Fachgebiet Simulation, Systemoptimierung und Robotik
Fachbereich Informatik
Technische Universität Darmstadt

Automatische Farbklassifikation zur Anwendung im RoboCup

Automatic color classification for application in RoboCup

Diplomarbeit

von

Josef Baumgartner

Darmstadt, September 2008

Aufgabenstellung: Prof. Dr. Oskar von Stryk
Betreuer: Dipl.-Inform. Sebastian Petters

Erklärung zur Diplomarbeit

Hiermit versichere ich, dass die vorliegende Diplomarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt wurde. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, September 2008

_____

Josef Baumgartner

Kurzzusammenfassung

Das Fachgebiet Simulation, Systemoptimierung und Robotik der Technischen Universität Darmstadt ist mit seinen autonomen humanoiden Robotern schon seit mehreren Jahren erfolgreicher Teilnehmer bei den internationalen RoboCup-Fußballmeisterschaften. Für die autonome Arbeitsweise der Roboter müssen die farblich unterschiedlichen Objekte auf dem Feld, die von einer Kamera auf dem Kopf des Roboters erfasst werden, korrekt erkannt werden. Aufgrund der nach Standort unterschiedlichen Lichtverhältnissen, ist für die korrekte Farbzuordnung eine so genannte Farbtabelle nötig, die man bisher mit erheblichem Zeitaufwand manuell erstellen musste.

In dieser Arbeit wird eine Automatisierung der Erstellung von Farbtabellen vorgestellt. Anhand von Flächen- und Objekterkennung wird die dazu erforderliche automatische Farbklassifizierung durchgeführt. Die Berechnungen geschehen nicht auf dem internen Computer des Roboters, sondern auf einem externen Rechner zwecks besserer Performance und einfacher Überprüfbarkeit der automatisch erstellten Farbtabelle.

Abstract

The Simulation Systems Optimization and Robotics Group (SIM) of the Department of Computer Science of the Technische Universität Darmstadt is a successful member at the international RoboCup Soccer Championships for several years with its humanoid robots. For the autonomous function of the robots the differential colored objects on the field which are detected by a camera on the robot´s head must be identified correctly. Due to different lighting conditions depending on the location it is necessary to have a so called color table which had to be created manually with significant expenditure of time so far.

In this work an automatic creation of color tables will be introduced. On the basis of object and surface recognition the required automatic color classification is executed. The calculations are not done on the robot´s internal computer but on an external computer due to better performance and an easy way of verification of the automatic created color table.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Vorwort

1.1. Ziel der Arbeit

Ziel dieser Arbeit ist die Möglichkeit der Erstellung von Farbtabellen für den RoboCup mit Hilfe einer automatischen Farbklassifikation anhand von Kamerabildern.

Realisiert ist dies durch einen eigens entwickelten Dialog in der von Dirk Thomas und Sebastian Petters entwickelten GUI[8], die mit dem bereits vorhandenen COLORTABLE-Dialog kommuniziert.

Prioritäten waren vor allem einfache Bedienung, gute Performance, hohe Erkennungsraten und hohe Unempfindlichkeit gegenüber unbekannten Bildinformationen.

1.2. Aufbau der Arbeit

Kapitel 2 - Beweggründe für die Entwicklung einer Software für eine automatische Farbklassifizierung

Kapitel 3 - Beispiele für bereits existierende Ansätze

Kapitel 4 - Wichtige Grundlagen, die zum Verständnis der weiteren Ausführungen beitragen

Kapitel 5 - Das dahinterliegende Konzept dieser Arbeit

Kapitel 6 - Die Funktionsweise der einzelnen Objekterkenner, die für die automatische Farbklassifikation notwendig sind

Kapitel 7 - Zusätzliche Funktionen, die zur Verbesserung der Farbtabelle beitragen

Kapitel 8 - Hinweise zur praktischen Vorgehensweise der Erstellung einer Farbtabelle von der Aufnahme von Kamerabildern bis zur manuellen Korrektur der automatisch erstellten Farbtabelle

Kapitel 9 - Erkennraten und Performance

Kapitel 10 - Eine kurze Zusammenfassung dieser Arbeit und ein Ausblick auf zukünftige Erweiterungsmöglichkeiten

Anhang A - Beschreibung aller Funktionen des AUTOCOLORTABLE-Dialogs

Anhang B - Beschreibung aller Parameter des AUTOCOLORTABLE-Dialogs

Anhang C - Literaturverzeichnis

2. Motivation

Das Robotermodell HR18 sowie das neueste Modell HR30 des Fachgebiets Simulation, Systemoptimierung und Robotik der Technischen Universität Darmstadt sind mit einer in der Position regelbaren Kamera auf dem Kopf ausgestattet und für das autonome Fußballspielen konzipiert. Autonom bedeutet, dass die Entscheidungen alle vom Rechner getroffen werden, es darf nicht ferngesteuert werden. Somit muss die Kamera und die Software in der Lage sein, für das Fußballspiel relevante Objekte wahrzunehmen. Eine wichtige Anforderung des RoboCup ist, dass die Software bzw. die Berechnungen alle auf einem Computer, der lokal auf dem Roboter installiert ist, ausgeführt werden müssen. Sie dürfen nicht auf einem externen Rechner stattfinden.

Da der Roboter möglichst leicht und stromsparend sein sollte, muss auch der Computer auf dem Roboter dafür entsprechend ausgelegt sein. Hohe Rechenleistung ist in der Regel mit hohem Stromverbrauch und aufwendiger Kühlung verbunden, die auch wieder ein höheres Gewicht mit sich bringt. Da der Roboter aufgrund von Gewichtseinsparungen auch nur mit relativ geringer Akku-Kapazität ausgestattet ist (in der Größenordnung um 20 Wh), sollte die Leistung des Computers für eine ausreichende Laufzeit im einstelligen Watt-Bereich liegen.

Seit dem RoboCup 2007 wird ein PC104-Board mit AMD Geode-Prozessor, der auf 500 MHz läuft, eingesetzt. Dieses ist klein, leicht, sehr stromsparend und kommt mit einem kleinen passiven Kühlkörper aus. Die Leistung ist aber lange nicht vergleichbar mit aktuellen Notebook- oder gar Desktop-CPUs. Deswegen ist eine sehr effiziente Bildverarbeitung erforderlich, um möglichst viele Bilder pro Sekunde verarbeiten zu können.

Eine Möglichkeit, dem Roboter ein Bild der Umwelt zu liefern, ist die klassische geometrische Objekterkennung. Bei dieser Methode wird versucht, durch die Form das Objekt zuzuordnen (z. B. rundes Objekt entspricht Ball). Da diese Erkennungsmethode im Allgemeinen relativ schwierig und rechenintensiv ist, wurde entschieden, die Objekte beim RoboCup farblich zu kennzeichnen, so dass allein aufgrund der Farbklasse die Objekte zugeordnet werden können. Der Ball ist z. B. orange, die Tore gelb und blau, das Feld grün und die Feldlinien weiß. Die Erkennung wird so erheblich vereinfacht und

benötigt auch viel weniger Rechenleistung als die klassische geometrische Objekterkennung.

Aber auch durch die Erkennung über Farben ergeben sich Probleme aufgrund unterschiedlicher Lichtverhältnisse je nach Standort und Beleuchtung. Wenn eine feste Zuordnung jedes Farbwertes zu einer Farbklasse verwendet wird, stimmt diese Zuordnung schon bei kleiner Änderung der Helligkeit nicht mehr. Dies hätte zur Folge, dass der Roboter die entsprechenden Objekte nicht mehr wahrnehmen könnte.

Aufgrund dieser Schwierigkeiten wird bei Wettbewerben darauf geachtet, dass möglichst einheitliche Lichtbedingungen gewährleistet sind. Es werden im Regelfall vor dem Spiel so genannte Farbtabellen (hier werden die Farbwerte den entsprechenden Farbklassen zugeordnet) per Hand von den Mannschaften erstellt und für das ganze Spiel beibehalten, in der Erwartung, dass sich die Lichtverhältnisse nicht ändern.

Diese Vorgehensweise hat sich bewährt. Allerdings ist zukünftig eventuell geplant, unter ungünstigeren Lichtbedingungen (z. B. unter freiem Himmel) zu spielen. In der MiddleSizeLeague hat dieser Umstieg beispielsweise schon teilweise stattgefunden. Spätestens dann ist eine adaptive (selbstständig anpassende) Farbtabelle unabdingbar, die aber eben sehr rechenaufwendig ist.

Eine große Hilfe wäre es aber bereits, dass vor dem Spiel selbstständig die Farbtabelle berechnet wird. Das würde einem das zeitaufwendige Erstellen per Hand ersparen. Außerdem kann hier das Problem der geringen Rechenleistung vernachlässigt werden, da die Berechnungen nicht auf dem Onboard-Computer des Roboters stattfinden müssen, sondern auf einen schnellen externen Computer verlagert werden können.

3. Existierende Realisierungen

3.1. Einführung

Es gibt bereits Ansätze für die automatische Farbklassifizierung. Dabei muss zwischen so genannten Online- und Offline-Verfahren unterschieden werden.

Beim Offline-Verfahren wird die Farbtabelle vor dem Spiel erlernt, gespeichert und dann eingesetzt. Während des Spiels wird sie allerdings nicht mehr angepasst und ist somit nur für gleich bleibende Lichtverhältnisse geeignet.

Eine Möglichkeit, auch während des Spiels die Farbtabelle anzupassen, bieten Online-Verfahren. Somit sind sie auch für plötzlich veränderte Lichtbedingungen geeignet. Dabei sollte aber die benötigte Rechenleistung berücksichtigt werden, da im Spiel auch viele andere Berechnungen laufen und Prozessorleistung benötigen.

3.2. Beispiel an vierbeinigen Robotern

An der Universität von Texas (Austin) wurde eine automatische Farbkalibrierung mit Sony AIBO Roboterhunden entwickelt[12]. Dabei wird ein Roboter auf dem Feld auf einen festdefinierten Startpunkt mit festgelegter Ausrichtung positioniert. Bekannt sind dem Roboter die interessanten Objekte, deren Größe, Form und Position auf dem Feld.

Bei Ausführung der automatischen Kalibrierung versucht der Roboter durch selbstständiges Bewegen auf dem Feld die interessanten Objekte wie Tore und Landmarken zu erkennen und deren Farben entsprechend zu klassifizieren. Da ihm seine Position bekannt ist (innerhalb einer gewissen Toleranz), vereinfacht dies die Objekterkennung, da in bestimmten Positionen das Auftauchen von entsprechenden Objekten im Bild erwartet wird. Außerdem werden so auch Fehlerkennungen vermieden.

Weiterer Vorteil dieses Verfahrens ist, dass alles automatisch geschieht, von der Bewegungsausführung bis zur Objekterkennung und entsprechenden Farbklassifizierung. Außerdem wird die Berechnung direkt auf dem Onboard-Computer des Roboters ausgeführt. Es wird also kein externer Rechner benötigt.

Nachteil ist allerdings, dass das Feld möglichst frei von Fremdobjekten sein muss. Der Roboter muss sich schließlich an seine verschiedenen Positionen bewegen können. Des Weiteren muss die Sicht auf die interessanten Objekte frei sein, um diese auch korrekt erkennen zu können.

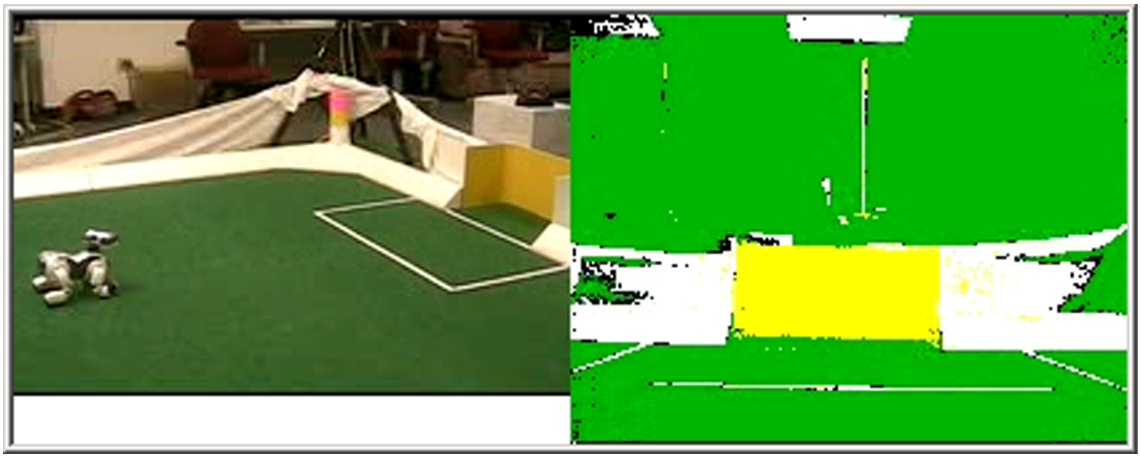

Abbildung 3.1.: Links: AIBO auf dem Spielfeld - Rechts: Segmentiertes Bild (Quelle [1])

In Abb. 3.1 sieht man links einen AIBO-Hund auf dem Spielfeld mit Blick in Richtung gelbes Tor. Auf der rechten Seite ist das erfasste Bild der Kamera in segmentierter Ansicht zu sehen, nachdem zuvor bereits das Feld, Linien und das gelbe Tor erkannt wurden.

3.3. Beispiel in der MiddleSizeLeague

3.3.1. Einführung

Im Unterschied zur HumanoidLeague bewegen sich Roboter der MiddleSizeLeague nicht auf Beinen, sondern auf Rädern fort. Dies vereinfacht die Bewegung erheblich, da die Roboter aufgrund der Konstruktion schon sehr stabil auf dem Untergrund stehen und nicht umfallen können.

Aufgrund des relativ hohen Eigengewichts der MiddleSize-Roboter (Abb. 3.2), ist es auch nicht so problematisch wie in der Humanoid KidSize League, noch weiteres an Gewicht hinzuzufügen. Dementsprechend können auch stärkere Akkus und Rechner zum Einsatz kommen. Für gewöhnlich werden Subnotebooks verwendet, die in ihrer Leistung schon nahe an schnelle Desktop-Rechner heranreichen. So können auf den MiddleSize-Robotern viel komplexere Berechnungen durchgeführt werden, als es bei der Humanoid-League möglich ist.

Es wird vermutlich noch einige Jahre dauern bis CPUs verfügbar sind, die so sparsam wie aktuell verwendete CPUs der Humanoid KidSize League sind und die Rechenleistung von derzeit verwendeten CPUs der MiddleSizeLeague erreichen. Der vor kurzem erschienene Intel Atom Prozessor ist schon ein Schritt in diese Richtung.

Abbildung 3.2.: Roboter der MiddleSizeLeague (Quelle [6])

3.3.2. Der ACT (Automatic color training) Algorithmus

An der Universität Tübingen wurde ein Verfahren für MiddleSize-Roboter[4] entwickelt, welches eine automatische Farbklassifizierung online (also während der Programmausführung) ermöglicht. Bedingung ist allerdings, dass der Roboter seine ungefähre Position kennt. Zu Beginn ist also eine vordefinierte Position einzuhalten. Sobald dann die Farben ausreichend gut klassifiziert sind, tritt die Selbstlokalisation in Aktion und ermöglicht auch Positionsbestimmungen während der Programmausführung.

Mit Hilfe der bekannten Positionen werden die von der Kamera erfassten Pixel in Weltkoordinaten umgerechnet. So wird die Information geliefert, welcher Pixel zum entsprechenden Spielfeldteil gehört. Statische Objekte wie Feld, Linien und Tore können so korrekt in den entsprechenden Farben klassifiziert werden. Beim Ball funktioniert dies aber nicht, da dessen Position auf dem Feld nicht statisch ist, sondern variieren kann.

Der Algorithmus ist sehr robust gegenüber spontan auftretenden Helligkeitsunterschieden. Selbst wenn die Änderungen so stark sind, dass fast alle Pixel im Bild unklassifiziert sind (siehe Abb. 3.3), ist die Farbtabelle nach acht Algorithmus-Durchläufen (ca. 0,2 Sekunden unter Berücksichtigung anderer laufenden Berechnungen) wieder sehr gut zu den aktuellen Helligkeitsbedingungen angepasst (siehe Abb. 3.4).

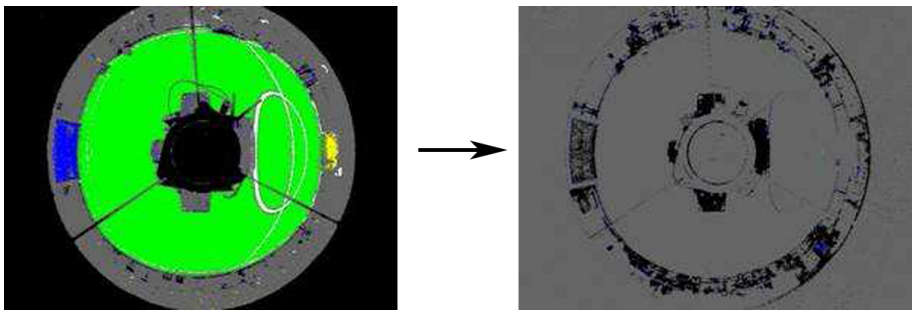

Abbildung 3.3.: Farbtabelle vor und nach Helligkeitsänderung (Quelle [4])

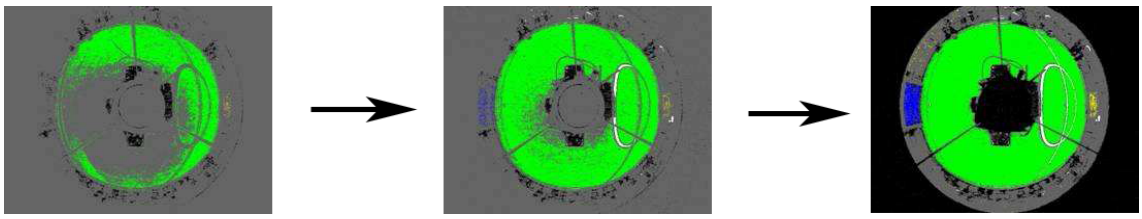

Abbildung 3.4.: Farbtabelle nach 1, 2 und 8 Ausführungen des ACT-Algorithmus (Quelle [4])

Die Ausführungszeit eines Durchlaufs des ACT-Algorithmus beträgt auf einem Rechner mit Athlon XP 2400+ (2 GHz) ca. 3-4 ms.

4. Grundlagen

4.1. Menschliches und maschinelles Sehen

Das menschliche und das maschinelle Sehen funktionieren sehr unterschiedlich. Die Eingabequelle, nämlich ein farbiges Bild, ist im Prinzip gleich; dessen Auswertung erfolgt jedoch sehr verschieden. Der Computer geht dabei sehr mathematisch vor, die Stärken liegen z. B. beim exakten Messen von Entfernungen sowie der exakten Bestimmung von Graustufen und Farben. Der Mensch kann solch eine Art von Messungen mit dem bloßen Auge nur schätzen.

Was für den Menschen sehr einfach ist, jedoch dem Computer große Probleme bereitet, ist das schnelle Erkennen von Objekten. In Abb. 4.1 links ist beispielsweise für einen Menschen in Bruchteilen einer Sekunde die Zahl Fünf erkennbar. Bei einem Computer muss dagegen ein Programm vorliegen, welches genau für die Erkennung solcher Texturunterschiede ausgelegt ist.

Beim rechten Bild der Abb. 4.1 ist es für den Menschen auch sehr leicht, ein Dreieck zu erkennen, obwohl Teile der Grenzen nicht sichtbar sind. Der Computer kann damit wenig anfangen, wenn er nicht speziell darauf programmiert ist.

Eine universelle Möglichkeit des „Bildverstehens“, so wie es der Mensch kann, ist bei einem Computer bislang undenkbar.

Beim RoboCup wurde auf die Problematik des maschinellen Sehens Rücksicht genommen und durch farbliche Kennzeichnung der Objekte entsprechende Vorkehrungen getroffen.

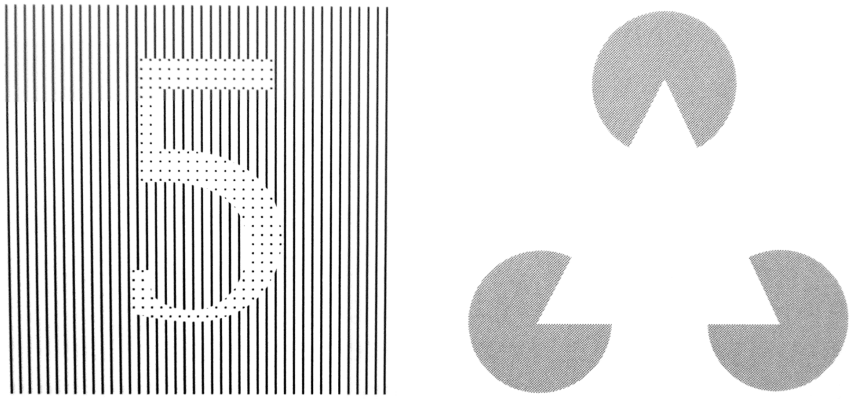

Abbildung 4.1.: Maschinelles Erkennen von Objekten (Quelle [5])

4.2. Farbmodelle

4.2.1. Grundlegendes

Farbmodelle dienen zur Modellierung von Farben auf dem Computer. Während bei einem Schwarz/Weiß-Bild für den Graustufenwert nur eine Dimension benötigt wird, sind es bei einem Farb-Bild gewöhnlich drei Dimensionen. Die weit verbreitesten Farbmodelle sind z. B. RGB, CMY, YUV und HSI.

Für die Erkennung von farblich gekennzeichneten Objekten im RoboCup galt es nun, für die vorliegende Arbeit eine Möglichkeit zu finden, die Farbklasse eines Pixels möglichst effektiv zu bestimmen. In den beiden folgenden Kapiteln werden dazu das RGB- und das YUV-Farbmodell miteinander verglichen.

4.2.2. RGB-Farbmodell

RGB ist das bekannteste aller Farbmodelle und steht als Abkürzung für die verwendeten Farben rot, grün und blau. Fernseher und Computer-Bildschirme verwenden es beispielsweise zur Darstellung von farbigen Bildern. Durch Mischung dieser drei Farbkanäle können je nach verwendeter Farbtiefe die meisten Farben dargestellt werden. Bei den oft verwendeten 8 Bit pro Farbkanal ergeben sich z. B. 16.777.216 verschiedene Farben. Das

RGB-Farbmodell lässt sich zur Veranschaulichung auch als ein dreidimensionaler Würfel darstellen (siehe Abb. 4.2 links).

Verschiedene Helligkeiten lassen sich durch die Intensitäten der Farbkanäle regeln. Umso höher die Intensitäten, desto heller erscheint die Farbe. Sind alle Intensitäten der R-, G- und B-Kanäle auf Maximum, entspricht dies der Farbe weiß.

Da für die Feststellung der Farbklasse beim RoboCup viele Farben relevant sind, müssten alle drei Farbkanäle zur Auswertung herangezogen werden.

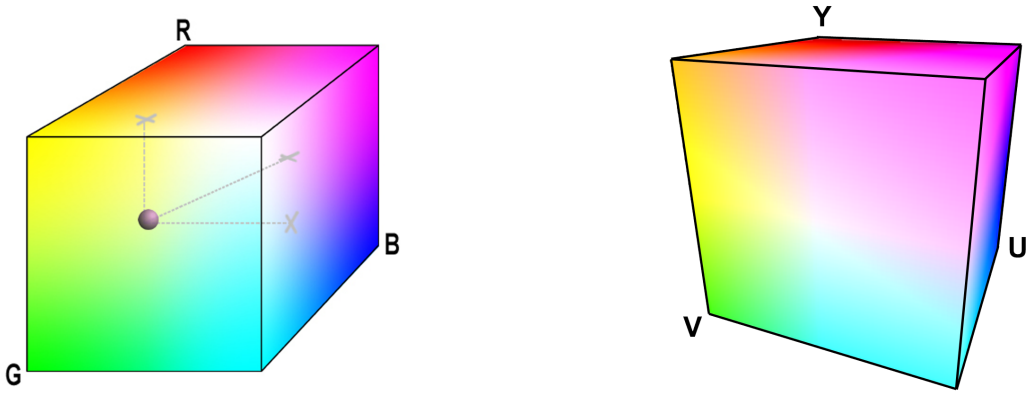

Abbildung 4.2.: RGB-(Quelle [3]) und YUV-(Quelle [11]) Farbmodell in dreidimensionaler Darstellung

4.2.3. YUV-Farbmodell

Das YUV-Modell wurde vor einigen Jahrzehnten aus der Not heraus entwickelt, als das Farbfernsehen auf den Markt kam. Damals wurde nach Möglichkeiten gesucht, ein Fernsehsignal in Farbe auszustrahlen, welches aber auch alte s/w-Fernseher noch korrekt in Graustufen darstellen konnten.

YUV ist wie RGB auch dreidimensional. Es werden aber nicht drei Grundfarben verwendet, sondern getrennte Helligkeits- und Farbinformationen. Der Y-Kanal entspricht der Helligkeit, auch Luminanz (Lichtstärke pro Fläche) genannt. Die U- und V-Kanäle in Kombination ergeben den Farbanteil bzw. die Chrominanz. Das alte Schwarz/Weiß(s/w)-Fernsehsignal (entsprechend dem Y-Wert) wurde also um die beiden U- und V-Kanäle ergänzt, welche von den s/w-Fernsehgeräten einfach ignoriert werden. Farbfernseher werten entsprechend alle drei Farbkanäle aus und können so das Fernsehsignal in Farbe anzeigen.

Verwendet wird das YUV-Modell sowohl im europäischen PAL- als auch im amerikanischen NTSC-Standard.

Zur Bestimmung der Farbklasse eines Pixels genügt es, die Kanäle für die Farbinformationen auszuwerten. Der entscheidende Vorteil von YUV gegenüber RGB ist, dass beim YUV-Modell dazu nur zwei Kanäle (U- und V- Kanal) betrachtet werden müssen.

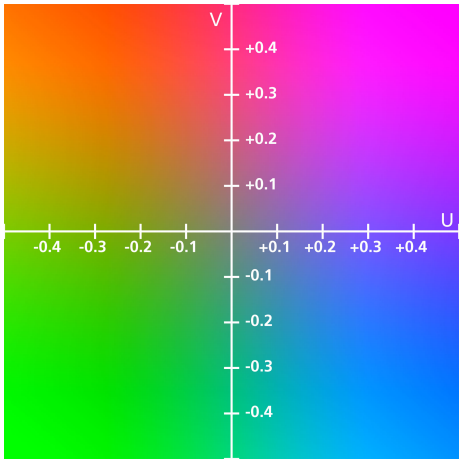

Abbildung 4.3.: UV-Ebene bei Y=0,5 (Quelle [2])

In Abb. 4.3 ist die UV-Ebene bei Y=0,5 (mittlere Helligkeit) dargestellt. Man kann hier gut erkennen, dass alle Farbklassen vorhanden sind. Um nun beispielsweise abzufragen, ob ein Pixel grünähnlich ist, genügt die Bedingung $U < 0$ und $V < 0$. Für andere Farbklassen funktionieren die Abfragen ähnlich, entsprechend mit anderen Bedingungen für U- und V-Werte.

Diese Methode zur Farbklassen-Bestimmung erschien plausibel und wird augrund dessen bei der automatischen Farbklassifizierung in dieser Arbeit verwendet.

Weitere Informationen zu Farbmodellen finden sich in [7].

4.3. Farbmodell digital

Theoretisch gibt es in einem Farbraum unendlich viele Farbwerte. In der digitalen Welt muss man sich jedoch auf eine gewisse Farbtiefe beschränken. Im RGB-Farbraum verwendet man in der Praxis z. B. oft 8 Bit Farbtiefe für jeden Kanal. Dies entspricht 256

Werten in jeder Dimension. Insgesamt ergeben sich so 256 * 256 * 256 = 16.777.216 mögliche Farben.

In der Software der TU Darmstadt für die humanoiden Roboter werden momentan aus Speicher und Performancegründen lediglich 6 Bit für den Y-Kanal und jeweils 5 Bit für U- und V-Kanal verwendet. Dies ergibt zusammen 16 Bit und entspricht 65.536 möglichen Farben. Dies ist aufgrund der deutlichen Farbunterschiede der verschiedenen Objekte ausreichend.

Für die Verarbeitung in der Applikation und auch für Parameter-Einstellungen in der GUI (Grafische Benutzeroberfläche) werden die Werte jedes Kanals jedoch auf 8 Bit hochgerechnet, liegen also in Bereichen zwischen 0 und 255.

4.4. Zusammenspiel Farbtabelle - Bildverarbeitung

Für jeden Farbwert gibt es in der Farbtabelle eine Zuordnung auf eine Farbklasse wie z. B. grün oder weiß. An die Bildverarbeitung wird während des Programmablaufs nur die Farbklasse weitergereicht (siehe auch Abb. 4.4).

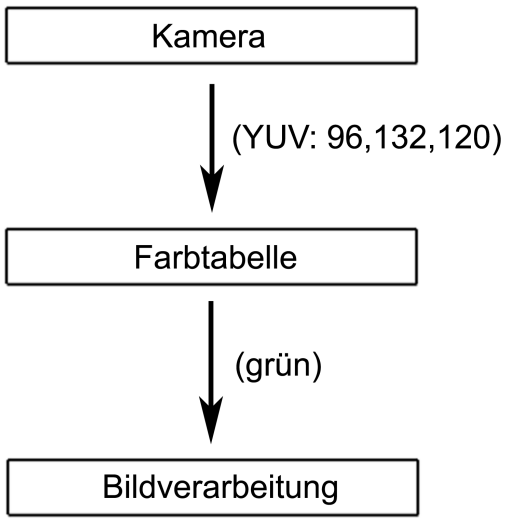

Abbildung 4.4.: Zusammenspiel Farbtabelle - Bildverarbeitung

Die Bildverarbeitung ist auf eine gut klassifizierte Farbtabelle angewiesen, da ausschließlich die klassifizierten Farben ausgewertet werden und keine Farbwerte. Gut klas-

sifiziert bedeutet, dass die relevanten Objekte möglichst mit der zugehörigen Farbe ausgefüllt sind und bei Fremdobjekten in der Umgebung die klassifizierten Farben möglichst selten, im Idealfall nie auftauchen.

Beides zugleich ist im Normalfall schwer erzielbar. Die Bildverarbeitung, welche für die humanoiden Roboter der TU Darmstadt eingesetzt wird, ist aber sehr robust. Ein paar wenige Fehlpixel in der Umgebung werden zuverlässig ignoriert, so dass es zu keinen Fehlerkennungen kommt. Allerdings sollten insbesondere orangeklassifizierte Pixel vermieden werden, welche im Bild in der Umgebung außerhalb des Spielfeldes auftreten. Diese könnten unter Umständen doch als Ball erkannt werden, falls sie gehäuft auftreten.

4.5. Median

Bei vielen Algorithmen in dieser Arbeit wird der so genannte Median-Wert berechnet. Es ist genau der Wert zwischen zwei Hälften, also wenn in einer Menge 50 Prozent der Werte kleiner und 50 Prozent der Werte größer sind. Im Gegensatz zum arithmetischen Mittel bzw. Durchschnitt ist der Median-Wert robuster gegenüber stark abweichenden Werten. Dieser Effekt lässt sich gut ausnutzen, um einen mittleren Farbwert einer dominanten Farbe in einem Bild zu bestimmen.

Wird beispielsweise der Median des Farbwerts auf einem Bild berechnet, worauf sich Spielfeld und Linien befinden, wird dies in jedem Fall ein Farbwert eines Spielfeld-Pixels sein. Bei der Durchschnitts-Berechnung hingegen würden die weißen Pixel der Linien das Ergebnis stark beeinflussen. Das Resultat wäre dann ein Farbwert zwischen grün und weiß.

In dieser Arbeit wird zur Berechnung des Median der effiziente Wirth-Algorithmus [13] eingesetzt.

5. Konzept

In dieser Arbeit war es das Ziel, eine automatische Farbklassifizierung für den RoboCup im Offline-Verfahren zu realisieren. D. h. die Farbklassifizierung wird in der Vorbereitungsphase vor dem eigentlichen Einsatz des Roboters durchgeführt. Die daraus resultierende Farbtabelle kann anschließend im Spielbetrieb verwendet werden. Da sie statisch ist, sich also während des Einsatzes nicht anpassen kann, ist dieses Verfahren nur für gleich bleibende Lichtverhältnisse geeignet.

Um eine relativ schnelle Ausführung der automatischen Farbklassifizierung zu ermöglichen, finden die Berechnungen nicht auf dem Computer des Roboters, sondern auf einem externen Rechner statt. Weiterer Vorteil dabei ist, dass das Ergebnis im COLORTABLE-Dialog der GUI sofort überprüft werden kann.

Für die einfache Bedienung wurde ein eigener Dialog in der GUI implementiert, der AUTOCOLORTABLE-Dialog. Im Normalfall genügt es, in diesem Dialog „FULL AUTO CLASSIFICATION“ anzuwählen, woraufhin eine vollständige automatische Farbklassifizierung durchgeführt wird. Das Resultat ist eine Farbtabelle mit allen für einen Spielbetrieb relevanten Farbzuordnungen (mit Ausnahme von Hindernissen), mit der der Roboter bei den aktuellen Lichtbedingungen eingesetzt werden kann.

Für die Berechnungen werden als Eingabe ausschließlich von der Kamera des Roboters aufgenommene Bilder verwendet - entweder durch direkte Verarbeitung oder nachträglich durch aufgezeichnete Bilder in einer oder mehreren Log-Dateien. Daten zur Position des Roboters auf dem Feld werden nicht benötigt. Die Schwierigkeit dabei ist, dass es keinerlei Informationen gibt, was sich in jedem einzelnen Bild an auswertbaren Objekten befindet. Es können also nur die Bilder selbst zur Auswertung verwendet werden.

Dies hat aber den Vorteil, dass die Aufnahme von Bildern relativ wahllos erfolgen kann. In der Praxis kann man so mit dem Roboter in der Hand auf das Spielfeld gehen, sich per WLAN oder LAN mit einem externen Rechner verbinden und entweder die Aufzeichnung der Bilder in eine Log-Datei oder die direkte automatische Farbklassifizierung starten. Die Kamera des Roboters braucht dann nur auf die wichtigen Objekte des RoboCups (Feld, Linien, Tore, Landmarken, Ball) gerichtet werden. Fremdobjekte wie andere

Personen oder Gegenstände auf dem Spielfeld, was bei Vorbereitungsphasen in wichtigen Turnieren oft der Fall ist, stören nicht, da sie bei der automatischen Farbklassifizierung ignoriert werden.

Zur Erkennung herangezogen werden Informationen über die Objekte wie Form, Farbwerte der zugehörigen Pixel und zum Teil umliegende Farbklassen. Bei der Erkennung des Balls wird z. B. überprüft, ob das untersuchte Objekt kreisförmig ist. Für die Linienerkennung ist das Vorhandensein von grünklassifizierten Pixeln um das untersuchte Objekt Voraussetzung.

Für die sinnvolle Anwendung wird der bereits existierende COLORTABLE-Dialog benötigt, der zur Visualisierung der Farbtabelle dient. Durch Aktivierung der „AUTO SEND COLORTABLE“-Funktion im AUTOCOLORTABLE-Dialog wird bei jedem Bild die aktuelle Farbtabelle an den COLORTABLE-Dialog gesendet. So kann anhand der segmentierten Bilder live mitverfolgt werden, wie sich die Farbtabelle entwickelt.

Bei manuellen Änderungen der Farbtabelle lassen sich diese aber auch über den Send-Button im COLORTABLE-Dialog wieder an den AUTOCOLORTABLE-Dialog zurücksenden und die automatische Erkennung mit dieser Farbtabelle fortsetzen. Der Datenaustausch kann also in beide Richtungen erfolgen (siehe auch Abb. 5.1).

Im AUTOCOLORTABLE-Dialog besteht außerdem die Möglichkeit, Erkenner und spezielle Zusatzfunktionen nach Belieben getrennt zu aktivieren. Für jedes Objekt gibt es einen eigenen Erkenner, also für Feld, Linien, Bälle, Tore und Landmarken.

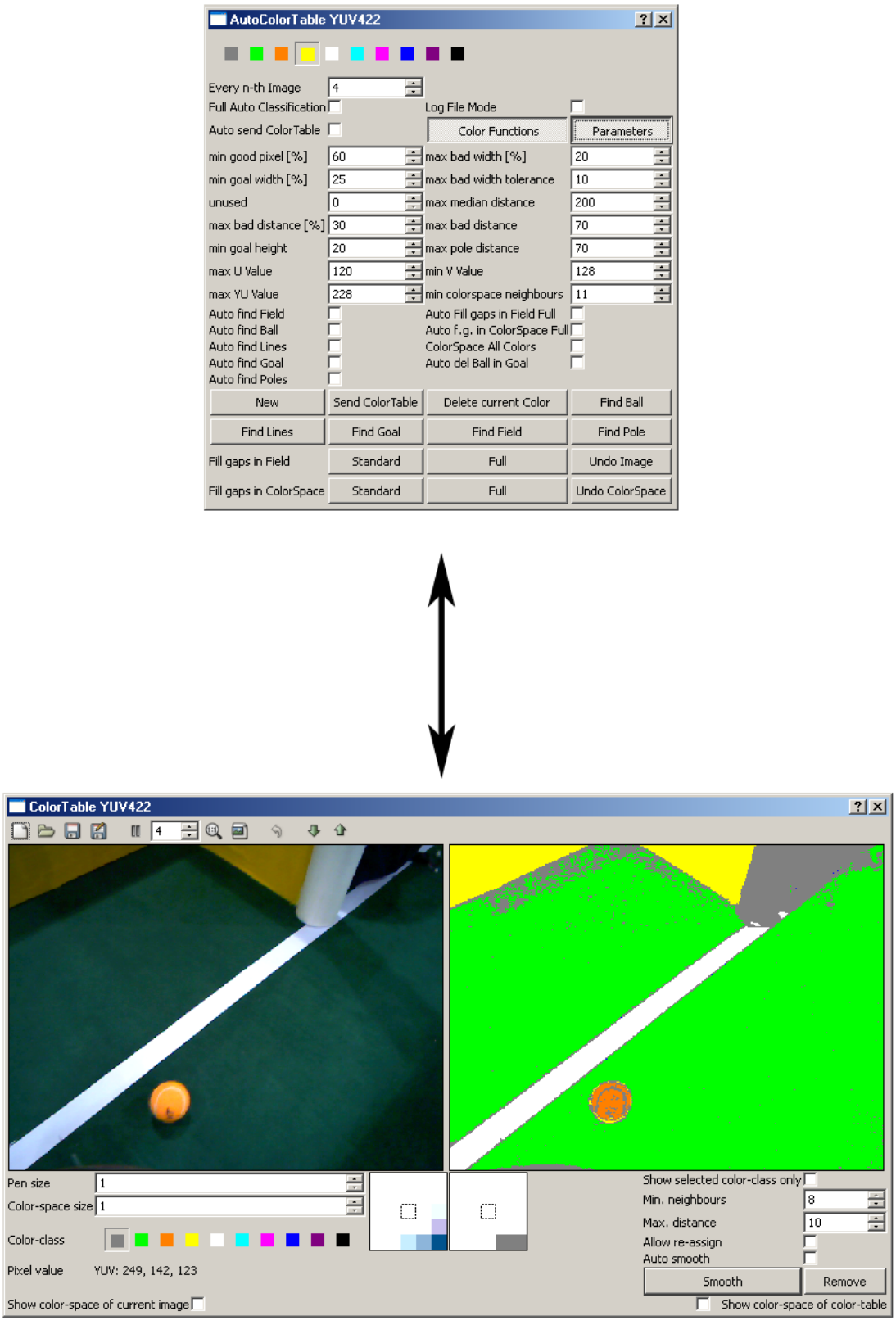

Abbildung 5.1.: Kommunikation zwischen den Farbtabellen-Dialogen

6. Erkenner

In diesem Kapitel wird die Funktionsweise der einzelnen Erkenner beschrieben. Die Implementierung der Erkenner war der wesentliche Teil dieser Arbeit. Nur durch die Erkennung der relevanten Objekte des RoboCups können deren Farbwerte den entsprechenden Farbklassen zugeordnet werden. Diese Zuordnung wird schließlich in einer Farbtabelle abgespeichert und im Spielbetrieb von der Bildverarbeitung verwendet.

Für jeden Erkenner lassen sich im AUTOCOLORTABLE-Dialog Parameter einstellen. Sie beeinflussen, wie tolerant der jeweilige Erkenner bei der Objekterkennung ist. Man darf sie nicht zu tolerant wählen, weil sonst Fehlerkennungen auftreten können. Die Toleranz darf aber auch nicht zu stark eingeschränkt werden, da sonst die Objekte eventuell gar nicht mehr erkannt werden. Die Standardeinstellungen der Parameter sind so gewählt, dass die Erkenner in den meisten Fällen gute Resultate erzielen.

In den folgenden Beschreibungen der Erkenner sind die Parameter in kursiver Schreibweise dargestellt. Die Bedeutung jedes einzelnen Parameters ist im Anhang, Kapitel B beschrieben.

Die horizontale Achse des Bildes wird in den Beschreibungen, wie im zweidimensionalen Raum üblich, als x-Achse bezeichnet - die vertikale als y-Achse.

6.1. Feld-Erkennung

6.1.1. Grundlegendes

Das Feld ist beim RoboCup in der Humanoid-Liga ein Teppich in grüner Farbe. Es befindet sich fast immer im Bild, welches von der Roboter-Kamera erfasst wird (in der Annahme der Roboter befindet sich auf dem Feld). Da es aber meistens eher im unteren Teil des Bildes zu sehen ist, wird die Erkennung auch nur auf dem unteren Drittel des Bildes

durchgeführt. So ist die Wahrscheinlichkeit sehr hoch, dass in diesem Bereich Feldpixel vorhanden sind.

Typische Objekte auf dem Feld, die im Bild auftreten, sind Linien und eventuell vorhandene Bälle. Auf Wettbewerben wie dem RoboCup befinden sich in der Vorbereitungsphase sehr häufig auch Fremdobjekte wie Personen, Notebooks und Roboter auf dem Spielfeld. Einfach das untere Bilddrittel grün zu klassifizieren, würde in den meisten Fällen zu keinem korrekten Ergebnis führen.

Eine wichtige Eigenschaft des Feldes ist, dass es abgesehen von Helligkeitsunterschieden einen relativ gleich bleibenden Farbwert hat. Lokale Differenzen durch Teppichfranzen und globale Abweichungen durch Schattenbildung treten aber sehr wohl auf.

6.1.2. Realisierung von FINDFIELD

Die Erkennung beruht darauf, dass im unteren Bilddrittel nach einer großen Anzahl an Pixeln gesucht wird, die einen ähnlichen Farbwert aufweisen. Dazu wird zuerst der Median-Farbwert berechnet. Das Vorhandensein von mindestens 50 Prozent Feldpixeln im unteren Bilddrittel genügt schon, dass der Median-Farbwert auf einen von diesen trifft.

Nun werden alle Pixel im unteren Bilddrittel auf Farbwert-Distanzen zum Median-Farbwert überprüft. Liegt eine sehr große Anzahl an Pixeln (*min good pixel[%]* - standardmäßig 75 Prozent) in der Nähe des Median-Farbwerts (*max distance*), bedeutet dies, dass eine große Anzahl an Pixeln mit ähnlichen Farbwerten gefunden wurde.

Mit hoher Wahrscheinlichkeit handelt es sich bei diesen Pixeln um Feldpixel. Es könnte sich aber z. B. auch um ein Bild eines gelben oder blauen Tores halten, aufgenommen aus sehr naher Distanz. Dies würde das untere Bilddrittel auch mit mehr als 75 Prozent Pixel mit ähnlichem Farbwert füllen. Um diese Fehlerkennung zu verhindern, ist eine Farbwert-Kontrolle implementiert. Bei einem grünähnlichen Farbwert sind die U- und V-Kanäle eher im niedrigen Bereich. Als Schwellwert ist standardmäßig beim U-Kanal 140 (*max U value*) und beim V-Kanal 120 (*max V value*) gewählt.

Sind der U- und V-Wert des Median-Farbwerts unterhalb der Schwellwerte, wird die Untersuchung fortgesetzt. Als weitere Vorsichtsmaßnahme werden allerdings nicht alle Pixel grün eingefärbt, die beim vorigen Test in der Nähe des Median-Farbwertes lagen, sondern die Toleranz der Farbwert-Nähe nochmals heruntergesetzt (die Hälfte von *max distance*). So werden nur die Pixel grün eingefärbt, die sehr nahe am Median-Farbwert liegen. Dies verhindert, dass eventuell vorhandene grünähnliche Objekte wie eine Person

mit einem grünen Pullover, miteingefärbt werden. Nachteil ist allerdings, dass eventuell auch Feldpixel nicht eingefärbt werden.

Das Feld wird also vermutlich noch unsegmentierte Pixel beinhalten, besonders bei Feldpixeln, die sich im oberen Bildbereich befinden, da diese ja gar nicht erst berücksichtigt werden.

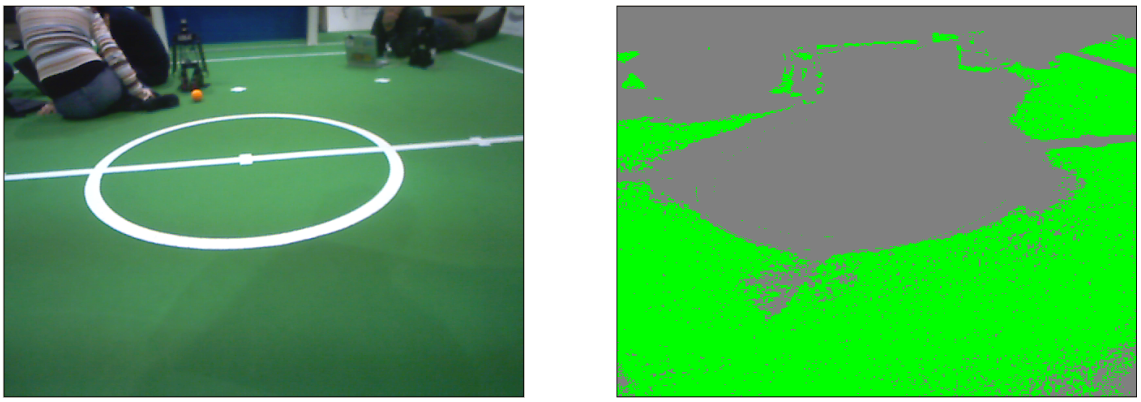

Abbildung 6.1.: Segmentiertes Bild nach Ausführung von „Find field“

In Abb. 6.1 sieht man ein Beispiel eines segmentierten Bildes nach Ausführung der „Find field“-Funktion.

6.2. Ball-Erkennung

6.2.1. Der Ball in der Humanoid KidSize-Liga

Der Ball ist das wichtigste Objekt beim RoboCup. Nur wenn dieser möglichst schnell in das gegnerische Tor befördert wird, kann man gewinnen.

Anders als beim Fußball für die menschliche Spezies sind die Bälle für den RoboCup in der KidSize-Liga nicht aus Leder, nicht weiß und ein ganzes Stück kleiner. Dies ist verständlich, da die Roboter auch erheblich kleiner sind verglichen zum Menschen und auch nicht soviel Kraft aufwenden können.

Um die Erkennung zu vereinfachen, ist der Ball außerdem orange gefärbt. Dies soll Verwechslungen verhindern, da die Farbe Orange beim RoboCup für kein anderes Objekt verwendet wird und in der Umgebung nur recht selten auftritt.

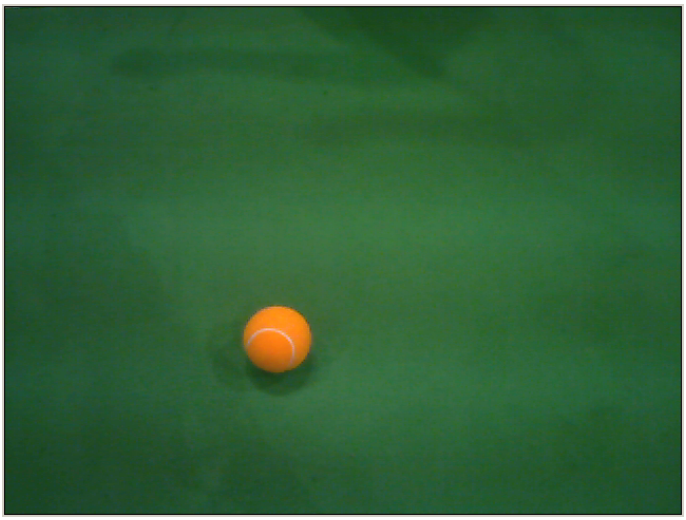

Abbildung 6.2.: Von der Roboter-Kamera aufgezeichneter Ball, der seit 2008 eingesetzt wird

Bis zum RoboCup 2007 wurden Plastikbälle verwendet, die von der AIBO-Liga übernommen wurden. Vorteil dieses Modells war das geringe Gewicht, so dass auch schwache Roboter leicht den Ball fortbewegen konnten. Nachteil war aber, dass leichte Unebenheiten auf dem Untergrund oft zu unerwarteten unverhersehbaren Richtungsänderungen des Balls während des Rollens führten. Da die Roboter mit der Zeit auch stabiler wurden, sah man sich nach einer Alternative mit höherem Gewicht um, die es aber auch serienmäßig zu kaufen geben sollte. So entschied man sich schließlich für einen handelsüblichen Tennisball. Seit dem Jahre 2008 werden diese nun offiziell verwendet.

6.2.2. Grundlegendes

Eine zuverlässige Ballerkennung ist von sehr hoher Bedeutung, da natürlich nur dann der Roboter sich gut positionieren und Richtung Tor schießen kann. Mindestens genauso wichtig ist aber auch, dass in der Umgebung Bälle nicht fehlerkannt werden, wenn sich dort etwas Orangefarbenes befindet, wie z. B. ein Zuschauer mit einer orangefarbenen Jacke.

Falls ein Fremdobjekt im Bild exakt die gleichen Farbwerte hat wie der Ball, ist es über die Farbtabelle nicht möglich, beides zu differenzieren. Dies ist dann Aufgabe der Bildverarbeitung, orangefarbene Fremdobjekte zu filtern. Es kann z. B. überprüft werden,

ob das Objekt möglichst rund und von möglichst viel grün oder weiß umgeben ist. Dies sind Indizien für einen Ball, der auf dem Spielfeld liegt.

Aber auch über die Farbtabelle kann man Fehlerkennungen vorbeugen, indem spezielle Farbwerte auf dem Ball explizit nicht orange gefärbt werden. Beim Plastikball, der bis 2007 verwendet wurde, traten z. B. Reflexionen der Beleuchtung auf dem Ball auf, die so hell waren, dass sie im Bild gelb oder sogar weiß erschienen. Eine Segmentierung dieser Farbwerte zu orange in der Farbtabelle wäre fatal, da so im gelben Tor und auf den Linien mit hoher Wahrscheinlichkeit orangesegmentierte Pixel auftreten und eventuell dort auch Bälle erkannt würden. Bei der manuellen Farbtabellenerstellung muss man darauf sorgfältig achten. Die automatische Erkennung berücksichtigt dies ebenso.

Beim Tennisball (siehe Abb. 6.2), der seit 2008 eingesetzt wird, sind die Reflexionen nicht mehr so stark ausgeprägt, da die Balloberfläche viel matter ist und nicht so stark glänzt. Dafür befinden sich weiße, geschwungene Linien und eventuell ein schwarzer Werbeschriftzug des Herstellers darauf. Auch hier muss drauf geachtet werden, dass diese weißen und schwarzen Pixel in der Farbtabelle nicht orange gefärbt werden.

6.2.3. Realisierung von FINDBALL

Die Ballerkennung funktioniert nach folgendem Prinzip. Es wird davon ausgegangen, dass der Ball auf dem Feld liegt, also nur von grün oder weiß umgeben ist. Außerdem ist Voraussetzung, dass das zu untersuchende Objekt möglichst rund und eine orangeähnliche Farbe aufweist. Um für gute Performance zu sorgen, werden Pixel möglichst schnell ausgeschlossen, die nicht einem Ball zugehörig zu seien scheinen.

Im ersten Schritt wird das Bild pixelweise nach noch nicht segmentierten Pixeln abgesucht. Wenn rechts, links, oben und unten nach den unsegmentierten Pixeln kein grün- oder weißklassifizierter Pixel auftritt, werden die Pixel als „verworfen“ markiert. Im anderen Fall, wenn in allen vier Richtungen ein grün- oder weiß-klassifizierter Pixel gefunden wird, werden diese Pixel für die weitere Untersuchung freigestellt. Zur Performanceverbesserung werden allerdings nur Objekte freigestellt, wenn standardmäßig mehr als 20 unsegmentierte Pixel (*min ball size*) vor dem Auftreffen auf grün/weiß gefunden werden. Sehr kleine Objekte werden so sofort ausgeschlossen, um Rechenzeit zu sparen.

Im zweiten Schritt wird für die gefundenen Objekte der Mittelpunkt bestimmt. Anschließend wird von diesem Mittelpunkt aus geprüft, ob es sich um ein möglichst rundes Objekt handeln (siehe Abb. 6.3). Dazu werden alle abgescannten Pixel mit denen eines

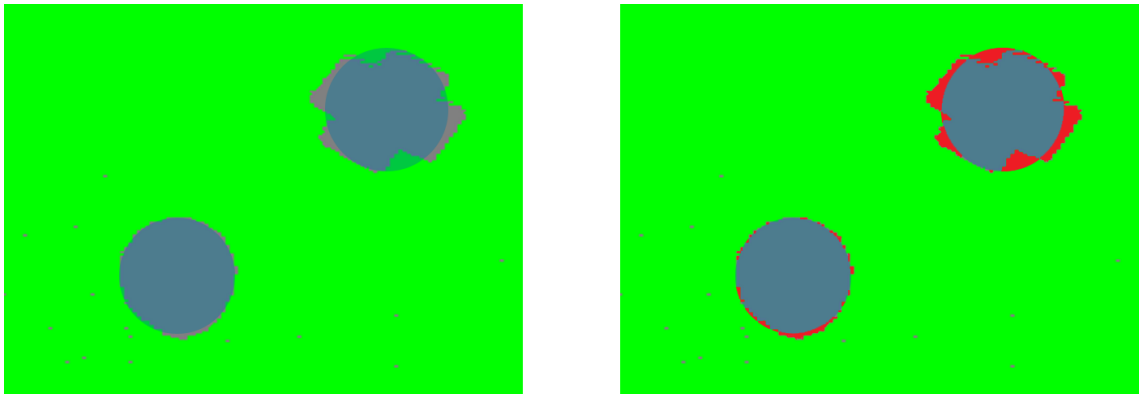

Abbildung 6.3.: Vergleich der Form des gefundenen Objektes mit einem Kreis

optimalen Kreises verglichen. Die Fehlpixel (in Abb. 6.3 rechts rot markiert) werden dabei aufaddiert. Ist die Gesamtdifferenz, also die Abweichung des Objektes von einem Kreis, höher als ein einstellbarer Maximalwert (tolerance), wird das Objekt nicht für die weitere Ballerkennung zugelassen.

Im letzten Schritt wird nun das Objekt farblich untersucht. Bedingung ist, dass es eine orangeähnliche Farbe besitzt und nicht zu viele Farbunregelmäßigkeiten aufweist. Zur Überprüfung von Farbunregelmäßigkeiten wird bei jedem Pixel im Objekt der Farb-Abstand zum linksbenachbarten Pixel gemessen. Dieser muss unterhalb von *max Distance* liegen, um als gut bewertet zu werden. Ist der Anteil an gut bewerteten Pixeln über den standardmäßigen 60% (*min good pixel [%]*), wird die Untersuchung fortgesetzt. Gelbsegmentierte Pixel werden außerdem ignoriert bzw. beim Scannen wie ein unsegmentierter Pixel behandelt, da Farbwerte des gelben Tores sich oft leicht mit Farbwerten des Balles überschneiden.

Bevor nun die bis jetzt als Ball zulässigen Pixel orange segmentiert werden, wird erst der Mittelwert der YUV-Werte des Objektes bestimmt. Bedingung ist, dass dieser Mittelwert orangeähnlich ist. Dazu werden die U- und V-Kanäle ausgewertet. Typisch für Orange sind ein kleiner U-Wert und ein großer V-Wert. Die Schwellwerte dazu sind hier standardmäßig für den U-Kanal maximal 110 (*max U value*) und beim V-Kanal minimal 140 (*min V value*). Nur wenn diese Werte nicht unter- bzw. überschritten werden, wird der Ball eingefärbt.

Orange werden nun nur die Pixel segmentiert, die einen gewissen Distanzwert vom Durchschnitts-Farbwert (*max average distance*) nicht überschreiten. Dies verhindert, dass insbesondere kleine helle Stellen, die durch die Reflexion der Lichtquellen hervorgerufen werden, nicht orange segmentiert werden.

Des Weiteren kann auch eingestellt werden, dass nur ein bestimmter Anteil des Ballradius zur Segmentierung (*fill only xx % of radius*) berücksichtigt wird. Dadurch wird verhindert, dass Pixel eingefärbt werden, welche sich am Rand befinden und gewöhnlich eine Mischfarbe mit grün (Feld) oder weiß (Linien) einnehmen. Standardmäßig werden 50% des Ballradius berücksichtigt.

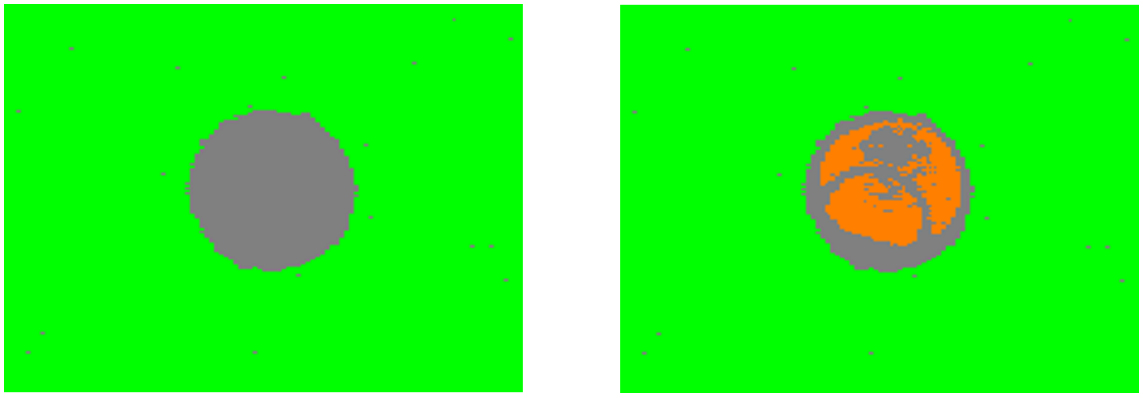

Abbildung 6.4.: Links: unklassifizierter Ball - Rechts: Ball nach „Find Ball“-Ausführung

In Abb. 6.4 sieht man die Farb-Klassifizierung des Balls vor und nach der Ausführung des FINDBALL-Algorithmus. Man sieht, dass der Ball noch nicht sonderlich gut orange ausgefüllt ist, da wie bereits beschrieben, sicherheitshalber nur die Pixel orange klassifiziert werden, die sehr nahe am Durchschnitts-Farbwert liegen.

Abbildung 6.5.: Segmentierter Ball nach „Find Ball“-Ausführung über alle Bilder einer gesamten Log-Datei

Führt man den FINDBALL-Algorithmus jedoch bei vielen Bildern aus, worauf der Ball

aus verschiedenen Perspektiven und verschiedenen Ausleuchtungen zu sehen ist, füllt sich der Farbraum weiter mit orange aus. In diesem Beispiel sieht der Ball nach dem vollständigen Abspielens der zugehörigen Log-Datei z. B. wie in Abb. 6.5 aus.

6.3. Linien-Erkennung

6.3.1. Grundlegendes

Erkannte Linien sind von entscheidender Bedeutung für die Selbstlokalisierung. Um eine gute Selbstlokalisierung zu gewährleisten, sollten die Linien deshalb möglichst vollständig mit weißer Farbe segmentiert sein, um der Bildverarbeitung die besten Voraussetzungen für eine gut funktionierende Linienerkennung zu bieten.

6.3.2. Realisierung von FINDLINES

Voraussetzung für eine funktionierende Linienerkennung ist, dass das Feld bereits größtenteils grün segmentiert ist. Bei der Suche nach einem passenden Objekt wird überprüft, ob es möglichst gerade verläuft, im Verlauf keine große Abweichung in der Breite hat, und wenn dann eine gleichförmige Änderung der Breite (durch Verzerrung hervorgerufen). Außerdem muss es eine weißähnliche Farbe aufweisen und nicht viele Farbunregelmäßigkeiten.

Im ersten Schritt wird in y-Richtung nach Grün/Unsegmentiert/Grün-Folgen gesucht. Nach dem Auffinden einer solchen Folge wird der Mittelpunkt des unsegmentierten Bereichs bestimmt und dessen Farbwert mit dem Farbwert des zuletzt gefundenen grünsegmentierten Pixels verglichen. Ist die Distanz kleiner als *minDistanceGreen* wird davon ausgegangen, dass es sich um einen grünen Pixel handelt, der noch nicht zu grün segmentiert wurde. Dieser wird nicht mehr zur weiteren Linienerkennung zugelassen.

Anschließend wird die Prozedur einen Pixel weiter rechts wiederholt. Dies geschieht solange, bis beim nächstrechten Pixel kein unsegmentierter Pixel gefunden wird oder das Bildende erreicht ist. Bei jedem Sprung in x-Richtung (nach rechts) wird die y-Position des Mittelpunktes des unsegmentierten Bereiches und die Höhe in y-Richtung gespeichert.

Im zweiten Schritt wird überprüft, ob das gefundene Objekt eine Mindestlänge (*minLineLength*) in x-Richtung erreicht. Falls nicht, ist das Objekt für die weitere Erkennung ausgeschlossen.

Beim dritten Schritt wird überprüft, ob das Objekt an allen x-Positionen möglichst gleich hoch ist. Dazu wird der Median-Wert aller Höhen-Werte bestimmt und mit allen gemessenen Höhen verglichen bzw. die Differenzen gebildet. Alle Differenzen die größer sind als *maxBadHeightTolerance*, werden als „schlecht“ gewertet. Wenn die Anzahl der schlechten Höhenwerte mehr als *maxBadHeight [%]* (maximale prozentual nicht akzeptierte Höhenwerte) beträgt, wird das Objekt verworfen.

Beim vierten Schritt wird überprüft, ob das Objekt einen linienförmigen Verlauf zeigt. Dazu wird der Start- und Endpunkt des Objektes anhand der gespeicherten y-Positionen bestimmt und eine virtuelle Linie gezogen, deren y-Werte mit den tatsächlich gespeicherten y-Werten verglichen werden. Alle Differenzen die größer sind als *maxBadYPosTolerance* werden als „schlecht“ gewertet. Wenn die Anzahl der schlechten y-Werte mehr als *maxBadYPos [%]* (maximale prozentual nicht akzeptierte y-Werte) beträgt, wird das Objekt verworfen. Siehe hierzu Abb. 6.6: Auf dem linken Bild sind die gemessenen Mittelpunkte gelb markiert. Rechts wird der Vergleich mit einer Geraden deutlich gemacht.

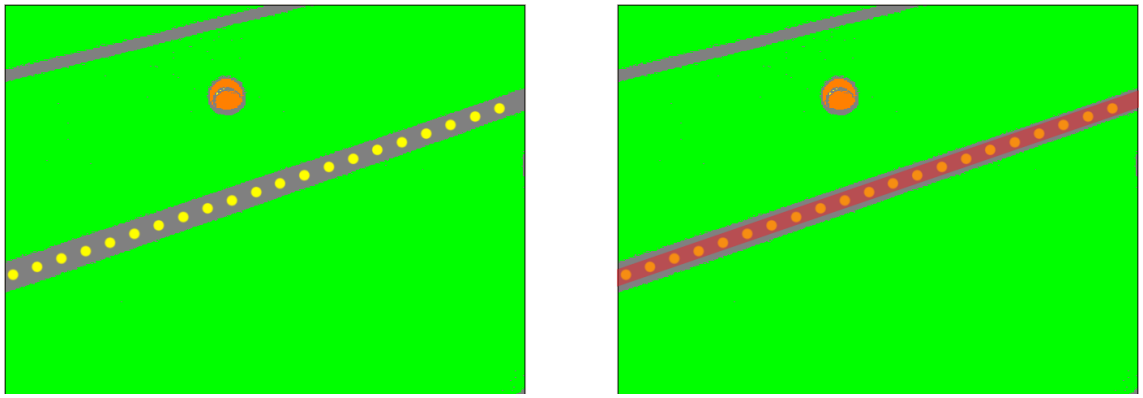

Abbildung 6.6.: Links: gemessene Mittelpunkte - Rechts: Vergleich mit Gerade

Im fünften Schritt wird die korrekte linientypische Verzerrung überprüft. Diese macht sich insofern bemerkbar, dass eine Linie in der Nähe breiter erscheint als in der Ferne. Nun wird überprüft, ob der Verlauf von dünn nach dick relativ gleichmäßig verläuft. Dies geschieht mit Hilfe einer linearen Funktion. Wenn dieser Verlauf mehr als die standardmäßigen 20% von den Idealwerten der linearen Funktion abweicht (max bend tolerance[%]), wird das Objekt verworfen. In Abb. 6.7 sieht man die gemessenen Höhenunterschiede an

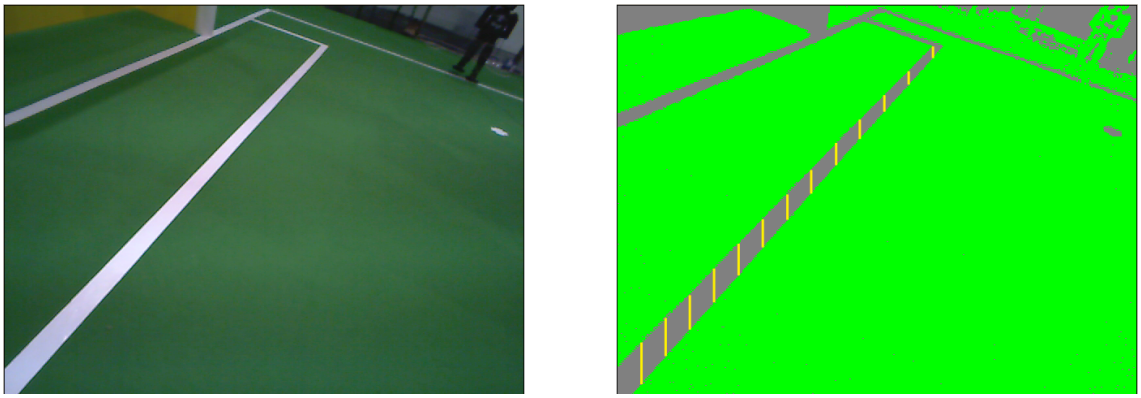

Abbildung 6.7.: Links: Kamerabild - Rechts: gemessene Höhenunterschiede an verschiedenen x-Positionen

verschiedenen x-Positionen und deren Verlauf. Im Beispiel ist die gemessene Höhe in der Ferne 15 Pixel, während sie in der Nähe 50 Pixel beträgt.

Beim letzten Schritt wird nun die korrekte Farbe des Objekts überprüft. Dazu wird zuerst über die Menge aller YUV-Werte der Mittelpunkte (in y-Richtung) der Median-Wert berechnet. Dann wird überprüft, ob dieser Median-Wert der Farbe weiß nahe kommt. Typisch für weiß ist eine relativ hohe Helligkeit, also ein hoher Y-Wert. Die U- und V-Werte sind bei einem idealen Weiß im Zentrum, bei einem unsigned 8 Bit-Wert also bei 128. Die Schwellwerte sind standardmäßig auf einen Mindest-Y-Wert von 160 (*min Y value*) und beim U- und V-Wert auf eine maximale Abweichung von 30 (*max U diff from 128* bzw. *max V diff from 128* - bezogen zum Zentrum, also zu 128) festgelegt.

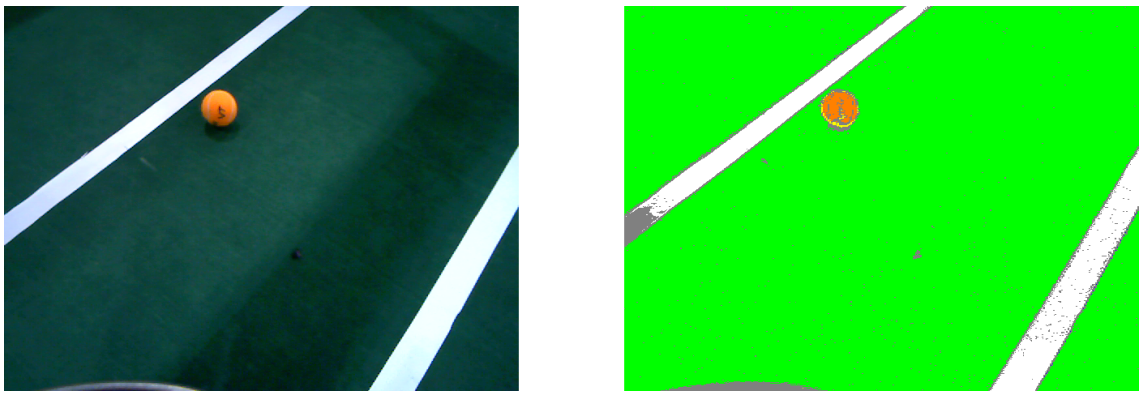

Abbildung 6.8.: Links: Kamerabild - Rechts: segmentiertes Bild nach Ausführung von FINDLINES

Schließlich werden die YUV-Werte aller Pixel mit dem Median-Wert verglichen. Falls

die Distanz eines Pixels kleiner als *max distance* ist, bekommt dieser die Bewertung „guter Pixel“. Wenn nun auch die prozentuale Anzahl der gut bewerteten Pixel größer als *minGoodPixel [%]* ist, werden die Pixel nun endgültig als weiß definiert und entsprechend segmentiert.

In Abb. 6.8 ist ein Beispiel für die automatische Linienerkennung zu sehen, nachdem die anderen Objekte bereits erkannt wurden.

6.4. Tor-Erkennung

6.4.1. Grundlegendes

Die Tore sind wichtiger Bestandteil beim RoboCup. Wie beim Vorbild, dem Fußball für die menschliche Spezies, müssen auch beim RoboCup in der HumanoidLeague die Bälle mit dem Fuß in das gegnerische Tor befördert werden.

Um die beiden Tore leicht unterscheiden zu können, sind sie unterschiedlich gefärbt: eines blau, das andere gelb. Die Farbtöne sind die gleichen, die auch für die Landmarken verwendet werden.

Die zuverlässige Erkennung der Tore ist wichtig, da die humanoiden Roboter der Darmstadt Dribblers so programmiert sind, dass sie nur auf das gegnerische Tor schießen, wenn es auch kurzzeitig vorher gesehen wurde. Außerdem sind erkannte Tore für die Selbstlokalisierung sehr hilfreich. Da die Linien auf dem Feld alle achsensymmetrisch zur Mittellinie verlaufen, kann der Roboter ohne erkannte Tore und Landmarken nämlich nicht erkennen, auf welcher Seite des Spielfeldes er sich momentan befindet.

6.4.2. Realisierung von FINDGOAL

Das Bild wird zu Beginn zeilenweise nach gelbähnlichen bzw. blauähnlichen Pixeln abgesucht. Dazu werden gewisse Grenzen für die Y-, U- und V-Farbwerte der Pixel überprüft. Für einen gelbähnlichen Pixel dürfen standardmäßig der U-Wert höchstens 120 (*max U value*), die Summe von Y und U maximal 228 (*max YU value*) und der V-Wert muss mindestens 128 (*min V value*) betragen. Beim blauähnlichen Pixel dürfen standardmäßig der Y-Wert höchstens 100 (*max Y value*), der V-Wert maximal 128 (*max V value*) und der U-Wert muss mindestens 140 (*min U value*) betragen.

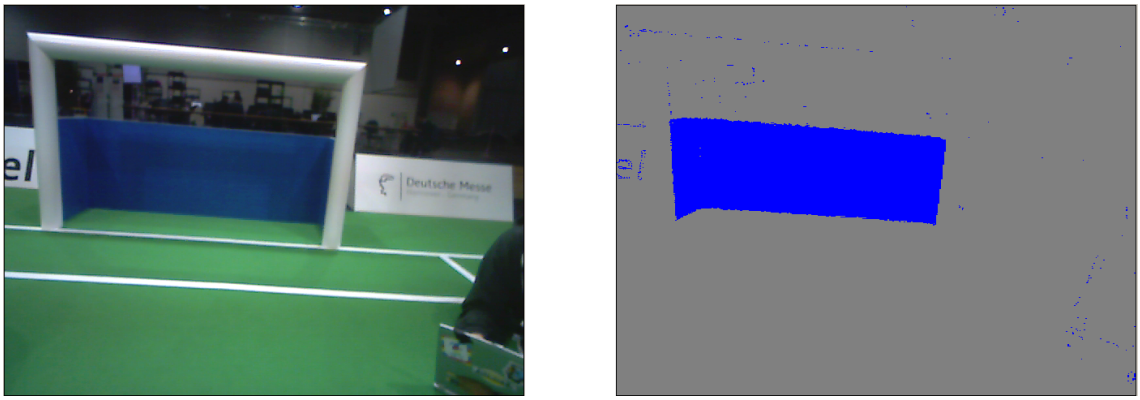

Abbildung 6.9.: Segmentiertes Bild nach Ausführung von FINDGOAL beim blauen Tor

Wird nun eine Kette hintereinander folgender Pixel in gelb- bzw. blauähnlicher Farbe gefunden, die standardmäßig mindestens ein Viertel der Bildbreite (*min goal width [%]*) einnimmt, wird die Untersuchung eingeleitet, ob es sich um ein Tor handeln könnte.

Dazu wird der Mittelpunkt zwischen dem ersten und letzten gefundenen gelb- bzw. blauähnlichen Pixel bestimmt. Anschließend wird eine Bildzeile tiefer vom dem Mittelpunkt aus in x-Richtung nach links und rechts wiederum nach hintereinander folgenden gelb- bzw. blauähnlichen Pixeln gesucht. Dies wird so lange wiederholt, bis kein gelb- bzw. blauähnlicher Pixel in der nächst tieferen Zeile gefunden wird oder die letzte Bildzeile erreicht ist. Die Anzahl der hintereinandergefundenen Zeilen muss dabei standardmäßig mindestens 20 Zeilen betragen (*min goal height*).

Dabei wird die Anzahl der hintereinander folgenden gelb- bzw. blauähnlichen Pixeln in jeder Zeile im Speicher behalten und miteinander verglichen, ob es jeweils etwa gleich viele Pixel sind. Dazu wird zuerst der Median-Wert der gemessenen Breiten gebildet. Danach werden die gefundenen Zeilen gezählt, deren gefundene Pixelanzahl zu stark vom Median-Wert abweichen. Der Schwellwert, ob eine Zeile zu stark abweicht, ist standardmäßig 25% des Median-Wertes (*max bad width tolerance [%]*). Die Anzahl der zu stark abweichenden Zeilen darf standardmäßig höchstens 20% (*max bad width [%]*) betragen.

Neben den korrekten geometrischen Ausmaßen wird auch überprüft, ob das Objekt einen gleichmäßigen Farbverlauf hat. Dazu wird bei jedem Pixel innerhalb des Objekts die Farbraum-Distanz zum linksbenachbarten Pixel gemessen. Diese darf standardmäßig höchstens 70 Einheiten betragen (*max bad distance*), sonst wird der Pixel als „schlecht“ bewertet. Der prozentuale Anteil der schlecht bewerteten Pixel bezogen auf die Pixenanzahl des gesamten Objektes darf standardmäßig 30% (*max bad distance [%]*) nicht überschreiten.

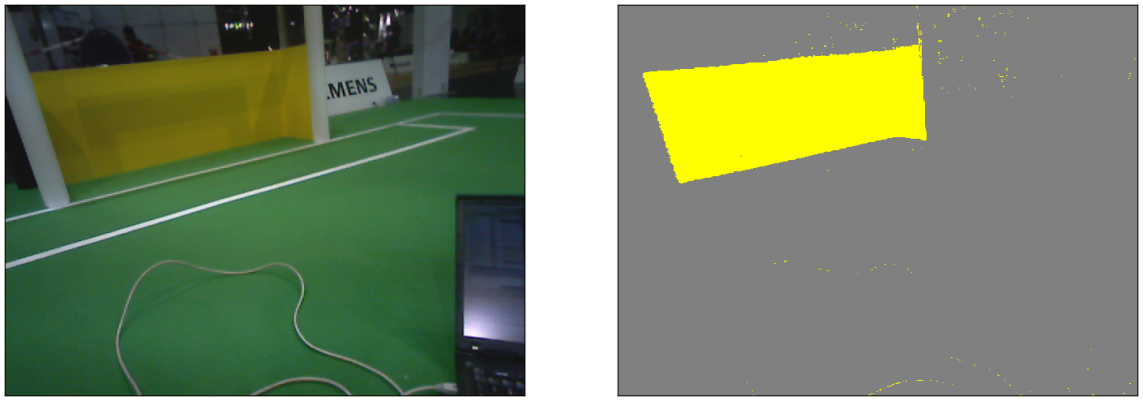

Abbildung 6.10.: Segmentiertes Bild nach Ausführung von FINDGOAL beim gelben Tor

Zuletzt wird noch eine globale Überprüfung durchgeführt, ob der Farbwert des gesamten Objekts etwa gleich ist. Dazu wird zuerst der Median-Farbwert aller Pixel des Objektes berechnet. Anschließend werden die Pixel gezählt, die standardmäßig nicht mehr als 100 Einheiten (blau) bzw. 200 Einheiten (gelb) (*max median distance*) vom Median-Farbwert abweichen. Außerdem werden diese Pixel zur eventuell späteren Klassifikation zu gelb bzw. blau markiert. Die Anzahl dieser Pixel darf standardmäßig 60% (*min good pixel [%]*) nicht unterschreiten, bezogen auf alle Pixel des Objektes.

Wenn alle Voraussetzungen erfüllt sind, werden die im letzten Schritt markierten Pixel schließlich gelb bzw. blau gefärbt. Falls andersklassifizierte Pixel im Tor auftreten, werden diese überschrieben. Insbesondere beim gelben Tor ist dies sinnvoll, da dort unter Umständen orangeklassifizierte Pixel auftreten können.

Beispiele für die automatische Torerkennung sind in Abb. 6.9 und Abb. 6.10 zu sehen.

6.5. Landmarken-Erkennung

6.5.1. Grundlegendes

Bis einschließlich 2007 gab es beim RoboCup in der Humanoid KidSize-Liga vier so genannte Landmarken an den Ecken des Spielfeldes. Sie waren zylinderförmig, 80cm hoch und farblich markant gekennzeichnet. Es gab zwei Varianten: eine hatte je 26,7 cm hohe Streifen in blau-gelb-blau, die andere in gelb-blau-gelb.

Seit dem RoboCup 2008 gibt es nur noch zwei Landmarken, die sich links und rechts am Rand des Spielfeldes in Höhe der Mittellinie befinden. Der Durchmesser, die Form und das Farbschema sind gleich geblieben, nur die Höhe hat sich auf 60 cm reduziert. Die drei Farbteile blau-gelb-blau bzw. gelb-blau-gelb sind wieder jeweils gleich groß, so dass jeder Farbabschnitt nun 20 cm hoch ist. In Abb. 6.11 sieht man eine von der Roboterkamera aufgezeichnete Landmarke.

Abbildung 6.11.: Landmarke der Humanoid KidSize-Liga

Erkannte Landmarken sind eine entscheidende Hilfe für die Selbstlokalisierung und werden aufgrund ihrer markanten Farbstruktur sehr zuverlässig von der Bildverarbeitung erkannt. Da außerdem seit 2008 von jeder der beiden Landmarken-Typen nur noch eine vorhanden ist, besteht auch keine Verwechslungsgefahr mehr bzw. die Selbstlokalisierung muss die jeweils beiden gleichen Typen nicht mehr logisch verrechnen, wie es noch 2007 der Fall war.

Auch bei der automatischen Farbtabellenerstellung ist die Erkennung der Landmarken der zuverlässigste aller Erkenner, da die eindeutige Farbgestaltung von blau-gelb-blau bzw. gelb-blau-gelb nur sehr unwahrscheinlich in ähnlicher Form in der Umgebung auftritt. Und selbst dann wird durch die Überprüfung von gleicher Höhe aller drei Farbabschnitte und gleich bleibender Breite des untersuchten Objektes ein anderes Objekt relativ zuverlässig ausgeschlossen.

6.5.2. Realisierung von FINDPOLE

Bei Ausführung der Landmarken-Erkennung wird das aktuelle Bild zwei Mal abgescannt, einmal für die Erkennung der gelb-blau-gelben und ein zweites Mal für die Erkennung der blau-gelb-blauen Landmarke. Alle gescannten Pixel mit korrektem Farbwert (dazu gleich mehr) werden beim gesamten Ablauf in zwei zweidemensionalen Arrays markiert. Eines dient der lokalen Markierung von gelb und blau für den aktuellen Durchlauf und wird gegebenfalls für die Einfärbung der Pixel verwendet. Das zweite wird global verwendet und zeigt, ob ein Pixel schon einmal untersucht wurde, um ihn nicht ein zweites Mal zu überprüfen.

Die folgende Beschreibung bezieht sich auf die Erkennung der gelb-blau-gelben Landmarke. Für die blau-gelb-blaue Landmarke verhält es sich entsprechend.

Das Scannen geschieht zeilenweise. Dabei wird nach gelbähnlichen Pixeln gesucht. Zur Farbbestimmung werden Schwellwerte für die Y-, U- und V-Farbwerte der Pixel überprüft. Für einen gelbähnlichen Pixel dürfen standardmäßig der U-Wert höchstens 120 (*max U value*), die Summe von Y und U maximal 228 (*max YU value*) und der V-Wert muss mindestens 128 (*min V value*) betragen. Beim ersten Auftreffen auf einen gelbähnlichen Pixel wird die Zeile nach weiteren hintereinander liegenden gelbähnlichen Pixeln abgesucht. Beim ersten Auftreffen auf einen nicht gelbähnlichen Pixel wird der Mittelpunkt in x-Richtung vom zuerst gefundenen und dem zuletzt gefundenen gelbähnlichen Pixel berechnet.

Nun wird das Scannen nach gelbähnlichen Pixeln eine Zeile weiter unten fortgesetzt. In der Annahme, dass man sich ungefähr in der Mitte (in x-Richtung) der Landmarke befindet, wird von hier nach links und nach rechts nach weiteren gelbähnlichen Pixeln gescannt und wieder der Mittelpunkt berechnet.

Die Mittelpunktsbestimmung in jeder einzelnen Zeile ist vorteilhaft, da so die Landmarke auch schief im Bild sein darf (hervorgerufen durch unpräzise Kameraführung bei der Aufnahme des Bildes). Ansonsten würde man bei Einbehaltung des ersten Mittelwertes mit jeder Zeile von der Landmarkenmitte abdriften und eventuell schon vor Erreichen des unteren Endes auf Pixel außerhalb der Landmarke treffen (siehe Abb. 6.12). Zusätzlich wird jeder Mittelpunktswert in einem Array abgespeichert.

Die Suche nach einem gelbähnlichen Pixel in der nächsten Zeile wird solange durchgeführt, bis in einer Zeile beim berechneten Mittelwert in x-Richtung kein gelbähnlicher Pixel auftritt. An dieser Stelle wird dann vom Erreichen des Endes des oberen Landmarken-Farbabschnitts ausgegangen. Ist die Höhe des ersten Farbabschnitts kleiner als 10 Pixel,

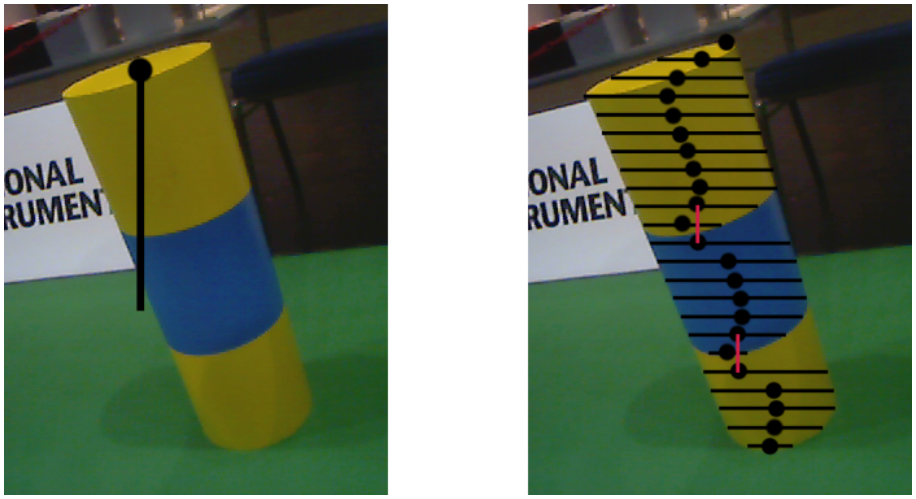

Abbildung 6.12.: Links: Beibehaltung des erstgemessenen Mittelpunktes, Rechts: Aktualisierung des Mittelpunktes

wird die Suche abgebrochen, da eine Untersuchung in dieser Pixelgrößenordnung zu ungenau werden würde.

In der Annahme, dass es sich um eine gelb-blau-gelbe Landmarke handelt, wird in der darauf folgenden Zeile nun nach blauähnlichen Pixeln gesucht. Beim blauähnlichen Pixel dürfen standardmäßig der Y-Wert höchstens 100 (*max Y value*), der V-Wert maximal 128 (*max V value*) und der U-Wert muss mindestens 140 (*min U value*) betragen. Die Suche wird allerdings nicht beim Pixel mit dem x-Wert des letzten Mittelpunktwertes fortgesetzt, sondern mit dem vor 10 Zeilen zuvor berechnete. Dadurch wird ein zu weites Abdriften von der Landmarken-Mitte verhindert, was am Ende eines Farbabschnitts der Landmarke durch Schieflage im Bild typischerweise auftritt.

Da im Bild der Wechsel von gelb auf blau gewöhnlich nicht in einer Zeile geschieht (beim Wechsel tritt eine Mischfarbe auf bzw. auch hervorgerufen durch Kompressionsartefakte), wird beim Auftreffen auf einen nicht-blauähnlichen Pixel in der nächsten Zeile das Objekt nicht gleich verworfen. Stattdessen wird unter Beibehaltung des Mittelpunktwertes in x-Richtung Zeile für Zeile nach unten nach einem blauähnlichen Pixel gesucht. Ist dieser gefunden, beginnt das Scannen des mittleren blauen Landmarken-Farbabschnitts in gleicher Weise wie beim oberen gelben Teil. Entsprechend wird auch der untere Farbabschnitt behandelt.

Ist nun ein Objekt in der gesuchten farblichen Reihenfolge gefunden, wird noch überprüft, ob es auch die Form einer Landmarke hat. Typisch für eine Landmarke ist, dass die

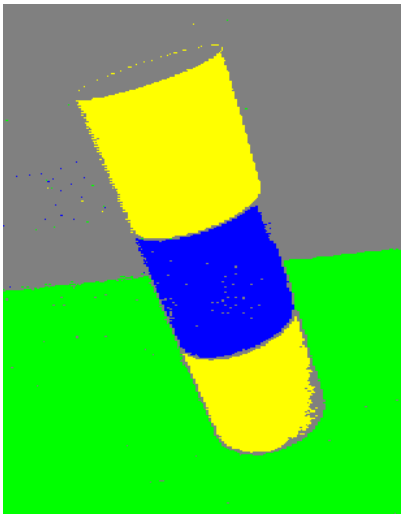

Abbildung 6.13.: Segmentierte Landmarke nach automatischer Erkennung

drei Farbabschnitte gleich hoch, durchgehend gleich breit und ohne Zwischenraum zwischen den drei Teilen sind. Mit Berücksichtigung von etwas Toleranz wird nun überprüft, ob die drei Farbabschnitte ungefähr gleich hoch, gleich breit und die Zwischenräume maximal 5 Pixel hoch sind.

Sind diese Voraussetzungen erfüllt, wird davon ausgegangen, dass es sich beim gefundenen Objekt um eine Landmarke handelt und kann eingefärbt werden. Hierzu wird das zweidimensionale Array verwendet, in dem alle Pixel des gefundenen Objekts mit gelb und blau markiert wurden.

Vor dem Einfärben wird zuerst der Median-YUV-Wert der gelb- und blaumarkierten Pixel getrennt berechnet. Gelb und blau eingefärbt werden schließlich nur die Pixel, die von den Median-YUV-Werten eine bestimmte Distanz im Farbraum (*max pole distance* - standardmäßig 70 Einheiten) nicht überschreiten. Dadurch wird verhindert, dass Pixel, die am Rand der Landmarke liegen und eventuell eine Mischfarbe mit dem danebenliegenden Objekt (z. B. grünes Feld) bilden, mit eingefärbt werden.

In Abb. 6.13 sieht man schließlich ein Beispiel eines segmentierten Bildes nach der automatischen Landmarken-Erkennung.

7. Zusätzliche Funktionen

In diesem Kapitel werden zusätzliche Funktionen der automatischen Farbklassifizierung beschrieben. Sie wurden implementiert, um die durch Ausführung der Erkenner (siehe vorheriges Kapitel) resultierende Farbtabelle weiter zu verbessern. Dazu wird aus bereits klassifizierten Pixeln bzw. Farbwerten abgeleitet, ob weitere Pixel bzw. Farbwerte zu entsprechenden Farbklassen segmentiert werden können.

7.1. Lückenfüllen im Bild

7.1.1. Grundlegendes

Bei der Felderkennung (siehe Kapitel 6.1) werden aus Sicherheitsgründen nur die Pixel zu grün klassifiziert, deren Farbwerte sehr dicht am Median-Farbwert der grünähnlichen Pixel liegen. Das Resultat sind häufig viele unsegmentierte Farbwerte, welche sich im segmentierten Bild als Lücken bemerkbar machen. Ein Feld mit möglichst vollständig grünsegmentierten Pixeln ist aber vorteilhaft, da die Bildverarbeitung für die Erkennung mancher Objekte davon abhängig ist.

Um dieses Problem der Lückenbildung bei grünklassifizierten Pixeln in den Griff zu bekommen, wurde der FILLGAPSINIMAGE-Algorithmus implementiert. Er versucht Lücken, die von bereits segmentierten Pixeln mit gleicher Farbklasse umgeben sind, zu erkennen und gegebenenfalls mit dieser Farbklasse auszufüllen.

Ein großes Problem dabei ist, das Einfärben von „falschen“ Lücken zu verhindern. Wenn beispielsweise ein sehr kleines, möglicherweise nur wenige Pixel einnehmendes Objekt auf dem Feld liegt, soll dieses natürlich nicht zu grün klassifiziert werden. Dies lässt sich relativ leicht mit einer Farbraumdistanz-Berechnung umgehen.

Schwieriger wird es, wenn z. B. eine Person mit einer grünen Jacke auf dem Feld steht und auf einem oder mehreren Bildern auftaucht. Wenn die Jacke einen ähnlichen Grünton

wie das Feld aufweist, ist es gut möglich, dass auf der Jacke lokal viele grünsegmentierte Pixel gehäuft auftreten.

Bei einer reinen lokalen Messung, ob sich an einer Stelle viele grünklassifizierte Pixel befinden, würden unsegmentierte Pixel auf der Jacke eingefärbt werden, deren Farbwerte eventuell nicht auf dem Feld, sondern an einer anderen Stelle in der Umgebung auftreten. Nun treten dort eventuell wieder gehäuft grünklassifizierte Pixel auf und es werden wieder unsegmentierte Pixel grün gefärbt, die mit der Feldfarbe nur noch wenig zu tun haben. Dies kann leicht in einer Kettenreaktion ausarten, sodass am Ende im schlimmsten Fall der ganze Farbraum und entsprechend auch das ganze Bild grün segmentiert sind.

Um dies zu verhindern, ist eine globale Messung implementiert. Diese überprüft, ob auch auf weitere Entfernungen möglichst viele grünklassifizierte Pixel auftreten. Erst dann kann in Kombination mit der lokalen Messung mit ziemlicher Sicherheit davon ausgegangen werden, dass der zu untersuchende Pixel tatsächlich einer Stelle auf dem Spielfeld entspricht.

7.1.2. Realisierung von FILLGAPSINIMAGE

Erste Voraussetzung ist, dass sich viele grünklassifizierte Pixel im Bild befinden. Ansonsten wird der Algorithmus gar nicht erst ausgeführt. Als Schwellwert sind standardmäßig 40 Prozent Grünanteil (*min green in image [%]*) eingestellt.

Um Pixelkandidaten zu finden, wird dann eine globale Suche über das Bild durchgeführt. Dabei wird jedem unsegmentierten Pixel eine Punktzahl zugewiesen.

Diese Punktzahl berechnet sich folgendermaßen: Alle Pixel werden zuerst von links oben beginnend, zeilenweise von links nach rechts aufgerufen und deren Farbklasse abgefragt. Die Punktzahl wird in jeder Zeile mit Null initialisiert. Bei einem grünsegmentierten Pixel wird die Punktzahl um 1 erhöht, bei einem unsegmentierten oder weißen Pixel wird die Punktzahl um *undefined negative score* (standardmäßig 5) verringert und die gerade aktuelle Punktzahl diesem unsegmentierten Pixel zugewiesen. Die Punktzahl gelangt aber nie in den negativen Bereich, da bei einem Wert unter Null die Punktzahl auf Null gesetzt wird. Beim Treffen auf einen andersfarbigen Pixel als grün oder weiß, wird die aktuelle Punktzahl ebenso wieder auf Null gesetzt. Bei einem weißen Pixel wird die Punktzahl nicht zurückgesetzt, weil Linien oft im Bild neben dem Feld auftreten und das Lückenfüllen nicht behindern sollen.

Dieser Vorgang wird in ähnlicher Weise noch drei Mal wiederholt, die Pixel werden allerdings in anderer Reihenfolge gescannt: zeilenweise von rechts nach links, spaltenweise von oben nach unten und spaltenweise von unten nach oben. Die Punktzahlen werden wieder wie bereits beschrieben berechnet und zu den bereits vergebenen Punkten von unsegmentierten Pixeln hinzuaddiert.

Schließlich haben die unsegmentierten Pixel mit hoher Punktzahl mehr grüne Pixel in Richtung aller vier Seiten als welche mit niedriger Punktzahl. Überschreitet die Punktzahl eines Pixels die Mindest-Punktzahl *min green score*, wird dieser Pixel als Kandidat für die weitere Untersuchung ausgewählt.

Da bei höheren Bild-Auflösungen auch höhere Punktzahlen erreichbar sind und in der Praxis auch erreicht werden, da das Verhältnis zwischen grünsegmentierten und unsegmentierten Pixeln in etwa gleich bleibt, passt sich die Mindestpunktzahl proportional zur Pixelanzahl des Bildes an. Die standardmäßige Mindestpunktzahl von 100 bezieht sich auf eine Auflösung von 320x240. Bei einer Auflösung von beispielsweise 160x120 hätte man nur ein Viertel der Pixel und dementsprechend eine Mindestpunktzahl von nur noch 25. Ist die Auflösung hingegen 640x480 hätte man die vierfache Pixelanzahl und eine Mindestpunktzahl von 400.

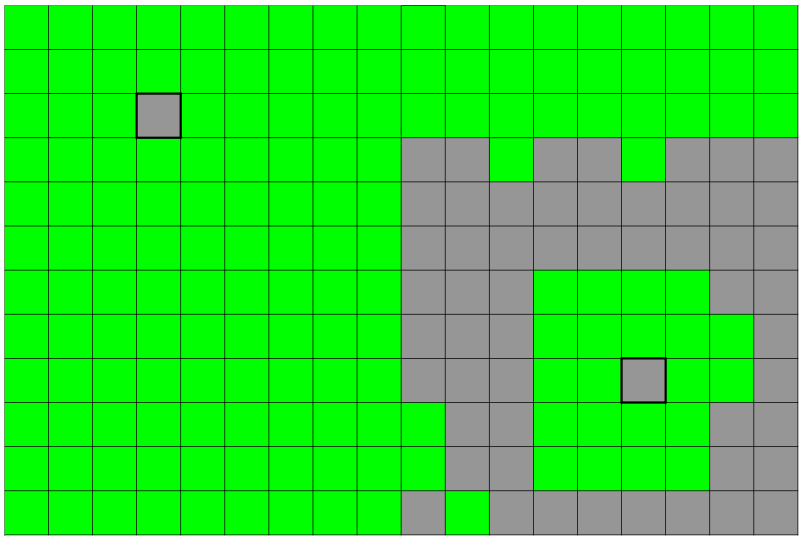

Abbildung 7.1.: Beispiel-Vorlage

In Abb. 7.1 sieht man einen Ausschnitt von 18x12 Pixeln mit grünen und unsegmentierten Pixeln. Insbesondere sollte man die beiden dickumrahmten Pixel beachten. Gemeinsam haben die beiden, dass sie lokal von vielen grünen Pixeln umgeben sind. Im 5x5-Pixelblock um die beiden Kandidaten sind bei dem links oben 24, beim rechts unten

immerhin 21 von 24 möglichen, was durchaus noch akzeptabel wäre (die lokale Messung von 5x5 Pixeln dient hier zur Vereinfachung des Beispiels; im Algorithmus der vorliegenden Arbeit wird, wie weiter unten erwähnt, ein 7x7-Pixelblock für die Messung verwendet).

Global gesehen befindet sich der Kandidat rechts unten jedoch relativ weit entfernt von der Stelle, an der sich das grüne Zentrum befindet. Dies könnte somit ein Fall für die bereits geschilderte Möglichkeit sein, dass es sich bei den grünen Pixeln gar nicht um das Feld, sondern um ein Fremdobjekt handelt. Genau hier greift der globale Überprüfungsalgorithmus und verhindert, dass ein solcher Pixelkandidat eingefärbt wird.

Im Beispiel wird nun der FILLGAPSINIMAGE-Algorithmus mit einem *unsigned negative score* von 3 ausgeführt. Das bedeutet, dass beim Auftreffen auf einen unsegmentierten Pixel drei Punkte subtrahiert und bei einem grünen Pixel ein Punkt hinzuaddiert werden.

In Abb. 7.2 und 7.3 ist die vergebene Punktzahl auf jedem Pixel eingetragen. Der rote Pfeil in der Grafik gibt dabei die Richtung an, in die der Algorithmus gerade ausgeführt wurde.

In Abb. 7.4 ist schließlich die Summe aller unsegmentierten Pixel über die vier Durchläufe eingetragen.

Hätte man die Mindestpunktzahl auf z. B. 10 festgelegt, würde nur der Kandidat mit 17 Punkten links oben für die lokale Überprüfung zugelassen werden. Der Kandidat unten rechts wäre mit 0 Punkten sehr weit davon entfernt. Wie man sieht, treten am Randbereich zwischen dem grünen und dem unsegmentierten Bereich auch viele Pixel mit relativ hohen Punktzahlen von 6 - 7 Punkten auf. Dies ist aber nicht weiter problematisch, da sie spätestens in der folgenden lokalen Überprüfung von der Einfärbung ausgeschlossen werden.

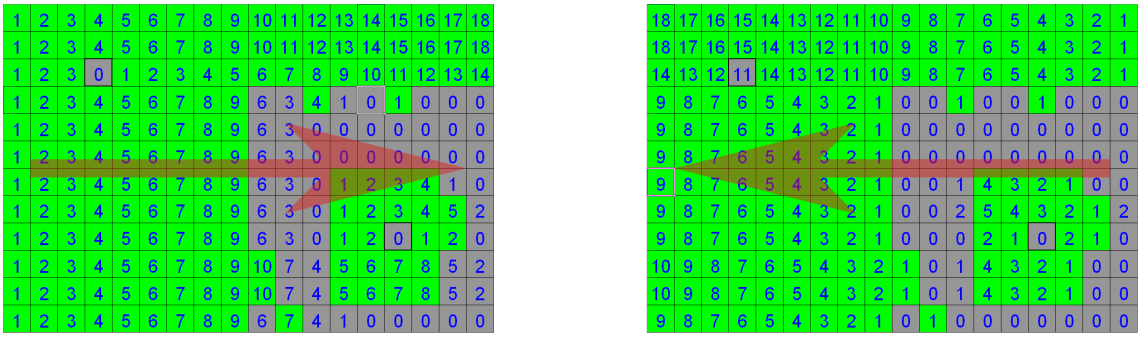

Abbildung 7.2.: FILLGAPSINIMAGE-Algorithmus von links bzw. rechts

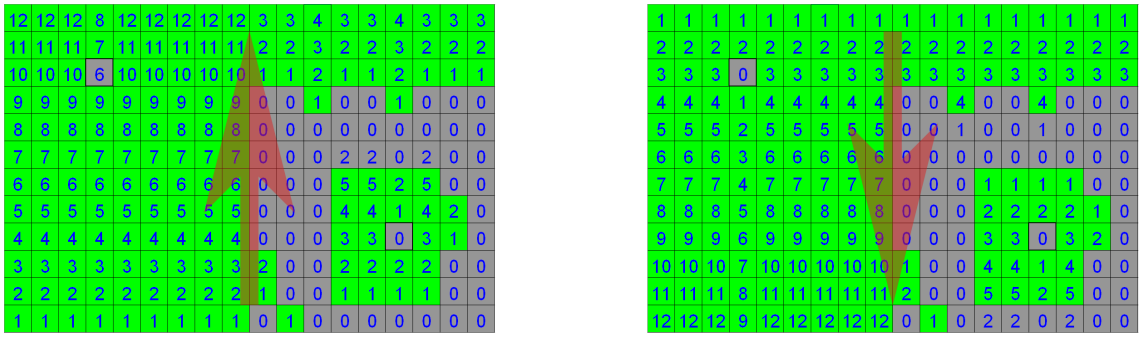

Abbildung 7.3.: FILLGAPSINIMAGE-Algorithmus von oben bzw. unten

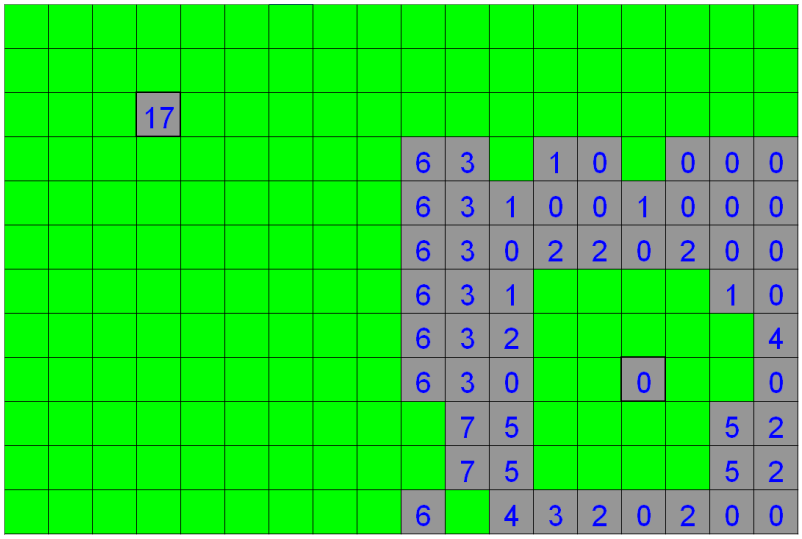

Abbildung 7.4.: FILLGAPSINIMAGE-Algorithmus in der Summe

Die ausgewählten Kandidaten werden nun einer weiteren, diesmal lokalen Untersuchung unterzogen. Hier wird überprüft, ob sich um den zu untersuchenden Pixel möglichst viele grünsegmentierte Pixel befinden. Dazu werden im 7x7-Pixelblock um den zu untersuchenden Pixel die grünsegmentierten Pixel gezählt. Von den 48 maximal möglichen werden standardmäßig 42 als Minimum (*min neighbours 7x7*) gewählt, um den Pixel zur weiteren Untersuchung zuzulassen.

Zuletzt wird noch die Farbwertdistanz des untersuchenden Pixels zum Durchschnittsfarbwert des umliegenden 7x7-Pixelblocks berechnet. Diese darf den standardmäßigen Maximalwert von 0,04 nicht überschreiten. Befindet sich die Distanz unter diesem Wert, sind alle Überprüfungen erfolgreich gewesen und der Pixel wird schließlich grün eingefärbt.

7.1.3. Beispiel in der Praxis

Nun folgt ein Beispiel anhand eines tatsächlich von der Kamera aufgenommenen Bildes.

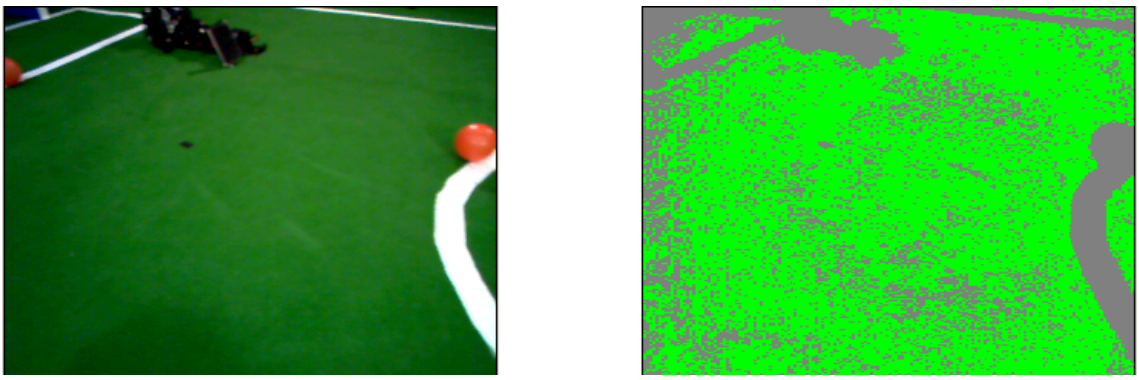

Abbildung 7.5.: Praxisbeispiel: Links das Kamerabild - Rechts das segmentierte Bild

In Abb. 7.5 ist auf der linken Seite das Bild in natürlicher Darstellung und rechts die Darstellung mit segmentierten Farben. So kann es aussehen, wenn nur der FINDFIELD-Algorithmus ausgeführt wird - es sind viele unsegmentierte Lücken im Bild vorhanden.

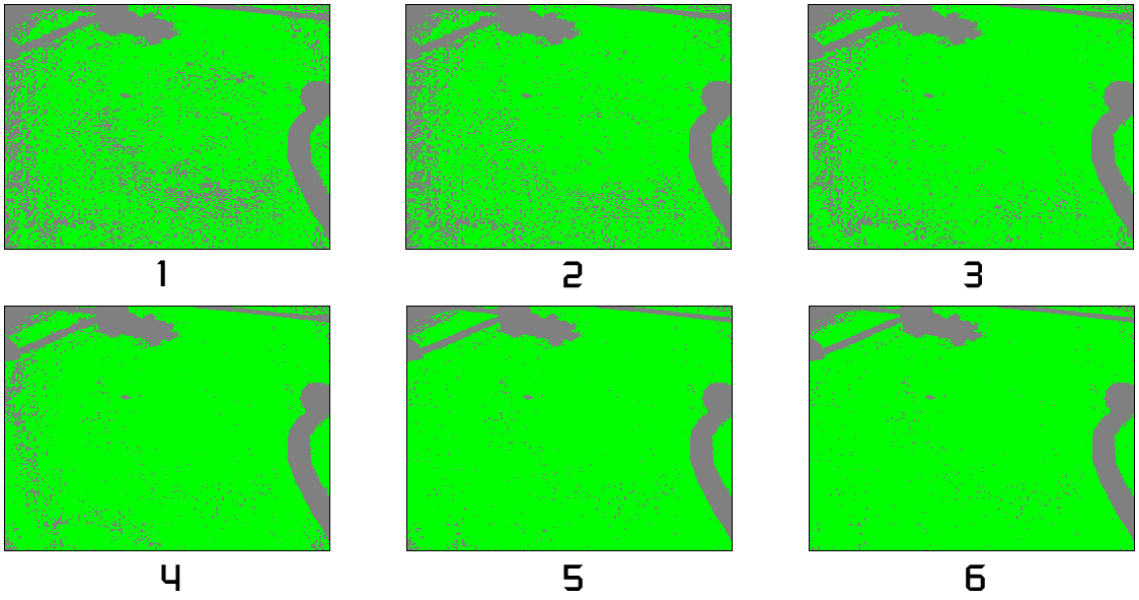

Abbildung 7.6.: FILLGAPSINIMAGE-Algorithmus in wiederholter Ausführung

In Abb. 7.6 wird nun gezeigt, wie die Lücken schrittweise grün gefüllt werden. Bei jedem Füllvorgang entstehen nämlich oft Lücken, die die Füllbedingungen erfüllen, welche es einen Schritt vorher noch nicht taten. So kann es mehrere Durchläufe benötigen

(im Beispiel sechs Durchläufe), bis kein Pixel mehr die Füllbedingungen erfüllt. Im Modus FILL GAPS IN FIELD FULL wird das berücksichtigt, und der FILLGAPSINIMAGE-Algorithmus so oft wiederholt, bis kein Pixel mehr gefüllt wird.

7.1.4. Ergänzungen

In der Praxis greift der FILLGAPSINIMAGE-Algorithmus nur dann, wenn das Feld größtenteils schon durch den FINDFIELD-Algorithmus eingefärbt wurde. Außerdem sollten sich nicht zu viele Fremdgegenstände im Bild befinden, wie es bei Wettbewerben oft der Fall ist. Dann hilft es, die Kamera möglichst nach unten zu richten bzw. den Roboter so zu halten, dass auf zumindest einigen Bildern ausschließlich Feld zu sehen ist. Linien stören dabei nicht so stark. Der Ball sollte sich aber möglichst nicht im Bild befinden, da beim Auftreffen auf orangefarbene Pixel die Punktzahl des globalen Messalgorithmus sicherheitshalber auf Null gesetzt wird.

7.2. Lückenfüllen im Farbraum

7.2.1. Grundlegendes

Im Farbraum nehmen die Farbwerte einer Farbklasse im Idealfall eine ovalförmige Struktur an, wenn alle Pixel eines Objekts der zugehörigen Farbklasse zugeordnet werden. In der Praxis ist es jedoch so, dass manche Farbwerte in der Vorbereitungsphase vor Spielen gar nicht im Bild auftauchen.

Da die Zuordnung von Farbwerten in der Farbtabelle über die Auswertung von Bildern geschieht, können so die Farbwerte, die nicht auf dem Bild erscheinen, auch nicht zugeordnet werden. Im anschließenden Spiel kann es aber passieren, dass dann genau diese nicht zugeordneten bzw. unklassifizierten Farbwerte auftauchen und im schlimmsten Fall dazu führen, dass teilweise Objekte deswegen nicht erkannt werden.

Zur Lösung dieses Problems ist der FILLGAPSINCOLORSPACE-Algorithmus implementiert. Dabei wird in der Farbraum-Umgebung von unsegmentierten Farbwerten nach Farbwerten gesucht, die von einer Farbklasse gehäuft, im Idealfall sogar ganz, umgeben sind. Dies ist dann ein Indiz dafür, dass dieser unsegmentierte Farbwert auch zu der betroffenen Farbklasse gehört, aber noch nicht im Bild aufgetaucht ist oder einfach nicht eingefärbt wurde.

7.2.2. Realisierung von FILLGAPSINCOLORSPACE

In dieser Arbeit werden beim Finden eines unklassifizierten Farbwertes die Farbwerte in direkter Nachbarschaft, also im 3x3x3-Block überprüft. Abzüglich des mittigen Farbwertes sind dies 26 Farbwerte. Falls alle diese 26 Farbwerte der gleichen Farbklasse zugeordnet sind, ist dies ein sehr sicheres Indiz dafür, dass dieser mittige Farbwert auch zu dieser Farbklasse dazugehört und entsprechend eingefärbt werden kann.

Es hat sich gezeigt, dass auch geringere Schwellwerte zu guten Ergebnissen führen. Liegt dieser Schwellwert z. B. bei den standardmäßig eingestellten 11 gleichen Farbwerten im 3x3x3-Block (*min colorspace neighbours*), zeigt dies immer noch das verstärkte Vorhandensein von gleichklassifizierten Farbwerten in der Nachbarschaft. Gerade in Randbereichen können so auch noch Farbwerte der entsprechenden Farbklasse zugeordnet werden. Das Resultat sind dann Bilder, in denen Objekte besonders am Rand auch ausgefüllter erscheinen, also gerade an den Stellen, wo Lückenfüllen im Bild nicht funktioniert.

Beim Lückenfüllen im Farbraum muss aber beachtet werden, dass bei jeder Farbklassifizierung die Gefahr von Fehlpixeln in der Umgebung größer wird. Besonders bei der Ballfarbe Orange kann dies fatal sein, wenn orangeklassifzierte Pixel in der Umgebung gehäuft auftreten und fälschlicherweise als Ball erkannt werden. Dementsprechend ist der Schwellwert für die Nachbarschafts-Farbwerte für Orange standardmäßig mit 13 Farbwerten etwas höher angesetzt als bei den anderen Farbklassen.

7.2.3. Beispiele in der Praxis

In Abb. 7.7 ist ein Beispiel für die Auswirkungen des Lückenfüllens im Farbraum gezeigt. Auf dem linken Bild sieht man die Farbwerte, wie sie nach Erstellung einer automatischen Farbtabelle ohne Lückenfüllen im Farbraum den Farbklassen zugeordnet wurden. Besonders bei den gelben und weißen Farbwerten existieren im linken Bereich offensichtlich Lücken, da diese Farbwerte im Bild vermutlich nicht oder nur vereinzelt auftraten.

Nach der Ausführung des FILLGAPSINCOLORSPACE-Algorithmus für alle Farben (Abb. 7.7 rechtes Bild) sieht man besonders an diesen Stellen, dass die Lücken verschwunden sind.

In Abb. 7.8 ist nun ein Beispiel gezeigt, wie sich Lückenfüllen im Farbraum bei neuen Bildern auswirken kann, die nicht für die Erstellung der Farbtabelle verwendet wur-

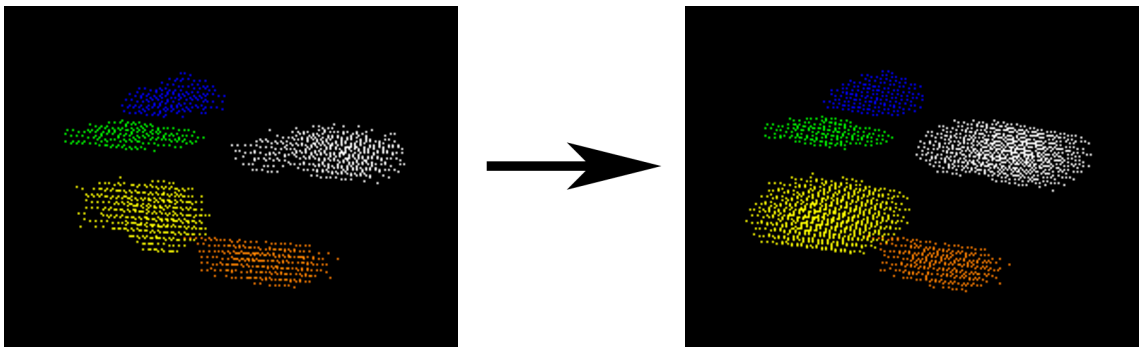

Abbildung 7.7.: Lückenfüllen im Farbraum - Farbraumansicht (vorher und nachher)

den. Links sieht man den vergrößerten Ausschnitt eines klassifizierten Bildes mit einer automatisch erstellten Farbtabelle ohne Ausführung des FILLGAPSINCOLORSPACE-Algorithmus. Auf der rechten Seite ist nun die gleiche Aufnahme nach der Ausführung des Algorithmus zu sehen. Man erkennt, dass die zweite Torlinie deutlich an weißklassifizierten Pixeln dazugewonnen hat. Dies würde mit Sicherheit zu höheren Erkennraten von Linien und damit zu einer besseren Selbstlokalisierung führen.

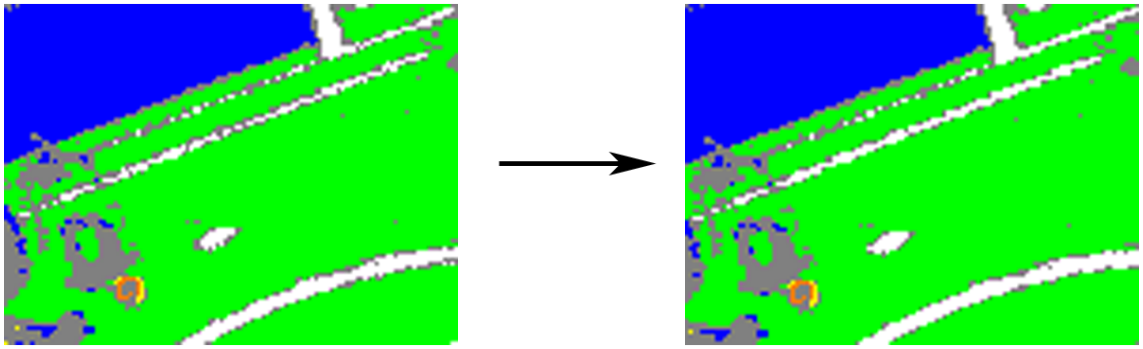

Abbildung 7.8.: Lückenfüllen im Farbraum - Bildansicht (vorher und nachher)

7.3. Löschen von orangeklassifizierten Pixeln im gelben Farbraum

Farbwerte des Balles und des gelben Tores liegen erfahrungsgemäß dicht aneinander. Bei der Farbklassifizierung des Balles können unter Umständen Farbwerte orange segmentiert werden, die auch im gelben Tor auftreten (siehe Abb. 7.9). Im Spielbetrieb kann es dann passieren, dass der Roboter im gelben Tor einen Ball erkennt und darauf zuläuft.

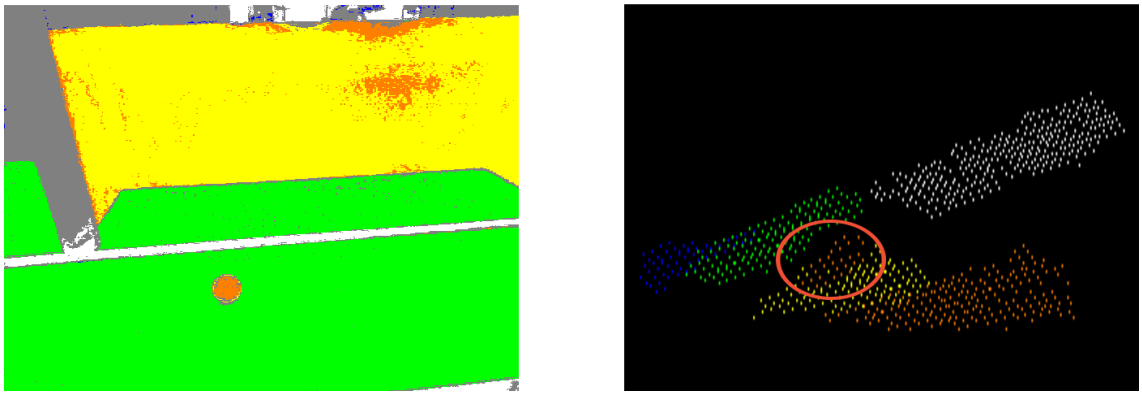

Abbildung 7.9.: Gelbes Tor mit orangeklassifizierten Pixeln

Um dies zu verhindern, werden bei der Torerkennung orangeklassifizierte Pixel überschrieben. An Randbereichen des Tores, besonders beim Übergang zum weißen Pfosten, liegen die Pixel aber außerhalb des Bereichs der Torerkennung und werden somit nicht neu klassifiziert. Falls in diesem Bereich orangeklassifizierte Pixel auftreten, werden sie durch die Torerkennung nicht entfernt (siehe Abb. 7.10).

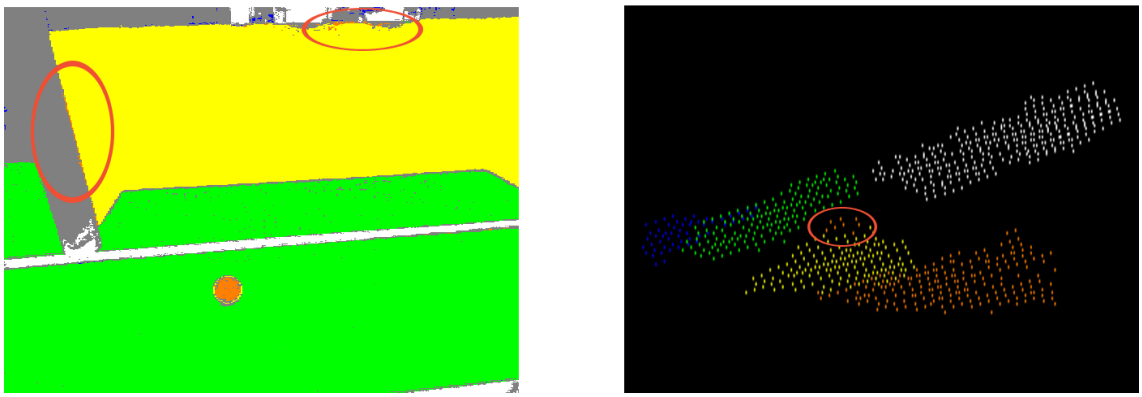

Abbildung 7.10.: Segmentiertes Bild und Farbraum nach Ausführung von FINDGOAL

Um dies zu verhindern, ist der DELBALLINGOAL-Algorithmus implementiert. Dabei wird im Farbraum die Vereinigung von gelb- und weißklassifizierten Farbwerten gebildet und alle orangeklassifizierten Pixel in diesem Bereich gelöscht (siehe Abb. 7.11). Dabei bleiben die korrekten orangeklassifizierten Farbwerte des Balles größtenteils bestehen. Die Toleranz des zu löschenden Bereichs lässt sich im AUTOCOLORTABLE-Dialog mit dem Parameter *del ball in goal distance* anpassen.

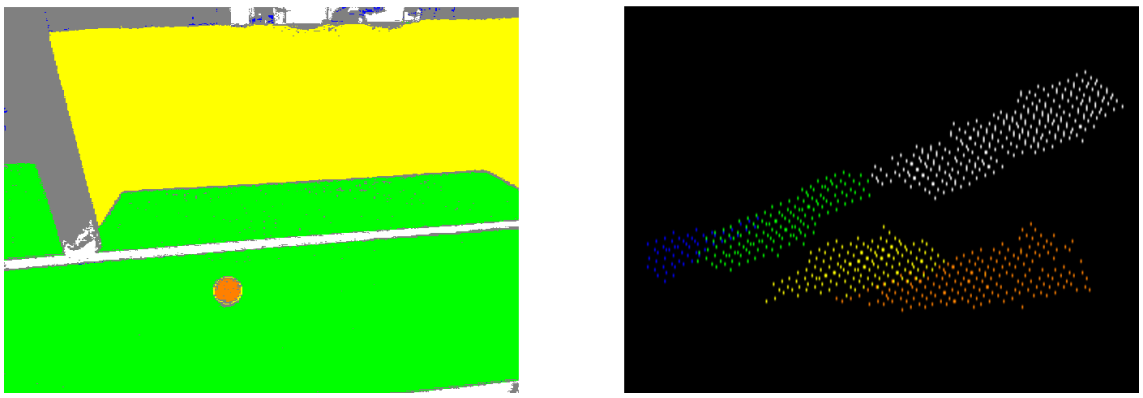

Abbildung 7.11.: Segmentiertes Bild und Farbraum nach Ausführung von DELBALLIN-GOAL

7.4. Schutz vor Puffer-Überlauf bei direkter Roboter-Verbindung

Bei direkter Verbindung mit dem Roboter kann es sein, dass die Bilder schneller ankommen als sie der AUTOCOLORTABLE-Dialog verarbeiten kann, besonders wenn alle Erkenner aktiv sind. Bei aktiviertem LOG FILE MODE wird jedes Bild berechnet und es kann so bei direkter Roboter-Verbindung zum Puffer-Überlauf und zum Absturz kommen.

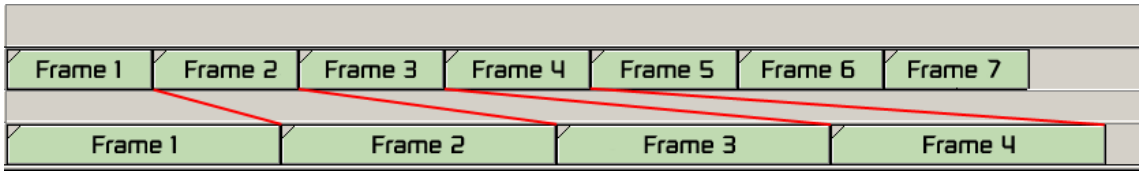

Abbildung 7.12.: Bild-Berechnung bei deaktiviertem Log File Mode und direkter Roboter-Verbindung

Dies kann am Beispiel in der Abb. 7.12 leicht erkannt werden. Die roten Linien zeigen hier deutlich, wie die zu berechnenden Bilder zeitlich immer weiter abdriften.

Ist der LOG FILE MODE abgeschaltet, greift ein Sicherheitsmechanismus. Nachdem ein Bild berechnet wurde, wird die Zeit zum nächsten ankommenden Bild gemessen. Ist die Zeitdifferenz geringer als 10ms, scheint das Bild aus dem Puffer zu kommen, da sonst von unrealistischen 100 fps ausgegangen werden müsste. Nun werden die nächsten ankommenden Bilder solange nicht berechnet, bis die Zeitdifferenz über 10ms gemessen wird. Dies spricht dafür, dass der Bildpuffer leer ist und schließlich das nächste Bild wieder berechnet werden kann.

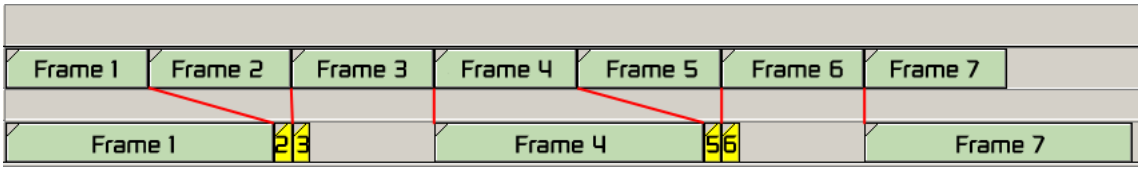

Abbildung 7.13.: Bild-Berechnung bei aktiviertem Log File Mode und direkter Roboter-Verbindung

In Abb. 7.13 ist die Funktionsweise veranschaulicht. Nachdem Frame 1 berechnet wurde, wird Frame 2 nicht berechnet und die Zeit gemessen bis Frame 3 ankommt. Diese beträgt weniger als 10ms, also wird Frame 3 nicht berechnet. Die anschließend gemessene Zeitdifferenz zwischen Frame 3 und 4 beträgt schließlich wieder weit über 10ms, so dass Frame 4 wieder berechnet werden kann. Dies wiederholt sich entsprechend.

8. Generieren einer Farbtabelle in der Praxis

8.1. Vorgehensweise zur Aufnahme von Bildern

Der wichtigste Punkt ist, dass bei der Aufzeichnung bzw. der direkten Verarbeitung der Bilder alle relevanten Objekte im Bild auftauchen. Die Objekte sollten außerdem möglichst aus verschiedenen Perspektiven aufgezeichnet werden, da die Beleuchtung zum Teil unterschiedlich ausfällt.

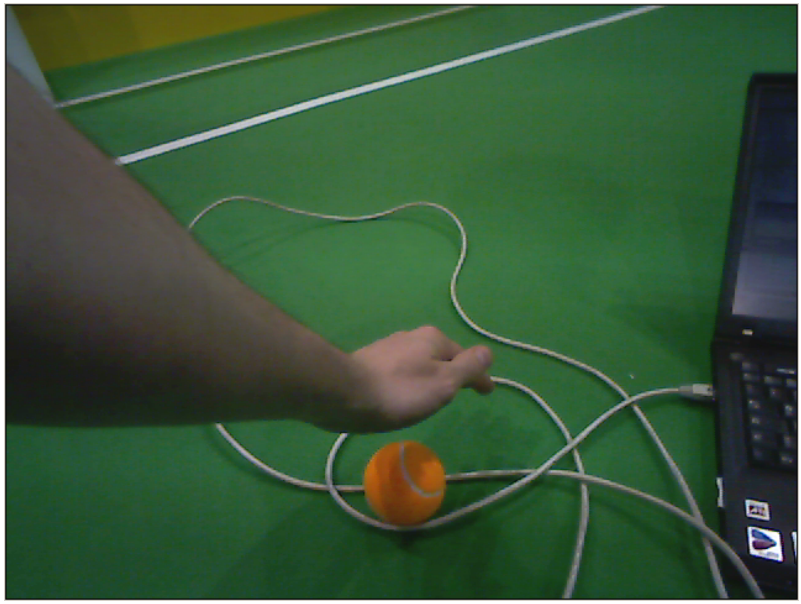

Abbildung 8.1.: Schattenbildung auf dem Ball provozieren

Insbesondere beim Ball ist es wichtig, dass auch Bilder aufgezeichnet werden, wenn der Ball im Schatten liegt. Dies lässt sich leicht provozieren, indem man eine oder beide

Hände vor den Ball hält - natürlich so, dass der Ball noch vollständig im Kamerabild sichtbar ist (siehe Abb. 8.1). Im Spiel selbst kann es nämlich zu Schattenbildungen kommen, wenn ein oder mehrere Roboter nah am Ball stehen. Dann ist es wichtig, dass auch die durch Schattenbildung resultierenden dunkleren orangefarbenen Pixel orange klassifiziert sind, sonst würde der Ball eventuell nicht erkannt werden.

Es hat sich gezeigt, dass eine Log-Datei mit ca. 200 Bildern ausreichend für die Erstellung einer guten Farbtabelle ist.

8.2. Anleitung zur Erstellung einer guten Farbtabelle

Zuerst öffnet man den AUTOCOLORTABLE-Dialog und aktiviert „Full Auto Classification“. Damit werden alle Erkenner und Zusatzfunktionen aktiviert. Die Daten zur Berechnung einer Farbtabelle liefern Bilder, die entweder direkt gesendet oder vorher in einer Log-Datei aufgezeichnet wurden. Bei Verwendung einer Log-Datei aktiviert man am besten noch den „Log File Mode“, da ansonsten nicht alle Bilder berechnet würden.

Möchte man die Entwicklung der Farbtabelle „live“ im COLORTABLE-Dialog mitverfolgen, aktiviert man noch „AUTO SEND COLORTABLE“. So wird bei jedem berechneten Bild die aktuelle Farbtabelle an den COLORTABLE-Dialog gesendet. Die Log-Datei lässt man am besten zwei Mal durchlaufen, da Ball- und Linienerkenner vom grünklassifizierten Feld abhängig sind und dies zu Beginn der Log-Datei wahrscheinlich noch nicht gegeben war.

Nach dem zweiten Durchlauf deaktiviert man „Full Auto Classification“ und kontrolliert im COLORTABLE-Dialog die segmentierten Bilder. Besonderen Wert sollte man auf die Umgebung legen, ob Fehlerkennungen aufgetreten sind. Falls dies der Fall ist, löscht man die betroffene Farbklasse durch Auswahl im AUTOCOLORTABLE-Dialog und Betätigung von „Delete current Color“. Anschließend versucht man, die Erkennung an einem anderen Punkt der Log-Datei zu starten - am besten an einer Stelle, an der das zugehörige Objekt der Farbklasse auftritt. Falls immer noch Fehlerkennungen auftreten, muss das Objekt manuell segmentiert werden.

Sind Objekte nicht vollständig segmentiert, sollte manuell nachgefärbt werden. Besonders beim Ball sollte darauf geachtet werden, dass auch in weiterer Entfernung (wenn der Ball nur noch wenige Pixel im Bild einnimmt) ausreichend orangesegmentierte Pixel vorhanden sind.

9. Ergebnisse

9.1. Erkennraten

Interessant sind die Erkennraten der einzelnen Objekte einer automatisch erstellten Farbtabelle gegenüber einer per Hand optimierten. Zur Auswertung wurden mehrere Log-Dateien aus den German Open 2008 in Hannover[9] und dem RoboCup 2008 in China[10] herangezogen. Die Bildauflösung beträgt jeweils 640x480.

Für die automatisch erstellte Farbtabelle wurde jede Log-Datei zwei Mal durchlaufen. Die per hand optimierten Farbtabellen entstanden auf Basis der automatisch erstellten mit Ergänzungen bei unsegmentierten Pixeln, so gut es möglich war.

Im Mittel erzielten die automatisch erstellten Farbtabellen dabei folgende Erkennraten, verglichen zu einer handoptimierten:

Objekt	Erkennungsrate
Linien	93 %
Bälle	87 %
Tore	98 %
Landmarken	93 %

Die für die Selbstlokalisierung wichtigen Linien, Tore und Landmarken werden im Mittel also zu ca. 95% erkannt. In der Praxis sollte das kaum Auswirkungen zeigen.

Die Ballerkennung liegt mit 87% etwas dahinter. Insbesondere bei Bällen in weiterer Entfernung um die zwei Meter, die bei einer Bildauflösung von 640x480 nur noch wenige Pixel im Bild einnehmen, fehlen der Bildverarbeitung oft die entscheidenden orangesegmentierten Pixel, um einen Ball erkennen zu können.

Grund dafür ist, dass bei weiteren Entfernungen Farbwerte im Ball auftauchen können, die bei näheren Entfernungen nicht auftreten. Die automatische Farbklassifizierung

ist aber nicht für die Erkennung von solch kleinen Bällen im Bild ausgelegt, da bei so geringer Pixelzahl eine Überprüfung auf Rundheit nicht mehr zuverlässig wäre.

Bei wichtigen Spielen ist es deswegen empfehlenswert, fehlende Farbwerte manuell zu ergänzen. Siehe hierzu auch Kapitel 8.2.

Für Testspiele oder bei Demos ist die reine automatische Farbklassifizierung im Normalfall ausreichend gut.

9.2. Performance

Die Performance-Messungen wurden auf einem IBM Thinkpad T43p mit Pentium M Dothan 2,26 GHz und 2GB RAM durchgeführt. Als Bildmaterial dienten mehrere Log-Dateien mit Bildern in der Auflösung 640x480 von den German Open 2008 aus Hannover, auf denen alle relevanten Objekte wie Feld, Linien, Tore, Landmarken und den neuen offiziellen Bällen (orangefarbener Tennisball) zu sehen sind.

Durchschnittlich ergaben sich für die einzelnen Algorithmen folgende Zeiten:

Algorithmus	Zeit
FINDFIELD	80 - 100 msec
FINDBALL	70 - 90 msec
FINDLINES	50 - 80 msec
FINDGOAL	15 - 20 msec
FINDPOLE	25 - 80 msec
FILLGAPSINIMAGEFULL	60 - 80 msec
FILLGAPSINCOLORSPACE	70 - 80 msec
DELBALLINGOAL	10 - 15 msec
Summe	380 - 535 msec

Auf einem aktuellen Rechner können durchschnittlich also etwa zwei Bilder pro Sekunde in der Auflösung 640x480 berechnet werden. Die benötigte Zeit ist etwa proportional zur Pixelanzahl der Bilder. Mit 320x240 wird beispielsweise nur etwa ein Viertel der Zeit benötigt. Damit könnten auf dem genannten Rechner etwa acht Bilder pro Sekunde berechnet werden.

10. Zusammenfassung und Ausblick

10.1. Zusammenfassung

Die in dieser Diplomarbeit entwickelte Software ermöglicht durch eine vollständig automatisierte Farbklassifikation die automatische Erstellung von Farbtabellen für den RoboCup. Die Berechnungen dazu werden auf einem externen Rechner ausgeführt.

Dabei dienen als verwertbare Daten ausschließlich von der Kamera des Roboters aufgezeichnete Bilder. Da die Farbtabelle das Spiel über statisch ist, dürfen sich die Lichtverhältnisse nicht oder nur in sehr geringem Maße ändern.

Die Erkennung der Objekte ist in den meisten Fällen zuverlässig. Treten doch Fehlerkennungen auf, lassen sich einzelne Farbklassen einfach löschen und sind manuell über den COLORTABLE-Dialog einstellbar.

Die Bedienung des AUTOCOLORTABLE-Dialogs ist einfach und die Verwendung erspart viel Zeit gegenüber der manuellen Erstellung einer Farbtabelle über den COLORTABLE-Dialog.

10.2. Ausblick

Die hier vorgestellte Software ist für den Betrieb auf einem externen Rechner konzipiert. Es handelt sich dabei um ein so genanntes Offline-Verfahren, da das Programm nicht während des Roboter-Betriebs abläuft, sondern die Farbtabelle vorher berechnet wird. Da beim RoboCup für gleich bleibende Lichtverhältnisse in der Humanoid KidSize League gesorgt wird, hat sich dieses Verfahren bewährt.

Momentan wäre die für ein Online-Verfahren nötige automatische Farbklassifizierung auf dem Computer des Roboters nicht praktikabel. Die Rechenleistung ist aufgrund des relativ langsamen AMD Geode-Prozessors mit 500 MHz stark begrenzt und die Rechenkapazität aufgrund anderer laufenden Prozessen schon weitgehend ausgeschöpft. In Zukunft ist jedoch mit schnelleren, ebenso sparsamen Prozessoren zu rechnen wie z. B. dem Intel Atom Prozessor, der bereits vor kurzem auf den Markt gekommen ist. Mit Taktraten von weit über 1 GHz stände einem so mehr als die doppelte Rechenleistung zur Verfügung.

Eine Portierung der automatischen Farbklassifizierung auf die Applikation, was die direkte Ausführung auf dem Computer des Roboters auch während des Spiel-Betriebs möglich machen würde, wäre beim Vorhandensein leistungsfähigerer Hardware denkbar. Dies hätte den Vorteil, dass auch ohne externen Rechner eine passende Farbtabelle für die aktuellen Lichtbedingungen berechnet werden und anschließend sofort verwendet werden könnte. Bei einer Demo auf einem unbekannten Feld könnte so z. B. ganz auf einen externen Rechner verzichtet werden.

Denkbar wäre auch eine Anpassung der Farbtabelle während des Spiels. Damit wäre der Betrieb bei Lichtänderungen möglich, unter Umständen auch unter freiem Himmel. Je nach der zur Verfügung stehenden Rechenleistung könnte auch versucht werden, Erkenner nur teilweise zu aktivieren. Bei leicht wechselnder Helligkeit würde wahrscheinlich beim Feld für eine gute Grünklassifizierung der „Lückenfüllen im Bild“-Algorithmus ausreichen. Auf die eigentliche Felderkennung könnte dann verzichtet werden.

A. Der AUTOCOLORTABLE-Dialog

A.1. Grundlegendes

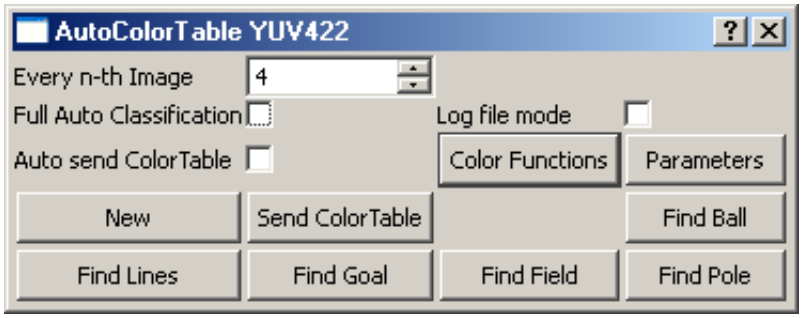

Abbildung A.1.: AUTOCOLORTABLE-Dialog mit Grundfunktionen

In Abb. A.1 sieht man den AUTOCOLORTABLE-Dialog im Standard-Format, wenn er gestartet wird. Es stehen die wichtigsten Grundfunktionen zur Verfügung.

Mit der Schaltfläche „Color Functions“ lassen sich noch weitere Funktionen sichtbar machen. Die Schaltfläche „Parameters“ dient zur Änderung von Parametern der einzelnen Erkenner.

A.2. Grundfunktionen

Every n-th Image
Hier lässt sich einstellen, wie viele Bilder bei direkter Verbindung zum Roboter angefordert werden sollen. Um möglichst viele Bildinformationen in kurzer Zeit zu gewinnen, ist es natürlich sinnvoll, möglichst viele Bilder oder sogar alle Bilder anzufordern. Bei einer Verbindung über Netzwerkkabel, einer Auflösung von 640x480 und teilweise über 10 Bildern pro Sekunde kann es aber leicht zu einer Netzüberlastung kommen, woraufhin

die GUI die Verbindung zum Roboter abbricht. Werte von 3-4 (nur jedes dritte bzw. vierte Bild wird angefordert) sind dann sinnvoll.

Full Auto Classification
Bei Anwählen werden alle Erkenner und zusätzliche Funktionen aktiviert.

Log File Mode
Standardmäßig ist ein Schutz vor Pufferüberlaufen aktiviert, der bei direkter Roboter-Verbindung auftreten kann. Lässt man die automatische Farbklassifizierung über eine Log-Datei laufen, kann man den LOG FILE MODE aktivieren, sodass jedes Bild berechnet wird.

Auto send ColorTable
Ist diese Funktion aktiviert, wird bei jedem ankommenden Bild die aktuelle Farbtabelle an den COLORTABLE-Dialog gesendet. So kann auf dem COLORTABLE-Dialog direkt mitverfolgt werden, wie die segmentierten Bilder aktuell aussehen.

New
Die Farbtabelle wird vollständig zurückgesetzt. Außerdem werden einige Daten neu initialisiert, die im Verlauf der automatischen Farbtabellenerstellung gespeichert werden.

Send ColorTable
Hier kann manuell die Farbtabelle an den COLORTABLE-Dialog gesendet werden.

Find Ball / Lines / Goal / Field / Pole
Find Ball: FINDBALL-Algorithmus wird im aktuellen Bild ausgeführt.
Find Lines: FINDLINES-Algorithmus wird im aktuellen Bild ausgeführt.
Find Goal: FINDGOAL-Algorithmus wird im aktuellen Bild ausgeführt.
Find Field: FINDFIELD-Algorithmus wird im aktuellen Bild ausgeführt.
Find Pole: FINDPOLE-Algorithmus wird im aktuellen Bild ausgeführt.

A.3. Erweiterte Funktionen

In Abb. A.2 ist der AUTOCOLORTABLE-Dialog gezeigt, wenn „Color Functions“ angewählt wurde. Damit werden weitere Einstellmöglichkeiten sichtbar.

Delete current Color
Beim Ausführen von „Delete current Color“ wird die aktuell ausgewählte Farbklasse

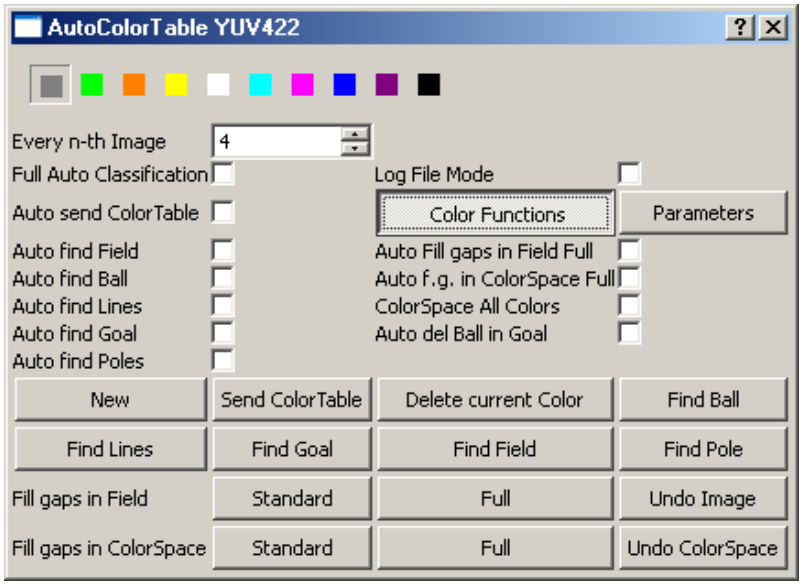

Abbildung A.2.: AUTOCOLORTABLE-Dialog mit erweiterten Funktionen

komplett aus dem Farbraum gelöscht. Außerdem werden gespeicherte Daten der betroffenen Farbe neu initialisiert.

Auto find - Field / Ball / Lines / Goal / Pole
Bei Aktivierung des jeweiligen Objektes wird bei jedem Bild automatisch der entsprechende Erkenner ausgeführt.

Auto Fill gaps in Field Full
Bei Aktivierung wird bei jedem Bild automatisch der FILLGAPSINIMAGE-Algorithmus für die Farbklasse Grün so oft ausgeführt, bis sich keine Änderungen mehr ergeben.

Auto Fill gaps in ColorSpace Full
Bei Aktivierung wird bei jedem Bild automatisch der FILLGAPSINCOLORSPACE-Algorithmus für die aktuell ausgewählte Farbklasse (bzw. für alle Farbklassen wenn „ColorSpace All Colors“ angewählt ist) so oft ausgeführt, bis sich keine Änderungen mehr ergeben.

All colors
Bei Aktivierung wird bei jedem Bild der FILLGAPSINCOLORSPACE-Algorithmus (falls ebenso aktiviert) für alle Farbklassen ausgeführt.

Auto del Ball in Goal
Bei Aktivierung wird bei jedem Bild der DELBALLINGOAL-Algorithmus ausgeführt.

Fill gaps in Field Standard
Bei Anwendung wird beim aktuellen Bild der FILLGAPSINIMAGE-Algorithmus für die Farbklasse Grün ausgeführt.

Fill gaps in Field Full
Bei Anwendung wird beim aktuellen Bild der FILLGAPSINIMAGE-Algorithmus für die Farbklasse Grün so oft ausgeführt, bis sich keine Änderungen mehr ergeben.

Fill gaps in ColorSpace Standard
Bei Anwendung wird beim aktuellen Bild der FILLGAPSINCOLORSPACE-Algorithmus für die aktuell ausgewählte Farbklasse (bzw. für alle Farbklassen wenn „ColorSpace All Colors“ angewählt ist) ausgeführt.

Fill gaps in ColorSpace Full
Bei Anwendung wird beim aktuellen Bild der FILLGAPSINCOLORSPACE-Algorithmus für die aktuell ausgewählte Farbklasse (bzw. für alle Farbklassen wenn „ColorSpace All Colors“ angewählt ist) so oft ausgeführt, bis sich keine Änderungen mehr ergeben.

Undo Image
Bei Anwendung werden die letzten Farbklassifikationen auf Bildebene (Objekterkenner und Lückenfüllen im Bild) rückgängig gemacht.

Undo ColorSpace
Bei Anwendung wird die letzte Ausführung des FILLGAPSINCOLORSPACE-Algorithmus rückgängig gemacht.

B. Parameter des AUTOCOLORTABLE-Dialogs

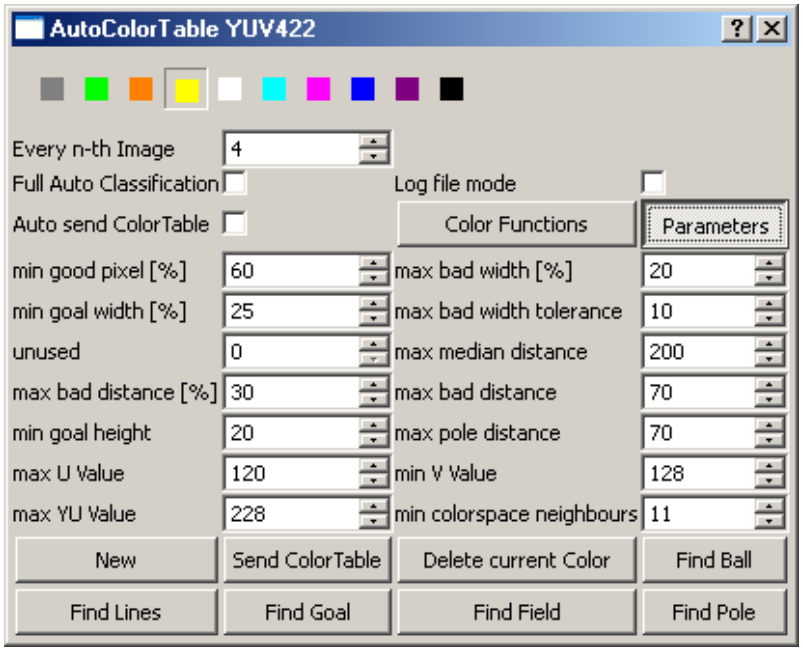

Abbildung B.1.: AUTOCOLORTABLE-Dialog mit Parameter-Einstellungen

In Abb. B.1 ist der AUTOCOLORTABLE-Dialog gezeigt, wenn die Schaltfläche „Parameters“ angewählt wurde. So können für die gerade ausgewählte Farbklasse bzw. dem zugehörigen Erkenner Parameter angepasst werden.

B.1. Feld-Parameter für „Find Field“

min good pixel[%] (Standard: 75)
minimaler Prozentsatz an gut bewerteten Pixeln

max distance (Standard: 80)
maximale Distanz vom Median-Farbwert aller Pixel des unteren Bilddrittels, um den aktuellen Pixel als „gut“ zu bewerten

max distance to green (Standard: 100)
maximale Distanz zum Median-Wert von grünsegmentierten Pixeln im aktuellen Bild, falls es mehr als 50 von diesen Pixeln gibt; ansonsten die maximale Distanz vom zuletzt gespeicherten grünen Median-Farbwert, um einen Pixel als grünähnlich zu bewerten

max U Value (Standard: 140)
maximaler U-Wert des Median-YUV-Farbwerts

max V Value (Standard: 120)
maximaler V-Wert des Median-YUV-Farbwerts

B.2. Feld-Parameter für „Fill Gaps in Field“

min green score (Standard: 100)
Mindestpunktzahl für einen unsegmentierten Pixel für die Grünfärbung, bezieht sich auf eine Auflösung von 320x240 und ist proportional zur Pixelanzahl des Bildes

min green in image [%] (Standard: 40)
Mindestprozentsatz an grünsegmentierten Pixeln im ganzen Bild

undefined negative score (Standard: 5)
Anzahl Punkte die von der aktuellen Punktzahl beim Auftreffen auf einen weißen oder unsegmentierten Pixel subtrahiert wird

max median distance (Standard: 100)
maximale Distanz des aktuellen Pixels zum Median-Farbwert aller grünsegmentierten Pixel im aktuellen Bild

min neighbours 7x7 (Standard: 42)
minimale Anzahl an grünsegmentierten Pixeln in der 7x7 Pixelblock-Nachbarschaft

max gaps distance (Standard: 40)
maximaler Abstand des aktuellen Pixels zum durchschnittlichen Farbwert der grünsegmentierten Pixel in der 7x7 Pixelblock-Nachbarschaft

B.3. Ball-Parameter

max average distance (Standard: 100)
maximale Distanz zum durchschnittlichen YUV-Wert aller Pixel des Objekts

max distance (Standard: 80)
maximale Distanz zum Farbwert des linken Nachbarpixels

min good pixel [%] (Standard: 60)
minimaler Prozentsatz an gut bewerteten Pixeln

min ball size (Standard: 20)
Mindestanzahl an Pixeln, die bei der Kreuzüberprüfung zu Beginn aufgerufen werden müssen

circle tolerance (Standard: 70)
Toleranz für die Rundheit des Objekts

del ball in goal distance (Standard: 200)
Umso höher der Wert, desto mehr orange wird im Farbraum in der Nähe von gelb gelöscht

fill only xx % of radius (Standard: 50)
nur der angegebene Anteil des Ballradius wird eventuell orange klassifiziert

max U Value (Standard: 110)
maximaler U-Wert des durchschnittlichen YUV-Farbwerts

min V Value (Standard: 140)
minimaler V-Wert des durchschnittlichen YUV-Farbwerts

B.4. Linien-Parameter

min good pixel [%](Standard: 30)
minimaler Prozentsatz an gut bewerteten Pixeln

min line length (Standard: 30)
Objekt muss in x-Richtung die angegebene Mindestlänge aufweisen

max bad height [%] (Standard: 50)
maximaler Prozentsatz von schlecht bewerteten Höhenmessungen

max bad height tolerance (Standard: 30)
maximale Toleranz der Pixelabweichung vom Median-Höhenwert

max bad y pos [%](Standard: 20)
maximaler Prozentsatz von schlecht bewerteten y-Positionen (verglichen zur Ideal-Linie)

max bad y pos tolerance (Standard: 5)
maximale Toleranz der Pixelabweichung vom idealen y-Wert

min distance to green (Standard: 200)
minimale Distanz des Pixels zum letzt gemessenen grünsegmentierten Pixel

max bend tolerance (Standard: 20)
maximale Toleranz für die Abweichung einer verzerrten Liniendarstellung

max distance (Standard: 200)
maximale Distanz vom Median-Farbwert des Objekts

min Y value (Standard: 160)
minimaler Y-Wert des Median-YUV-Farbwerts

max U diff from 128 (Standard: 30)
maximale Abweichung des U-Werts des Median-YUV-Farbwerts vom Zentrum (entspricht 128)

max V diff from 128 (Standard: 30)
maximale Abweichung des V-Werts des Median-YUV-Farbwerts vom Zentrum (entspricht 128)

B.5. Tor-Parameter

min good pixel [%] (Standard: 60)
minimaler Prozentsatz an gut bewerteten Pixeln

max bad width [%] (Standard: 40)
maximaler Prozentsatz an Pixeln mit schlecht bewerteter Breite

min goal width [%] (Standard: 25)
minimaler Prozentsatz der Torbreite im Verhältnis zur Bildbreite

max bad width tolerance (Standard: gelb-10, blau-25)
maximale Toleranz für die prozentuale Abweichung der Median-Torbreite

max median distance (Standard: 100)
maximale Distanz zum Median-Farbwert

max bad distance [%] (Standard: 30)
maximaler prozentualer Anteil an Pixeln mit zu großer Farbwert-Distanz zum Nachbarpixel

max bad distance (Standard: 70)
maximale Distanz zum Nachbarpixel in x-Richtung

min goal height (Standard: 20)
minimale Tor-Höhe in Pixeln

gelb - ***max U Value (Standard: 120)***
maximaler U-Wert des Median-YUV-Farbwerts

gelb - ***min V Value (Standard: 128)***
minimaler V-Wert des Median-YUV-Farbwerts

gelb - ***min YU Value (Standard: 228)***
minimale Summe von Y- und U-Wert des Median-YUV-Farbwerts

blau - ***min U Value (Standard: 140)***
minimaler U-Wert des Median-YUV-Farbwerts

blau - ***max V Value (Standard: 128)***
maximaler V-Wert des Median-YUV-Farbwerts

blau - ***max Y Value (Standard: 100)***
maximaler Y-Wert des Median-YUV-Farbwerts

B.6. Landmarken-Parameter

max pole distance (Standard: gelb-70, blau-50)
maximale Farbwert-Distanz eines Pixels zum Median-Farbwert der Farbklasse des jeweiligen Landmarken-Teils

Die YUV-Schwellwerte für gelb und blau sind identisch mit den Schwellwerten für die Torerkennung.

B.7. Farbraum-Parameter

min colorspace neighbours (Standard: 11, orange-13)
Für jede Farbklasse lässt sich dieser Parameter getrennt einstellen. Er gibt die Mindestanzahl an Nachbarn mit gleicher Farbklasse im 3x3x3-Block um einen unklassifizierten Farbwert an, die nötig ist, um diesen Farbwert im „Fill gaps in color space“-Algorithmus ebenso zu dieser Farbklasse zu segmentieren.

C. Literaturverzeichnis

[1] AUSTIN VILLA ROBOT SOCCER TEAM: *Austin Villa Robot Soccer Team Webseite:* `http://www.cs.utexas.edu/users/AustinVilla/`, 2008

[2] COMPUTERBASE LEXIKON: *ComputerBase Webseite:* `http://computerbase.de/lexikon/`, 2008

[3] FUJIFILM: *Fujifilm Webseite:* `http://www.fujifilm-digital.de/`, 2008

[4] HEINEMANN, P. ; SEHNKE, F ; STREICHERT, F. ; ZELL, A.: *Towards a Calibration-Free Robot: The ACT Algorithm for Automatic Online Color Training*, University of Tübingen, 2007

[5] JÄHNE, B.: *Digitale Bildverarbeitung*, 2005

[6] NEUROINFORMATICS GROUP: *MiddleSizeLeague Tribots Webseite: http://www.ni.uos.de*, Universität Osnabrück, 2006

[7] NISCHWITZ, A. ; FISCHER, M. ; HABERÄCKER, P.: *Computergrafik und Bildverarbeitung*, 2007

[8] PETTERS, S. ; THOMAS, D.: *RoboFrame - Softwareframework für mobile autonome Robotersysteme*, Technische Universität Darmstadt, Diplomarbeit, 2005

[9] ROBOCUP GERMAN OPEN: *RoboCup German Open Webseite:* `http://www.robocup-german-open.de/`, 2008

[10] ROBOCUP 2008: *RoboCup 2008 in Suzhou, China Webseite:* `http://robocup-cn.org/`, 2008

[11] SOFTPIXEL: *Softpixel Webseite:* `http://softpixel.com/`, 2008

[12] SRIDHARAN, M. ; STONE, P.: *Autonomous Planned Color Learning on a Legged Robot*, The University of Texas at Austin, 2006

[13] WIRTH, N.: *Algorithms + data structures = programs*, 1976