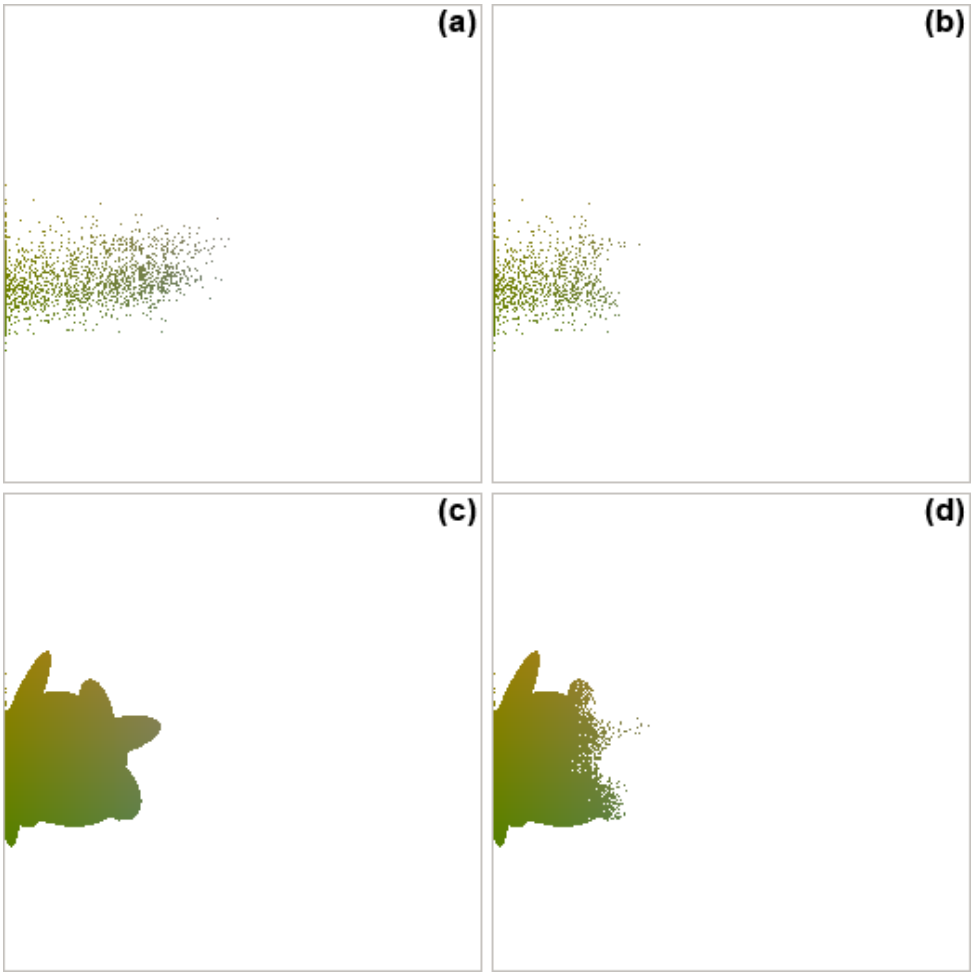


Abbildung 6.4.: Umsetzungstabellen aufbauend auf einer Bereinigung. *(a)* Zeigt die *Grundlagen*-Tabelle, *(b)* die *Bereinigung 1*-Tabelle nach einer Bereinigung, *(c) Bereinigung 2*-Tabelle nach Anwendung des EM-Algorithmus und *(d)* nach *Bereinigung 3*-Tabelle nach einem weiteren Bereinigungsschritt.

Die sieben mit dem CoCa erstellten Umsetzungstabellen werden zuzüglich der ursprünglichen Tabellen mit statistischen Bewertungsmethoden evaluiert.

6.2. Bewertungsmethoden

Für die Bewertung der acht Umsetzungstabellen werden 13 Testbilder verwendet. Abbildung 6.5 zeigt, dass auf diesen nicht nur der gesuchte Ball, sondern auch Objekte erfasst werden, die welche nicht erkannt werden sollen. Bei diesen handelt es sich

einerseits um Bälle anderer Farbe und andererseits um weitere Gegenstände, wie ein Karton aus Pappe oder ein Feuerlöscher. In diesem Abschnitt werden zunächst die verwendeten Bewertungsmethoden erläutert, bevor diese in Abschnitt 6.3 für die Analyse der acht Umsetzungstabellen herangezogen werden. Für alle Bilder werden folgende

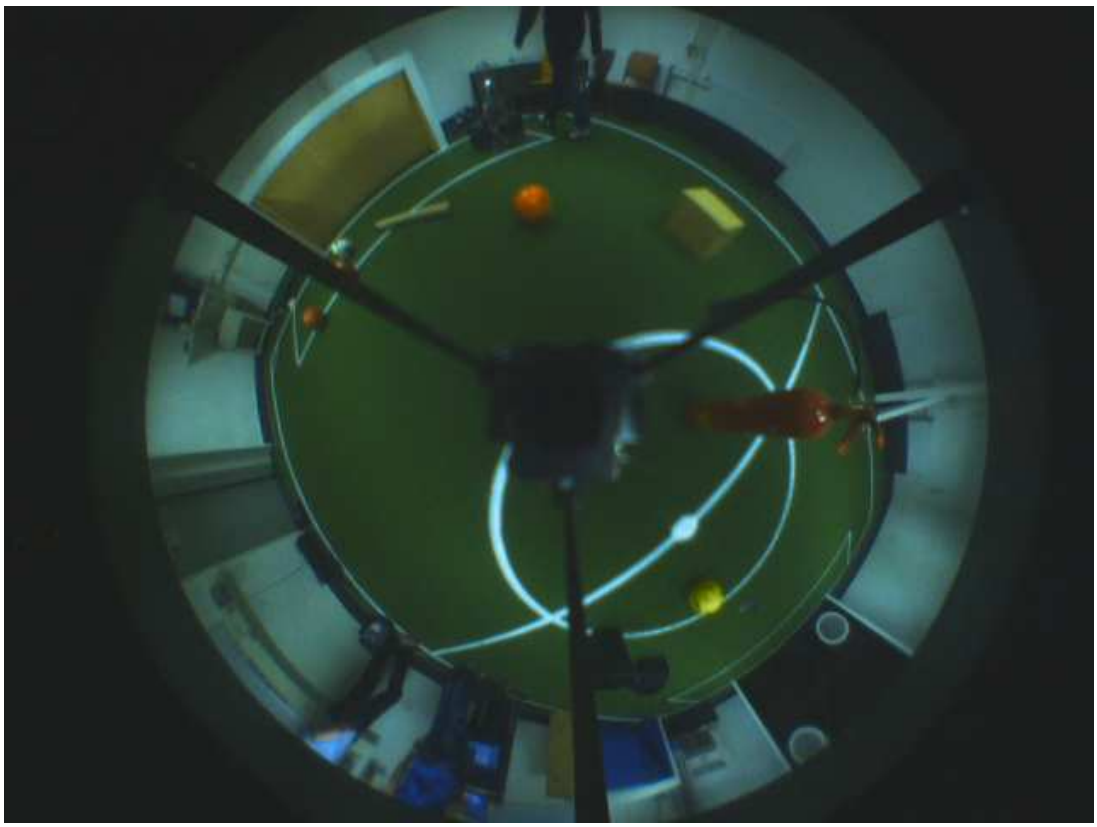


Abbildung 6.5.: Auf den Testbildern befinden sich alle realitätsnahen Objekte, Bälle, Tore, Spielfeldboden und -begrenzung, Roboter, Menschen und anderer mögliche Störfaktoren.

Werte ermittelt:

Erkannt - Ballpunkt $|B \cap E|$ Alle korrekt erkannten Ballpunkte.

Erkannt - Kein Ballpunkt $|\neg B \cap E|$ Alle Punkte, die fälschlicherweise als Ballpunkte erkannt werden.

Nicht Erkannt - Ballpunkt $|B \cap \neg E|$ Alle nicht erkannten Ballpunkte.

Nicht Erkannt - Kein Ballpunkt $|\neg B \cap \neg E|$ Alle Punkte, die nicht als Ballpunkte erkannt wurden und keine Ballpunkte sind.

B : Ballpunkte	$\neg B$: keine Ballpunkte
E : Als Ballpunkt erkannte Punkte	$\neg E$: Als „nicht-Ballpunkt“ erkannte Punkte
M : alle beobachteten Punkte	es gilt: $M = B \cup \neg B = E \cup \neg E$

Für die Erfassung dieser Daten wird ein rudimentäres Evaluations-Werkzeug genutzt. (siehe Abbildung D.2)

6.2.1. Wahrscheinlichkeitstabellen

Um die gesammelten Ergebnisse miteinander vergleichen zu können, werden zunächst Wahrscheinlichkeitstabellen für alle Umsetzungstabellen berechnet. Eine allgemeine Wahrscheinlichkeitstabelle ist in Tabelle 6.1 zu sehen. Die Wahrscheinlichkeiten werden dabei mit den relativen Häufigkeiten abgeschätzt. Um diese zu bestimmen wird der Anteil der gemessenen Werte von allen beobachteten Punkten M ermittelt,

$$\begin{aligned} P(B \cap E) &= \frac{|B \cap E|}{|M|}, \\ P(\neg B \cap E) &= \frac{|\neg B \cap E|}{|M|}, \\ P(B \cap \neg E) &= \frac{|B \cap \neg E|}{|M|}, \\ P(\neg B \cap \neg E) &= \frac{|\neg B \cap \neg E|}{|M|}. \end{aligned} \tag{6.1}$$

	B	¬B
E	$P(B \cap E)$	$P(\neg B \cap E)$
¬E	$P(B \cap \neg E)$	$P(\neg B \cap \neg E)$

Tabelle 6.1.: Allgemeine Wahrscheinlichkeitstabelle

Die vollständigen Wahrscheinlichkeitstabellen sind in Tabelle 6.2 zu sehen.

6.2.2. Bewertungsmethoden für Klassifikatoren

Um die Umsetzungstabellen näher vergleichen zu können, werden Bewertungsmethoden für Klassifikatoren verwendet [14]. Die Sensitivität gibt den Anteil der richtig erkannten Punkte $B \cap E$ von allen Ballpunkten B und somit die Trefferquote an,

$$\text{Sensitivität} = \frac{P(B \cap E)}{P(B)}. \tag{6.2}$$

Ursprüngliche Umsetzungstabelle

	B	¬B
E	0,066 %	4,780 %
¬E	0,007 %	95,148 %

Grundlage

	B	¬B
E	0,038 %	1,226 %
¬E	0,035 %	98,701 %

EM-Algorithmus 1

	B	¬B
E	0,072 %	21,232 %
¬E	0,001 %	78,695 %

Bereinigung 1

	B	¬B
E	0,031 %	0,046 %
¬E	0,042 %	99,881 %

EM-Algorithmus 2

	B	¬B
E	0,045 %	0,251 %
¬E	0,028 %	99,676 %

Bereinigung 2

	B	¬B
E	0,058 %	0,525 %
¬E	0,015 %	99,402 %

EM-Algorithmus 3

	B	¬B
E	0,045 %	0,199 %
¬E	0,028 %	99.728 %

Bereinigung 3

	B	¬B
E	0,051 %	0,267 %
¬E	0,022 %	99,660 %

Tabelle 6.2.: Wahrscheinlichkeitstabellen für alle betrachteten Umsetzungstabellen

Die Spezifität definiert den Anteil der korrekt als nicht-Ballpunkte erkannten Punkte $\neg B \cap \neg E$ zu allen nicht-Ballpunkten $\neg B$,

$$\text{Spezifität} = \frac{P(\neg B \cap \neg E)}{P(\neg B)}. \tag{6.3}$$

Der Anteil der richtig erkannten Ballpunkte $B \cap E$ von allen erkannten Ballpunkten E wird als Genauigkeit beschrieben. Eine andere Bezeichnung ist der positive Vorhersagewert,

$$\text{Genauigkeit} = \frac{P(B \cap E)}{P(E)}. \tag{6.4}$$

Die Korrektklassifikationsrate gibt den Anteil aller korrekt Klassifizierten Punkte zu allen Punkten an,

$$\text{Korrektklassifikationsrate} = \frac{P(B \cap E) + P(\neg B \cap \neg E)}{P(B \cap E) + P(\neg B \cap E) + P(B \cap \neg E) + P(\neg B \cap \neg E)}. \tag{6.5}$$

In Tabelle 6.3 sind alle Werte für die acht Umsetzungstabellen aufgeführt.

	Sensitivität	Spezifität	Genauigkeit	Korrekt.
Ursprüngliche	90,909 %	95,217 %	1,369 %	95,214 %
Grundlage	52,042 %	98,774 %	3,006 %	98,739 %
EM-Algorithmus 1	98,419 %	78,752 %	0,337 %	78,767 %
EM-Algorithmus 2	61,660 %	99,749 %	15,220 %	99,721 %
EM-Algorithmus 3	61,528 %	99,801 %	18,400 %	99,773 %
Bereinigung 1	42,688 %	99,954 %	40,424 %	99,912 %
Bereinigung 2	79,513 %	99,474 %	9,947 %	99,460 %
Bereinigung 3	70,158 %	99,732 %	16,073 %	99,711 %

Tabelle 6.3.: Ergebnisse der Bewertungsmethoden

6.2.3. Effektivitätsmaß

Mit Hilfe des Effektivitätsmaß E werden die Sensitivität und die Genauigkeit zueinander in Beziehung gesetzt [15]. Damit liefert E ein weiteres Gütemaß, anhand dessen die Umsetzungstabellen verglichen werden können,

$$E = 1 - \frac{1}{\alpha(\frac{1}{\text{Genauigkeit}}) + (1-\alpha)(\frac{1}{\text{Sensitivität}})}. \tag{6.6}$$

Das Effektivitätsmaßes E enthält den Faktor α. Dieser bietet die Möglichkeit, die Sensitivität und Genauigkeit zu gewichten. Dafür können Werte zwischen 0 und 1 für α gewählt werden, wobei $\alpha = 0,5$ die beiden Werte gleich gewichtet. Das Effektivitätsmaß E liegt zwischen 0 und 1, wobei 0 für eine gute und 1 für eine schlechte Effektivität steht. Für die Objekterkennung des *Carpe Noctem*-Frameworks, die für das Finden des Balls zuständig ist, spielt die Sensitivität eine größere Rolle, weil die fälschlicherweise als Ballpunkte erkannten Punkte in der Regel herausfallen. Wird die Ballfarbe hingegen gar nicht erkannt, kann die Objekterkennung keine Objekte finden und somit auch keinen Ball wahrnehmen. Aus diesem Grund wird in Abbildung 6.6 der Verlauf der Effektivität, im Bereich von $\alpha = 0,9$ bis $\alpha = 0,1$, betrachtet.

6.3. Analyse

Die Evaluation dient dazu, den Nutzen des entwickelten Werkzeuges festzustellen. Es gibt zwei Punkte, deren Prüfung sinnvoll ist. Erstens wird ein Vergleich zwischen der ursprünglichen Umsetzungstabelle und den mit dem Werkzeug erstellten Umsetzungstabellen in Abschnitt 6.3.1 gezogen und zweitens findet die Bewertung der Interpolation durch den EM-Algorithmus in Abschnitt 6.3.2 statt.

In den Wahrscheinlichkeitstabellen in Tabelle 6.2 gibt es drei Auffälligkeiten: Die

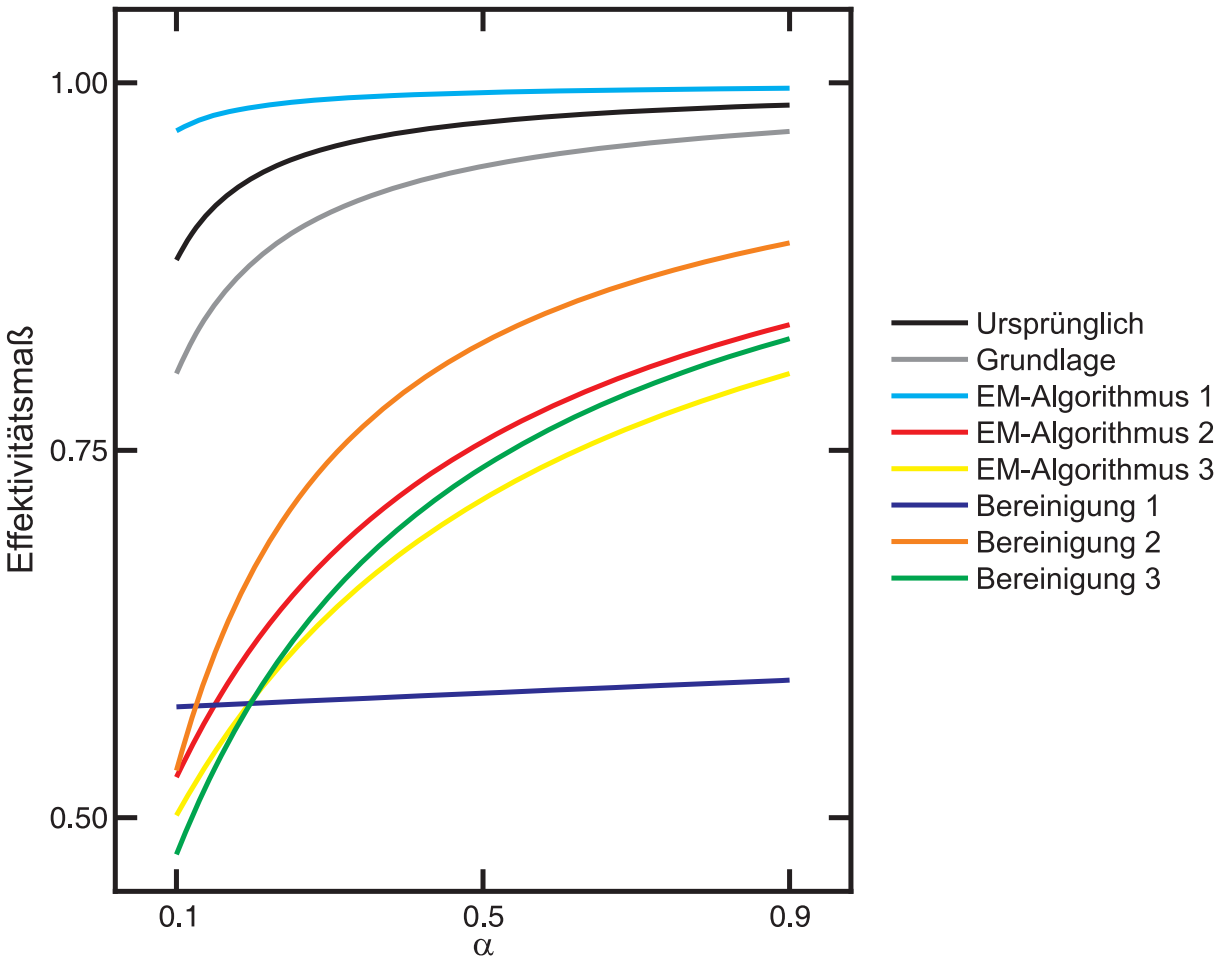


Abbildung 6.6.: Das Effektivitätsmaß wird für alle Umsetzungstabellen mit einer variierenden Gewichtung α aufgetragen. Durch ihren Linienverlauf lassen sich die Umsetzungstabellen in drei Gruppen einteilen.

Ergebnisse der mit dem CoCa erstellten Umsetzungstabellen weichen stark voneinander ab. Zum Beispiel haben Tabellen *EM-Algorithmus 1* und *Bereinigung 1* eine Wahrscheinlichkeit $P(B \cap E)$ von $0,072\%$ und $0,031\,\%$ und $P(\neg B \cap \neg E)$ von $78,695\,\%$ und $99,881\,\%$. Mit einer Wahrscheinlichkeit $P(B \cap E)$ von $0,066\,\%$, dass ein Ballpunkt erkannt wird, liefert die ursprüngliche Umsetzungstabelle ein relativ gutes Ergebnis, da im Mittel $0,073\,\%$ der Punkte eines Bildes Ballpunkte sind. Das liegt daran, dass bei dieser ausgenutzt wird, dass fälschlicherweise erkannte Punkte herausgefiltert werden können; nicht erkannt Ballpunkte hingegen eine verlorene Information bedeuten. Fünf der sieben Umsetzungstabellen, welche mit dem Werkzeug erstellt wurden, haben die relative Häufigkeit $P(\neg B \cap \neg E)$, von über $99\,\%$, dass ein nicht als Ballfarbe erkannter Punkt auch kein Ballpunkt ist.

Bei der Betrachtung des Effektivitätsmaßes in Abbildung 6.6 fällt zunächst auf, dass alle Verläufe abfallend sind. Dies bedeutet, dass alle Umsetzungstabellen eine größere Trefferquote als Genauigkeit haben. Dieser Umstand ist auch in Tabelle 6.3 ersichtlich. Es fällt weiterhin auf, dass die Effektivität der *Bereinigung 1*-Tabelle für alle α-Werte verhältnismäßig gut ist, allerdings hat diese Tabelle die niedrigste Sensitivität. Das liegt daran, dass das Effektivitätsmaß die Ausgewogenheit zwischen Sensitivität und Genauigkeit begünstigt. In Anbetracht der niedrigen Sensitivität der *Bereinigung 1*-Tabelle kann das Effektivitätsmaß alleine nicht für die Bewertung der Umsetzungstabellen herangezogen werden. Allerdings lassen sich diese mit Hilfe des Effektivitätsmaßes in drei Gruppen einteilen:

Die erste Gruppe besteht aus der ursprünglichen Umsetzungstabelle, der *Grundlagen*-Tabelle und der *EM-Algorithmus 1*. Diese Gruppe zeigt eine niedrige Effektivität. Bei sinkendem α ändert sich diese nur schwach. Das liegt daran, dass alle Tabellen einen Genauigkeitswert von unter $3,1\,\%$ haben. Dieser dominiert sogar die hohen Sensitivitäten der ursprünglichen Tabelle und der *EM-Algorithmus 1*-Tabelle.

Die zweite Gruppe beinhaltet die *EM-Algorithmus 2*-Tabelle, die *EM-Algorithmus 3*-Tabelle, die *Bereinigung 2*-Tabelle und die *Bereinigung 3*-Tabelle. Diese Tabellen haben gegen $\alpha = 0,1$ einen niedrigen Effektivitätswert; für $\alpha = 0,1$ sogar geringer als die *Bereinigung 1*-Tabelle. Die Umsetzungstabellen haben das ausgeglichenste Verhältnis von Sensitivität und Genauigkeit und gleichzeitig eine Sensitivität von über $60\,\%$.

Die letzte Umsetzungstabelle ist die *Bereinigung 1*-Tabelle. Diese hat mit einer nahezu konstanten Effektivität von circa $0,6$ für fast alle α-Werte die niedrigste Effektivität, allerdings hat sie eine Sensitivität von $42,688\,\%$. Die niedrige Sensitivität kommt daher, dass bei der Bereinigung zu viele Informationen verloren gehen. Ohne eine Interpolation mit dem EM-Algorithmus liefert diese Umsetzungstabelle ein für die Ballerkennung ungenügendes Ergebnis.

6.3.1. Vergleich der ursprünglichen mit den neuen Umsetzungstabellen

Wie bereits erwähnt, ist die Sensitivität in Tabelle 6.3 der wichtigste Wert. Aus diesem Grund hat die ursprüngliche Umsetzungstabelle für die Ballerkennung gut funktioniert. Lediglich die *EM-Algorithmus 1*-Tabelle hat eine höhere Sensitivität als die ursprüngliche Umsetzungstabelle. Diese Tabelle hat allerdings eine Genauigkeit von 0,337 %, was zur Folge hat, dass die erkannten Ballpunkte zu unzuverlässig sind. Auch die ursprüngliche Umsetzungstabelle hat mit 1,369 % eine relativ geringe Genauigkeit, was daran liegt, dass diese sowohl für gelbe als auch orangefarbene Bälle kalibriert ist. Aus diesem Grund ist die mangelnde Genauigkeit der ursprünglichen Umsetzungstabelle kein alleiniges Ausschlusskriterium. Das gilt allerdings für die *EM-Algorithmus 1*-Tabelle nicht, da diese nur für den gelben Testball kalibriert ist. Die letzte Umsetzungstabelle aus der Gruppe 1 ist die *Grundlagen*-Tabelle. Diese hat zwar eine höhere Genauigkeit als die vorherigen Tabellen, fällt allerdings durch die niedrige Sensitivität heraus. Die Ergebnisse der *Bereinigung 1*-Tabelle sind mit einer Sensitivität von 52,042 % zu gering, deshalb werden lediglich die Tabellen aus Gruppe 2 näher untersucht.

Die Umsetzungstabellen aus Gruppe 2 haben alle eine Sensitivität von über 60 %. Die Spezifität und Korrektklassifikationsrate aller vier Tabellen liegt über 99 %. Das bedeutet, dass die Ergebnisse, die diese Tabellen liefern, für die Ballfindung zuverlässig genug sind. Das bestätigt auch die Genauigkeit, welche mit Werten zwischen 9 und 19 %, deutlich über der ursprünglichen Umsetzungstabelle liegt. Beim Betrachten aller Testbilder und der entsprechenden ROI-Kanälen fällt auf, dass alle vier Umsetzungstabellen aus Gruppe 2 den Ball zuverlässig abbilden. Es fallen zwei Flächen auf, die von allen fälschlicherweise als Ballfarbe erkannt werden. Zum einen die Oberseite des Karton, in Abbildung 6.7 zu sehen, und zum anderen die Seite des holzfarbenen Tors in Abbildung 6.8. Die Farben dieser Flächen sind zum großen Teil identisch mit den Farben des Ball. Diese falsch erkannten Punkte spielen bei der farblichen Segmentierung keine Rolle, da sie nicht vermieden werden können. In Abbildung 6.9 ist ein entscheidender Vorteil der mit CoCa erstellten Umsetzungstabellen. Die ursprüngliche Umsetzungstabelle klassifiziert nahezu alle Hautfarben als Ballfarben.

6.3.2. Bewertung der Interpolation

Für die Bewertung der Interpolation werden, da der EM-Algorithmus zweimal angewendet wurde, vier Umsetzungstabellen herangezogen. Es werden jeweils die vorherigen mit den resultierenden Umsetzungstabellen verglichen. Bei dem Vergleich der Werte aus Tabelle 6.3 fällt auf, dass sich die Güte der Resultate unterscheidet. Im Fol-

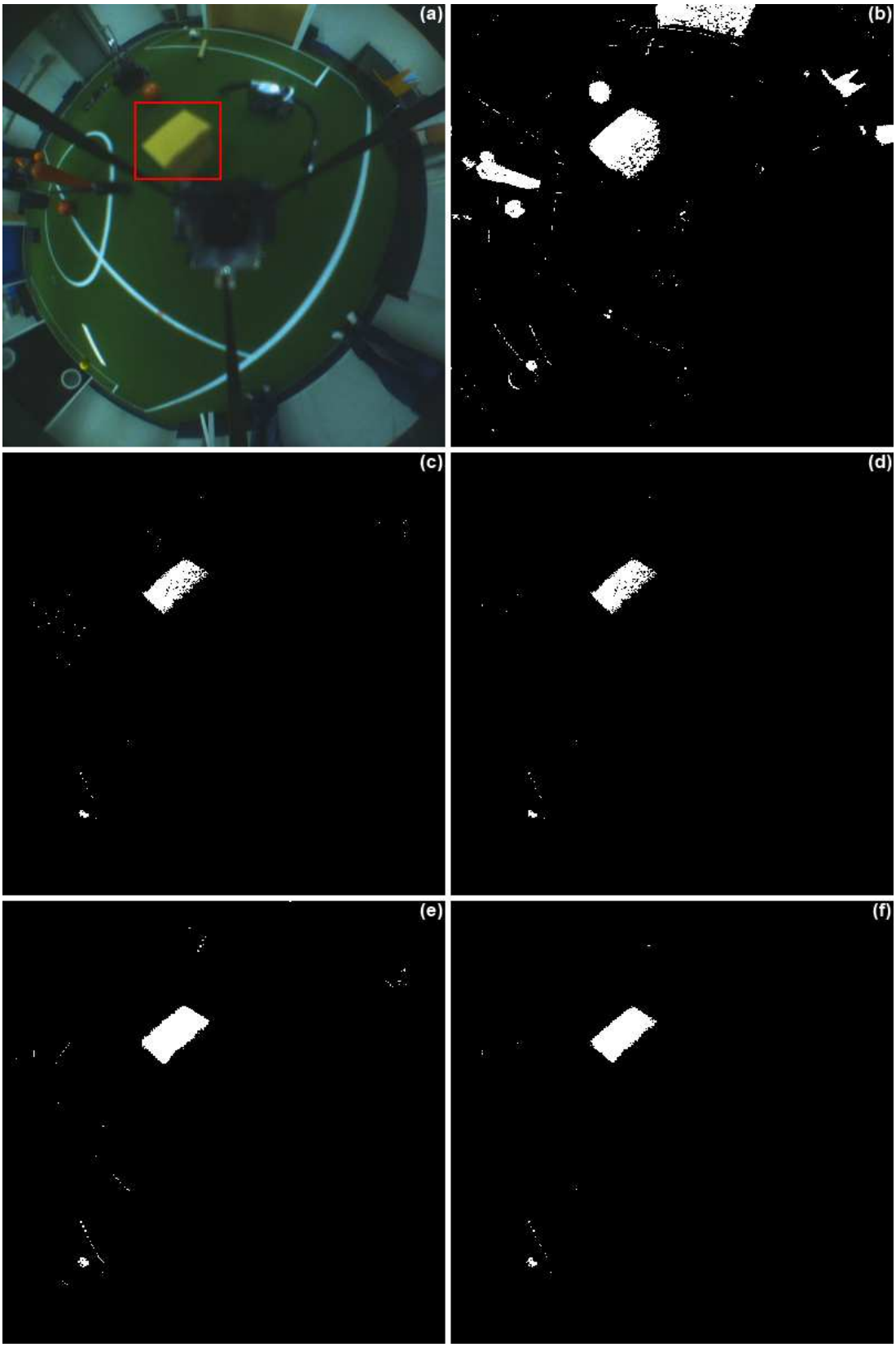


Abbildung 6.7.: Segmentierungsproblem: Karton. *(a)* Ausgangsbild mit markierem Problembereich. *(b)* Ursprüngliche Umsetzungstabelle, *(c) EM-Algorithmus 2*-Tabelle, *(d) EM-Algorithmus 3*-Tabelle, *(e) Bereinigung 2*-Tabelle und *(f) Bereinigung 3* erkennen im Karton Ballfarben.

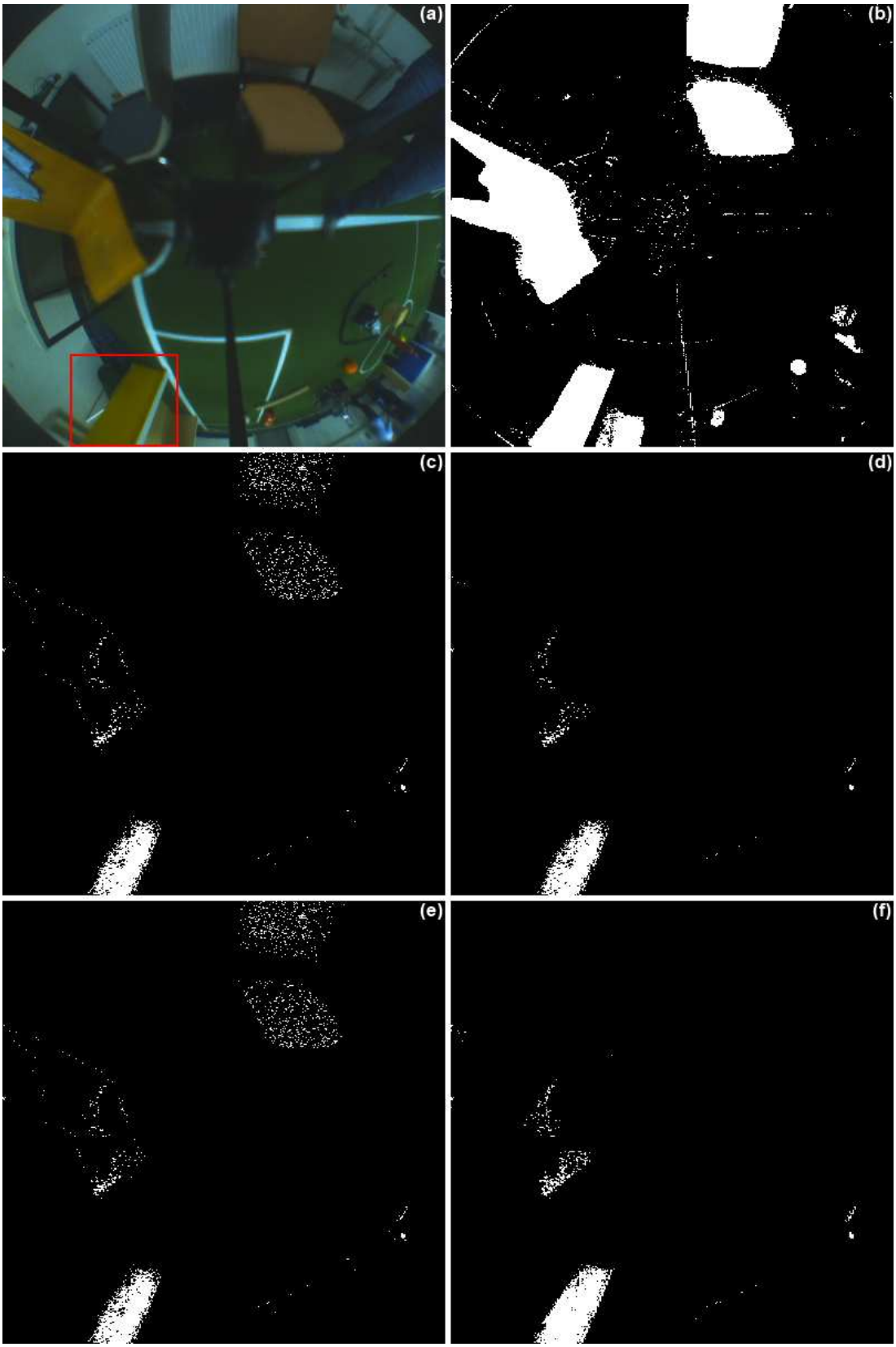


Abbildung 6.8.: Segmentierungsproblem: Torseite. *(a)* Ausgangsbild mit markierem Problembereich. *(b)* Ursprüngliche Umsetzungstabelle, *(c) EM-Algorithmus 2*-Tabelle, *(d) EM-Algorithmus 3*-Tabelle, *(e) Bereinigung 2*-Tabelle und *(f) Bereinigung 3* erkennen in der Torseite Ballfarben.

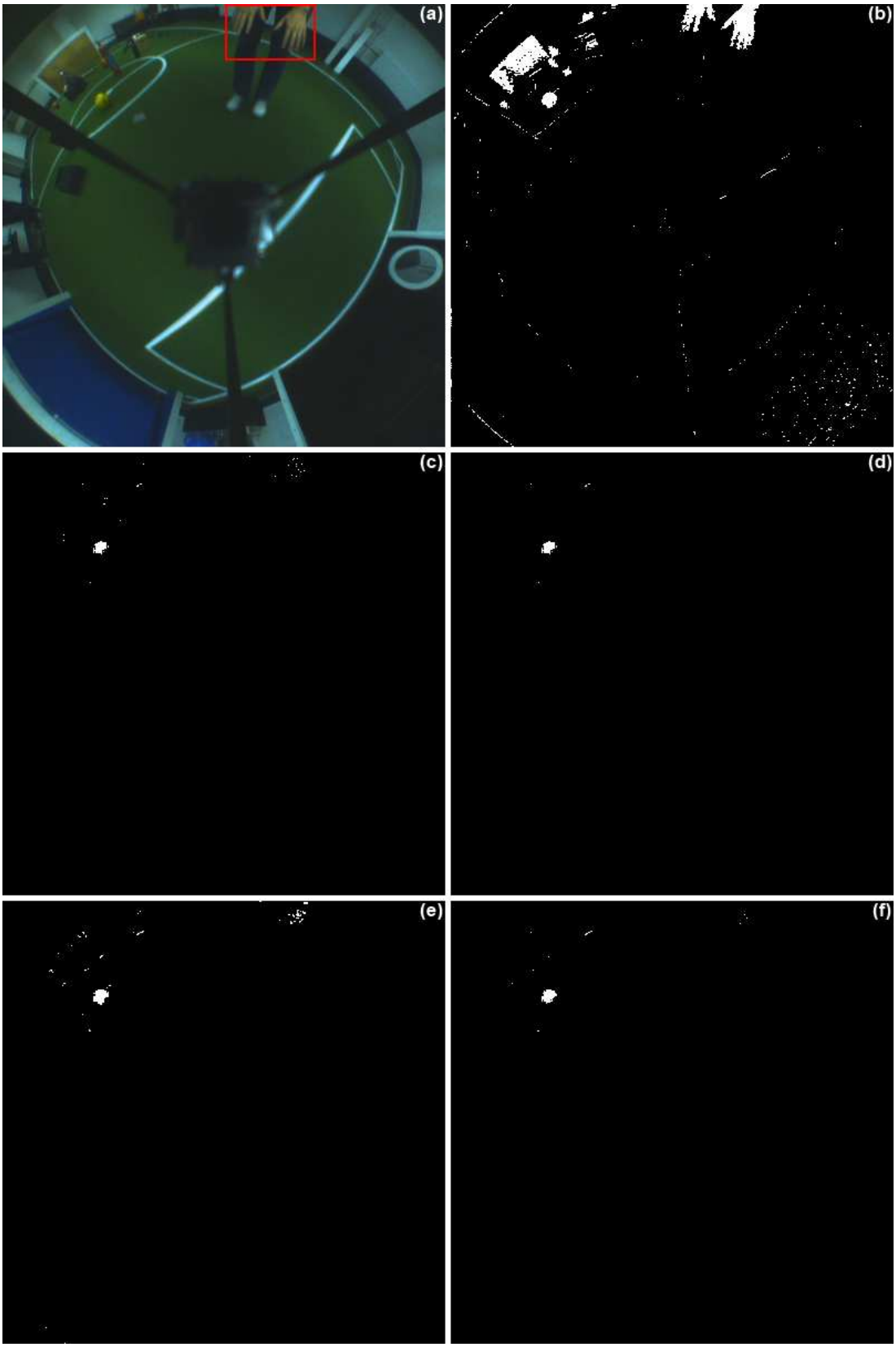


Abbildung 6.9.: Segmentierungsproblem: Hände. *(a)* Ausgangsbild mit markierem Problembereich. *(b)* Die ursprüngliche Umsetzungstabelle erkennt in den Händen Ballfarben, hingegen erkennen die *(c) EM-Algorithmus 2*-Tabelle, *(d) EM-Algorithmus 3*-Tabelle, *(e) Bereinigung 2*-Tabelle und *(f) Bereinigung 3* in den Händen keine Ballfarbe.

genden werden beide Varianten analysiert.

Bei der Anwendung auf die *Grundlagen*-Tabelle steigt die Sensitivität von 52,042 % auf 98,419 %. Das lässt zunächst eine Verbesserung des Ergebnisses vermuten, allerdings sind die Spezifität und die Korrektklassifikationsrate von circa 98 % auf 78 % gesunken. Diese beiden Werte geben Aufschluss darüber, wie zuverlässig die Umsetzungstabellen sind. Bei der Betrachtung der beiden Umsetzungstabellen in Abbildung 6.3 fällt auf, dass beide grüne Farbwerte enthalten. Das hat zur Folge, dass in Teilen der Spielfeldboden als Ballfarbe klassifiziert wird. Dieses Ergebnis aus der Verwendung des EM-Algorithmus ist für die weiteren Verarbeitungsschritte der Ballerkennung nicht ausreichend. Aus diesem Grund sind weitere Verarbeitungsschritte notwendig.

Die Anwendung des EM-Algorithmus auf die *Bereinigung 1*-Tabelle hat eine Steigerung der Sensitivität von 42,688 % auf 79,513 % zur Folge. Dies ist zwar auch bei der vorherigen Anwendung der Fall, allerdings sind die Spezifität und Korrektklassifikationsrate nahezu konstant bei circa 99 % geblieben. Diese Ergebnis ist für die weiteren Verarbeitungsschritte der Ballerkennung ausreichend. Dies bestätigen auch die Ergebnisse in Abbildung 6.7, 6.8 und 6.9.

6.4. Fazit

Im Laufe der Evaluation hat sich gezeigt, dass es mit CoCa möglich ist, eine spezielle Ballfarbe zu erkennen. Allerdings hat besonders die Bewertung der Interpolation in Abschnitt 6.3.2 gezeigt, dass die Vorgehensweise bei der Erstellung der Umsetzungstabelle von Bedeutung ist. Obwohl die *EM-Algorithmus 1*-Tabelle unzureichende Ergebnisse liefert, führt die Bereinigung der Tabelle zu einem ausreichenden Ergebnis, wie in der Analyse in Abschnitt 6.3.1 zu sehen ist. Die Ergebnisse der mit CoCa erstellten Umsetzungstabellen in Tabelle 6.2 und Tabelle 6.3 weichen erheblich voneinander ab. Das lässt vermuten, dass bei steigender Erfahrung im Umgang mit dem CoCa, noch bessere Ergebnisse erzielt werden können. Bei der Evaluation wurden lediglich die Umsetzungstabellen miteinander verglichen. Zusätzlich ist es durch den Einsatz des CoCa möglich, mit einer geringen Einarbeitungszeit schnell zu einem für die Ballfindung ausreichenden Segmentierungsergebnis zu kommen.

Kapitel 7

Verwandte Arbeiten

Die farbliche Segmentierung von Bildern und die Kalibrierung von Kameraparametern ist ein wichtiges Thema im Bereich der Bildverarbeitung. In diesem Kapitel wird über einige Arbeiten, die sich mit diesem Thema befassen, eine Übersicht gegeben.

In [16] wird die automatische Kalibrierung einer Umsetzungstabelle für das Erkennen von Gesichtsfarbe vorgenommen. Dabei wird das zu erkennende Gesicht in einer Umrandung eingegrenzt, damit bei der Klassifizierung einfacher zwischen Gesicht und Hintergrund unterschieden werden kann. Dieses Vorgehen ist mit der Markierung des Balles in dieser Arbeit vergleichbar. Es werden ebenfalls lediglich die Farbinformationen für die Klassifizierung verwendet. Die Helligkeitswerte werden nicht betrachtet. Es werden zwei Kalibrierungsverfahren vorgestellt; eins unter Verwendung einer Clusteranalyse und eins zweites mit Hilfe einer Histogrammanalyse. Für die Clusteranalyse wird der *Fuzzy C-Means Algorithmus* verwendet. Dieser nutzt Clusterzentren für die Positionierung von Clustern. Die zu klassifizierenden Punkte werden mit einer entsprechenden Gewichtung zugeordnet. Bei der Histogrammanalyse werden die Punkte anhand von Auftrittswahrscheinlichkeiten eingeteilt. Es wird festgestellt, dass die Kalibrierung mit Hilfe der Histogrammanalyse mit circa 6 Sekunden um 8 Sekunden schneller ist als die Clusteranalyse. Außerdem wurde eine Fehlerrate von circa 17 % gemessen.

In der RoboCup-Domäne existieren viele Arbeiten zu diesem Thema. Eine Möglichkeit zur automatischen Anpassung eines Segmentierungsfilters wird in [17] beschrieben. Dabei werden zunächst aus den im Bild enthaltenen Farbwerten Histogramme abgeleitet. Auch hier wird der YUV-Farbraum verwendet, wobei die Luminanz vernachlässigt wird. In diesen Histogrammen werden die Maxima mit Hilfe des Hill-Climbing-Verfahrens ermittelt. Aus den Maxima werden Cluster gebildet, in denen die restlichen Farbwerte den am nächstliegenden Maxima zugeordnet werden. Schlussendlich werden die Cluster unter Verwendung eines Schwellwertes einer gesuchten Farbe zu-

geordnet und zusammengefasst. Alle Cluster, die nicht zugeordnet werden können, werden verworfen. Aus den verbleibenden Farbwerten wird eine Umsetzungstabelle erstellt. Da die gesuchten Farben vorher festgelegt wurden, kann ein Farbwert nur einem Cluster zugeordnet werden. Im Gegesatz dazu können bei dem in dieser Arbeit verwendeten EM-Algorithmus die Punkte in mehreren Clustern auftreten.

In [18] wird der *Automatic Color Training Algorithmus* (ATC-Algorithmus) vorgestellt. Dieser soll die Grundlage für die automatisierte Kalibrierung einer Umsetzungstabelle bieten. Mit Hilfe des ATC-Algorithmus ist es den Robotern möglich, sich an wechselnde Lichtverhältnisse anzupassen. Bei dem ATC-Algorithmus werden die Positionsdaten des Roboters genutzt, um lediglich die im Spielfeld befindlichen Pixel zu betrachten. Die auftretenden Farben werden in Cluster eingeteilt. Da auf jedem Bild die Farben Grün und Weiß auftreten, werden zu Beginn alle im Spielfeld befindlichen Punkte als eine dieser Farben klassifiziert. Für die Farbe des Balles bedarf es einer manuellen Initialisierung, da diese variiert. Bei jeder Iteration des ATC-Algorithmus wird eine bestimmte Anzahl an Pixeln des aktuellen Bildes klassifiziert. Zu diesem Zweck muss die euklidische Distanz zwischen dem aktuellen Farbwert und den Mittelwerten aller Farbwerte der Cluster unter einem bestimmten Schwellwert liegen. Nicht klassifizierte Farben werden aus der Umsetzungstabelle entfernt, so wird diese konstant bereinigt. Der ATC-Algorithmus wird erfolgreich bei einem *Middle Size League*-Roboter eingesetzt.

In [19] werden zwei Umsetzungstabellen für zwei unterschiedliche Farbräume eingesetzt um ähnliche Farben, wie einen orangefarbenen Ball und ein gelbes Tor besser voneinander unterscheiden zu können. Es werden ebenfalls der YUV- und zusätzlich der HSI-Farbraum verwendet. Bei diesem werden der Farbwert H und Farbsättigung S für die Segmentierung genutzt. Die Lichtintensität I wird vernachlässigt. Um die Initialisierung der Umsetzungstabellen zu erleichtern, wird eine lineare Klassifikation gebraucht. Dieses Verfahren wird erfolgreich für die Segmentierung mit einem Fußballroboter verwendet. Dabei wurde gemessen, dass die Segmentierung eine Bildes lediglich 3 ms benötigt.

Kapitel 8

Zusammenfassung und Ausblick

In dem abschließenden Kapitel folgt zunächst eine Zusammenfassung der aus dieser Arbeit entstandenen Ergebnisse. Anschließend werden zukünftige Arbeiten, welche auf dieser Arbeit aufbauen könnten, thematisiert.

8.1. Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Kalibrierung einer farblichen Segmentierung von Bildern. Dieses Problem wurde in der Domäne des RoboCup beleuchtet. Dort wird die farbliche Segmentierung von Bildern für die Ballerkennung verwendet. Für die Segmentierung wird eine Umsetzungstabelle genutzt, welche für alle Farbwerte die Information enthält, ob diese relevant sind. Ziel dieser Arbeit ist es, die Kalibrierung dieser Umsetzungstabelle zu erleichtern. Diese ist bisher nur in einem ungenauen Maß und mit hohem zeitlichen Aufwand möglich. Zu diesem Zweck wurde das Softwarewerkzeug *ColorCalibrator* entwickelt, welches die Kalibrierung der Umsetzungstabelle anhand von aktuellen Kamerabildern der Roboter ermöglicht. Der ColorCalibrator arbeitet direkt mit dem Roboterframework zusammen und kann somit auf dessen Funktionalitäten zugreifen. Die Bilder werden von den Robotern direkt an den *ColorCalibrator* übertragen. Dies ermöglicht die Ballfarbe direkt in die Umsetzungstabelle hinzuzufügen. In der Umsetzungstabelle treten häufig Ballungsgebiete auf diese gilt es zu erkennen und mit Hilfe des EM-Algorithmus zu komplettieren. Zusätzlich bietet der *ColorCalibrator* das Speichern und Laden der Bilder aus einer Datei, um spätere Anpassungen und Fehleranalysen zu ermöglichen.

Für eine fehlerfreie Segmentierung ist eine korrekt kalibrierte Kamera des Roboters unverzichtbar. Deshalb bietet der *ColorCalibrator* die Möglichkeit, die Parameter der Kamera zur Laufzeit der Bildverarbeitung anzupassen. Alle Roboter können folglich separat konfiguriert werden, da für jede Kamera in der Regel andere Einstellungen

nötig sind, um ein optimales Bild zu erhalten.

Der *ColorCalibrator* wird mit Hilfe von Testbildern, in einer originalgetreuen Umgebung getestet. Die Güte der mit dem *ColorCalibrator* erzeugten Umsetzungstabellen wird mit der herkömmlichen Methode verglichen. Dabei wird festgestellt, dass sich die Anzahl der falsch erkannten Punkte verringern lässt. Zusätzlich wird die Notwendigkeit der Interpolation gezeigt.

8.2. Ausblick

In diesem Abschnitt werden mögliche zukünftige Arbeiten beschrieben, welche einerseits die Weiterentwicklung des *ColorCalibrators* beinhalten und andererseits auf diesem aufbauen könnten.

8.2.1. Interpolation mit anderen Clusteranalysen

Der *ColorCalibrator* interpoliert die Farbwerte in der Umsetzungstabelle mit dem EM-Algorithmus, andere Verfahren zur Clusteranalyse wären allerdings ebenfalls denkbar. Zum Beispiel könnte ein dichtebasiertes Clustering verwendet werden. Bei diesem werden die Positionen der Cluster nicht durch Mittelwerte aller enthaltenen Punkte definiert, sondern durch deren Menge. Die zu klassifizierenden Punkte müssen von einer bestimmten Anzahl von Punkten aus dem Cluster in einem Mindestabstand liegen, damit sie zu dem jeweiligen Cluster zugeordnet werden. Im Gegensatz zu dem EM-Algorithmus ist die Anzahl der Cluster nicht beschränkt. Außerdem sind einige dichtebasierte Clusteringverfahren nicht von einer Initialisierung abhängig.

8.2.2. Farbliche Segmentierung anderer Objekte

Mit Hilfe des *ColorCalibrator* können schnell und einfach Umsetzungstabellen für bestimmte Objektfarben erstellt werden. Bisher liegt lediglich der Ballerkennung eine farbliche Segmentierung zu Grunde. Es könnten weitere Umsetzungstabellen verwendet werden, um zum Beispiel die Positionen von Gegen- oder Mitspielern zu bestimmen, da jedes Team eine Art Trikotfarbe hat.

8.2.3. Automatische Kamerakalibrierung

Die Kameraparameter werden derzeit per Hand eingestellt. Es ist allerdings auch denkbar, diese Kalibrierung automatisch durchzuführen. Dabei könnten Kamerabilder beispielsweise mit den Bildern eines zuvor optimal kalibrierten Roboters verglichen wer-

den. Alternativ könnten für diesen Zweck möglicherweise lokal gespeicherte Bilder genutzt werden.

8.2.4. Kalibrierungsroutine

Bei der Erstellung der Kalibrierungsbilder für den *ColorCalibrator* werden derzeit die betreffenden Roboter mit der Hand an die gewünschten Positionen auf dem Spielfeld geschoben. Es könnte eine Routine implementiert werden, bei der der Roboter das Spielfeld durchquert und in einem bestimmten Intervall oder an bestimmten Positionen Kamerabilder an den *ColorCalibrator* sendet. Es ist auch denkbar, dass diese Routine vorher in einer grafischen Darstellung des Spielfeldes definiert wird. In dieser kann der Benutzer die gewünschte Route und Positionen, an denen ein Bild übermittelt werden soll, festlegen.

Mit diesen Erweiterungen könnte man in Zukunft die Erkennung des Balles und die Unterscheidung von anderen Objekten im Sichtfeld noch weiter optimieren.

Anhang A

Ursprüngliche Umsetzungstabelle

Bisher wurde die Umsetzungstabelle direkt im Programmcode des Frameworks initialisiert. Deshalb musste bei der Kalibrierung der Bildverarbeitungsprozess bei jeder Änderung neugestartet werden. Die gesuchten Farben wurden durch einen Winkel und eine Richtung im YUV-Farbraum charakterisiert.

```
void FilterYUVExtractSubImages::init(){
  int center = 128;
  double angle = atan2(150 - center, 70 - center);

  for(int u = 0; u < 256; u++){
    for(int v = 0; v < 256; v ++){
      int value = (int) lrint(((cos(angle)*(u-center) + sin(angle)*(v-
    center)) + 128)*2.5);
      double diffAngle = fabs(atan2(v-center, u-center) - angle);
      double distance = sqrt((v-center)*(v-center) + (u-center)*(u-center))
    /128.0;
      if(diffAngle > M_PI)
        diffAngle = fabs(diffAngle - 2.0*M_PI);
      value -= (int) lrint(7.5*diffAngle*180.0/M_PI*(0.3*distance + 1.0));
      int value2 = 0;
      if(value2 > value)
        value = value2;
      if(value < 0)
        value = 0;
      if(value > 255)
        value = 255;
      lookupTable[u*256 + v] = (unsigned char) value;
    }
  }
}
```

Listing A.1: Die ursprüngliche Initialisierung der Umsetzungstabelle

Anhang B

Robot Operating System

Als erster Ansatz werden die Bilder in einer ROS Nachricht versendet. Dazu werden die Bildinformationen in einem Integerarray gespeichert. Des Weiteren sollen die Roboter-ID, eine Identifikationsnummer und die Abmessungen des Bildes übertragen werden.

```
int32       senderID
uint32      imageID
uint8[]     imageData
int32       width
int32       height
```

Listing B.1: Erster Ansatz der Bildübertragung

Die zweite Variante verfolgt als Lösungansatz die Segmentierung der Bilder in mehrere ROS Nachrichten. Dies ist notwendig, da sich herausgestellt hat, dass die Nachrichten im ersten Ansatz zu groß sind um mit einer einzelnen ROS Nachricht versendet werden zu können. Zu diesem Zweck werden eine Identifikationsnummer für die Segmente und die Gesamtanzahl aller Segmente übertragen.

```
int32       senderID
uint32      imageID
uint8       segmentCount
uint8       segmentID
uint8[]     imageData
int32       width
int32       height
```

Listing B.2: Zweiter Ansatz der Bildübertragung mit Hilfe einer Segmentierung

Anhang C

Kamerakonfiguration

In den Robotern sind unterschiedliche Kameramodelle verbaut. Diese haben verschiedene Minimal- und Maximalwerte für ihre Parameter. Um den korreken Wertebereich in dem ColorCalibrator zur Verfügung zu haben, werden diese Werte in einer Konfigurationsdatei hinterlegt.

```
#
# Configuration for the Cameras

[CameraVendor]
  [Point]
    BrightnessMin = 0
    BrightnessMax = 1023
    ExposureMin = 1
    ExposureMax = 1023
    WhiteBalance1Min = 0
    WhiteBalance1Max = 1023
    WhiteBalance2Min = 0
    WhiteBalance2Max = 1023
    HueMin = 0
    HueMax = 4095
    SaturationMin = 0
    SaturationMax = 4095
    GammaMin = 512
    GammaMax = 4095
    ShutterMin = 0
    ShutterMax = 2388
    GainMin = 178
    GainMax = 860
  [!Point]
  [Imaging]
    BrightnessMin = 0
```

```
27      BrightnessMax = 255
        ExposureMin = 0
29      ExposureMax = 2000
        WhiteBalance1Min = 0
31      WhiteBalance1Max = 95
        WhiteBalance2Min = 0
33      WhiteBalance2Max = 95
        HueMin = 0
35      HueMax = 359
        SaturationMin = 0
37      SaturationMax = 255
        GammaMin = 10
39      GammaMax = 22
        ShutterMin = 1
41      ShutterMax = 4000
        GainMin = 180
43      GainMax = 1023
    [!Imaging]
45  [!CameraVendor]
```

Listing C.1: Die minimalen und maximalen Werte für die unterschiedlichen Parameter werden in einer Konfigurationsdatei festgehalten tabsize

Anhang D

Benutzeroberflächen

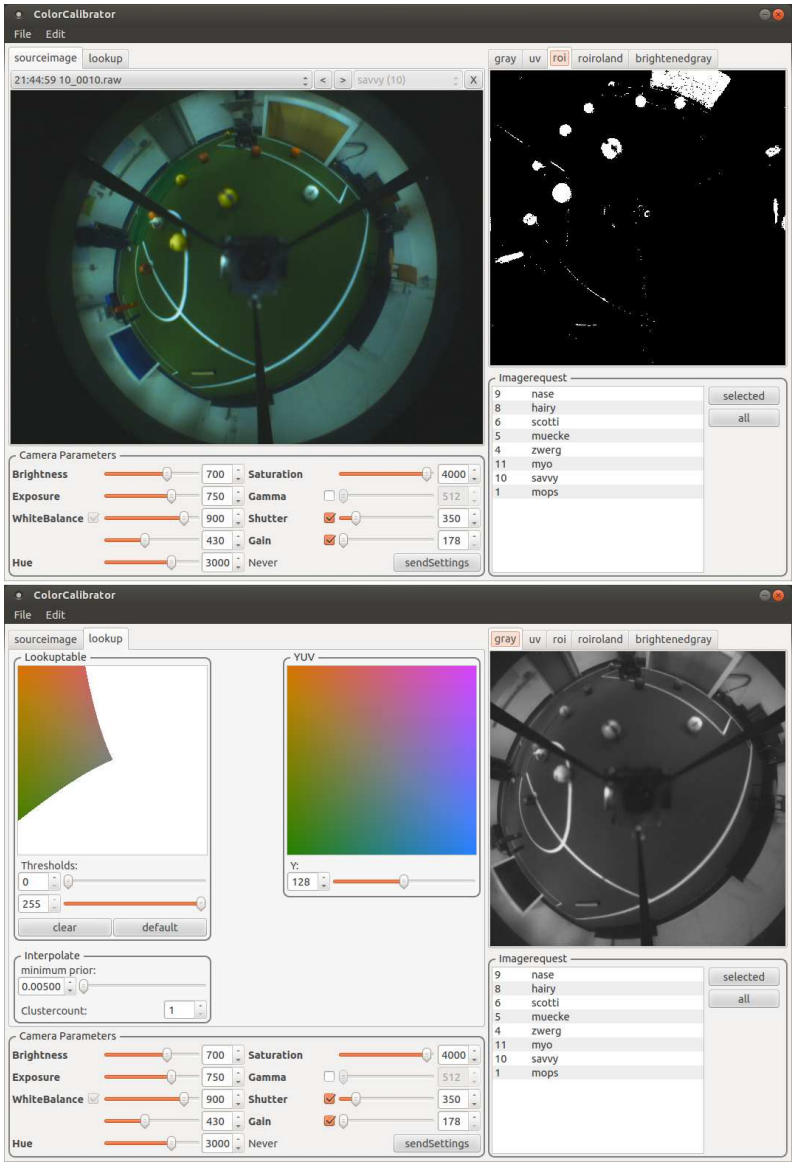


Abbildung D.1.: Benutzeroberfläche des entwickelten Werkzeugs

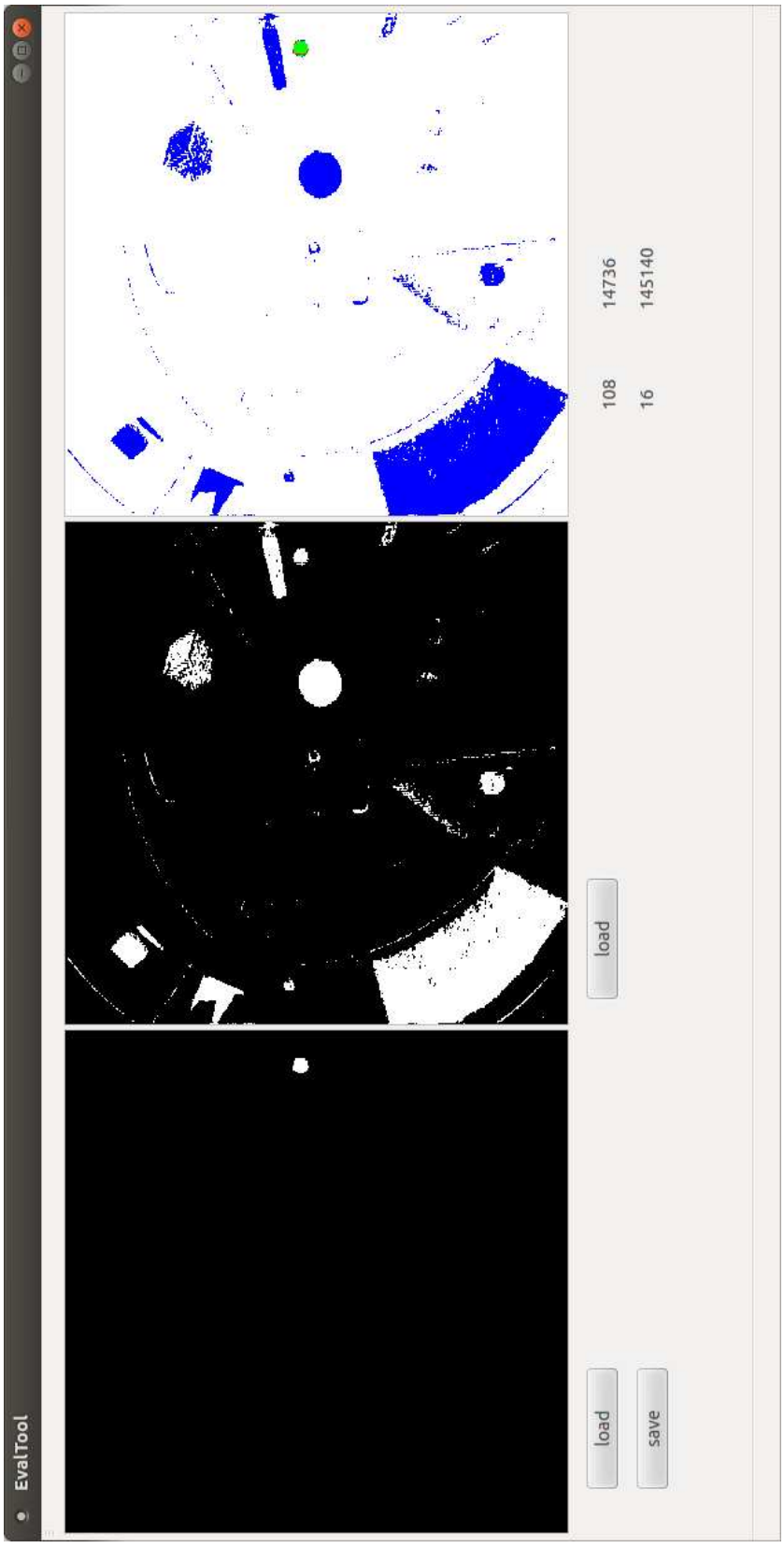


Abbildung D.2.: Werkzeug zur Evaluation

Literaturverzeichnis

[1] Robocup. `http://www.robocup.org` (Abgerufen am 09.08.2013).

[2] Amma, T., et al. (2013) Team Description: Carpe Noctem 2013. Tech. rep.

[3] Strutz, T. (2007) *Bilddatenkompression : Grundlagen, Codierung, JPEG, MPEG, Wavelets*. Programm Informationstechnik, Vieweg, 4., aktualisierte und erw. Aufl.

[4] Understanding YUV data formats. `http://www.ptgrey.com/support/kb/index.asp?a=4&q=313` (Abgerufen am 07.10.2013).

[5] Witsch, A. (2009), Formbasierte Ballerkennung im Robocup. Projektarbeit.

[6] Azad, P., Gockel, T., and Dillmann, R. (2007) *Computer Vision: Das Praxisbuch*. Elektor-Verlag Aachen.

[7] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009) ROS: an open-source Robot Operating System. *ICRA workshop on open source software*, vol. 3.

[8] ROS Wiki. `http://wiki.ros.org` (Abgerufen am 04.09.2013).

[9] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003) The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, **35**, 114–131.

[10] Ester, M. and Sander, J. (2000) *Knowledge Discovery in Databases*. Springer DE.

[11] Mahalanobis, P. C. (1936) On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, **2**, 49–55.

[12] Tönnies, K. (2005) *Grundlagen der Bildverarbeitung*. I, Informatik, Pearson Studium.

[13] Qt Project. `http://qt-project.org` (Abgerufen am 09.08.2013).

[14] Han, J. and Kamber, M. (2006) *Data mining : concepts and techniques*. The Morgan Kaufmann series in data management systems, Kaufmann, 2. edn.

[15] van Rijsbergen, C. (1979) *Information Retrieval*. Butterworth.

[16] Schröter, S. (1997) Automatic Calibration of Lookup-Tables for Color Image Segmentation. *3D Image Analysis and Synthesis (Proceedings), pp 123–129, Infix*, pp. 123–129.

[17] Hoppe, C. (2007), Adaptive Parametrisierung eines Segmentierungsfilters. Bachelorarbeit.

[18] Heinemann, P., Sehnke, F., Streichert, F., and Zell, A. (2007) Towards a calibration-free robot: The act algorithm for automatic online color training. *RoboCup 2006: Robot Soccer World Cup X*, pp. 363–370, Springer.

[19] Liu, F., Lu, H., and Zheng, Z. (2007) A modified color look-up table segmentation method for robot soccer. *Proceedings of the 4th IEEE LARS/COMRob*, **7**.