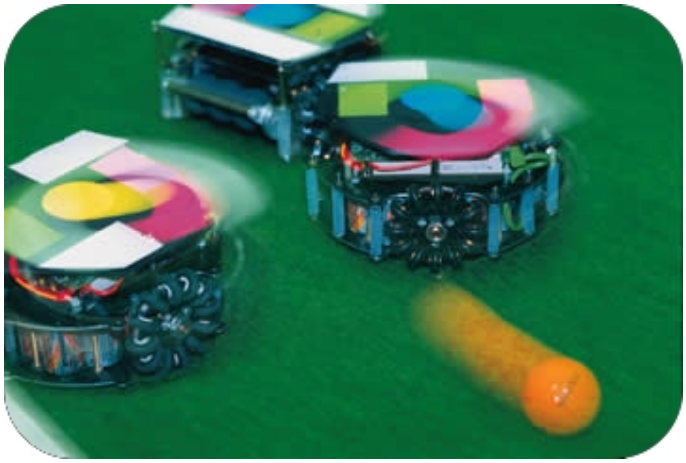
Ein robustes und adaptives Echtzeit-Vision-System für die RoboCup Small-Size Liga

Diplomarbeit in Informatik

vorgelegt von

Mark Simon

geboren 09. Februar, 1976, in Berlin
simon@inf.fu-berlin.de



vorgelegt am

Institut für Informatik
Freie Universität Berlin

Betreuer: Prof. Dr. Raúl Rojas

Berlin, den 6. Februar 2006

Zusammenfassung

In der Small-Size Liga wird das Spielfeld mit Videokameras von oben erfasst. Der Spielball ist ein orangener Golfball, und Roboter werden auf der Oberseite mit farbigen Markierungen versehen. Die Aufgabe eines Vision-Systems ist, die Positionen und Orientierungen aller Roboter und des Balls in Echtzeit im Bildstrom zu finden und in Spielfeldkoordinaten der Steuerungssoftware zur Verfügung zu stellen. Dabei muss es an die 100 Bilder pro Sekunde auswerten und mit Bildartefakten, ungleichmäßiger Beleuchtung und starker Verzerrung der Kamera umgehen können.

In dieser Arbeit beschreibe ich *FU-Vision*, ein robustes und adaptives Computer-Vision System, das ich für die FU-Fighters entwickelt habe. Es ermöglicht die Verarbeitung von Bildströmen zweier Kameras bei einer Bildwiederholungsrate von 53Hz und einer Bildauflösung von 780×582 Pixeln an einem einzigen Rechner bei durchschnittlicher Rechenlast in beinahe allen Situationen. Nach dem Prinzip von variablen Suchrahmen werden Objekte verfolgt, ihre zukünftige Position vorhergesagt, so dass nur kleine Bereiche des Bildes untersucht werden müssen. Mithilfe von qualitätsabhängiger Adaption und adaptiven Farbkarten ist es dem System möglich, sowohl mit starken räumlichen Unterschieden, als auch langsamen zeitlichen Veränderungen der Beleuchtung umzugehen. Es wurde eine Vielzahl verschiedener Robotermodelle implementiert, so dass sich unterschiedlichste Roboter anderer Teams mit höherer Präzision als nur anhand des Teammarkers verfolgen lassen.

Das System kann in wenigen Minuten kalibriert werden. Aufgrund des Prinzips des Trackings und der adaptiven Farbkarten können die Farben sogar während des Spiels automatisch kalibriert werden. Es genügt jede Farbe an einer Stelle einmal per Mausklick zu initialisieren, die restliche Karte wird automatisch erlernt, während sich der Roboter über das Feld bewegt. Zur Kamerakalibrierung genügen wenige Kalibrierungspunkte, um eine akkurate Transformationsfunktion zu bestimmen. Mithilfe analytischer Methoden, die die dreidimensionale Kameraposition anhand der projektiven Transformation ermitteln, kann sowohl der Abstand der Kamera zum Spielfeld, als auch deren Lotpunkt auf dem Feld automatisch bestimmt werden. Schließlich wurde eine automatische Kalibrierungsmethode entwickelt, die die Parameter der Transformationsfunktion anhand der Linien auf dem Feld selbstständig findet und optimiert. Auf diese Weise ist es dem Benutzer möglich die gesamte Kamerakalibrierung nur mit dem Klick eines Buttons durchzuführen.

FU-Vision wurde auf etlichen Turnieren und Präsentationen erfolgreich eingesetzt und hat zu den Erfolgen der FU-Fighters entscheidend beigetragen. Diese sind vierfacher Gewinner der *German Open*, *European Champion 2000*, sowie zweifacher und amtierender Weltmeister. 2004 erhielt das Vision-System zudem den *Engineering Award* in der Small-Size Liga. Schließlich wurde *FU-Vision* 2005 veröffentlicht und anderen Teams zur Verfügung gestellt. So stand auf der Weltmeisterschaft 2005 in Osaka neben den *FU-Fighters* mit *BigRed*, das sehr erfolgreiche Team der Cornell Universität, noch ein zweites Team im Finale, das das *FU-Vision* System verwendete.

Als letztes wurde das System erfolgreich auf einem doppelt so großen Feld getestet. Es wurden keine Veränderungen oder Parameteranpassungen vorgenommen, lediglich die neue Feldgröße eingegeben. Das System verhielt sich dabei ohne erkennbaren Unter-

schied zum jetzigen Feld. Somit wurde gezeigt, dass *FU-Vision* jetzt schon für künftige Feldvergrößerungen vorbereitet ist.

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Berlin, den 7. Februar 2006

[Mark Simon]

Ich wittme diese Arbeit meiner Familie

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation - Die RoboCup Initiative . . . 1
- 1.2 Die RoboCup Small-Size Liga . . . 3
- 1.3 Überblick über die FU-Fighters - Das Small-Size Team der Freien Universität Berlin . . . 4

2 Grundlagen **9**
- 2.1 Farbräume . . . 9
- 2.2 Kameras . . . 13
  - 2.2.1 1 CCD und 3 CCD Kameras . . . 14
  - 2.2.2 Objektive und Linsen . . . 15

3 Problembeschreibung und Bildvorverarbeitung **19**
- 3.1 Problembeschreibung – Warum ist es schwierig ? . . . 19
- 3.2 Bilderfassung und -vorverarbeitung . . . 22
  - 3.2.1 Demosaicing . . . 22
  - 3.2.2 Software-Weißabgleich . . . 23

4 Detektion und Tracking von Farbmarken **25**
- 4.1 Tracking - Verfolgung von bewegten Objekten . . . 25
  - 4.1.1 Positionsvorhersage . . . 25
  - 4.1.2 Variable Suchrahmen . . . 26
- 4.2 Modell von Ball und Robotern . . . 27
- 4.3 Verfolgen von Farbmarken . . . 29
  - 4.3.1 Schritt 1 – Schnelles finden von Hypothesen . . . 29
  - 4.3.2 Schritt 2 – Verifizieren von Hypothesen: Segmentierung . . . 29
    - 4.3.2.1 Farbklassifizierung . . . 30
    - 4.3.2.2 Maß für Kompaktheit . . . 31
    - 4.3.2.3 Qualitätsfunktion . . . 32
    - 4.3.2.4 Anpassen des Modells . . . 33
    - 4.3.2.5 Wahl der zu segmentierenden Region . . . 34
- 4.4 Adaptive Farbkarten . . . 35
  - 4.4.1 Suchen mit Farbkarten . . . 36
  - 4.4.2 Adaptation . . . 36
  - 4.4.3 Initialisierung . . . 36
  - 4.4.4 Pixelmaskierung . . . 38
  - 4.4.5 Diskussion und Ausblick . . . 39

5 Ball- und Robotersuche **41**
5.1 Ballverfolgung . . . 41
5.1.1 Tracken geschossener Bälle . . . 41
5.1.2 Erkennen von Hochschüssen . . . 42
5.2 Robotersuche . . . 43
5.2.1 Lokale Suche . . . 45
5.2.2 Globale Suche . . . 45
5.2.2.1 Stabile Framerate . . . 47
5.2.3 Robotermodelle . . . 48
5.2.3.1 1/2/3-Marker-Modelle . . . 48
5.2.3.2 Orientierungs- und Identifizierungsmodelle . . . 50
5.2.3.3 3Bit/4Bit Schwarz-Weiß Kodierung . . . 50
5.2.3.4 Modelle mit radialem Scan . . . 51
5.2.4 Entfernen gefundener Roboter aus dem Bild . . . 55

6 Kalibrierung und Mehrkamerasysteme **57**
6.1 Kalibrieren einer Kamera über dem Spielfeld . . . 57
6.1.1 Ein erster Ansatz . . . 59
6.1.2 Kalibrieren in zwei Schritten . . . 60
6.1.2.1 Eliminieren der radialen Verzerrung . . . 60
6.1.2.2 Projektive Transformation . . . 61
6.1.2.3 Diskussion . . . 63
6.1.3 Automatische Kalibrierung . . . 63
6.2 Mehrkamerasysteme . . . 64
6.2.1 Suchen in partiellen Ansichten . . . 65
6.2.2 Erstellen eines globalen Weltbilds . . . 67
6.2.3 Pixelmaskierung im Übergangsbereich . . . 68
6.3 *Client-Server*-Architektur . . . 68

7 Messen und Eliminieren der Latenzzeit des Systems **69**
7.1 Messen der Latenzzeit des Systems . . . 69
7.2 Eliminieren der Latenzzeit - Vorhersage aller Objekte . . . 70

8 Ergebnisse und Ausblick **75**
8.1 Messen des Rauschens des Vision-Systems . . . 75
8.2 Zusammenfassung und Ausblick . . . 76

Literaturverzeichnis **79**

1 Einleitung

1.1 Motivation - Die RoboCup Initiative

Ganze 50 Jahre mussten seit der Entwicklung des ersten Computers ins Land streichen, bis im Mai 1997 der amtierende Schachweltmeister Garry Kasparov von IBM's Supercomputer Deep Blue geschlagen wurde, und damit ein zentrales Problem der Künstlichen Intelligenz Forschung (KI) gelöst wurde.

Gleichzeitig musste ein neues Standardproblem gefunden werden, welches den augenblicklichen Stand der Forschung widerspiegelt und Entwicklern auf der ganzen Welt eine Domäne bereitstellt, neue Theorien, Algorithmen und Technologien untereinander gezielt zu vergleichen und auszutauschen.

In diesem Vorhaben wurde 1997 die RoboCup Initiative gegründet [12], die als neue Herausforderung den Roboterfußball propagierte, mit dem klaren Ziel:

Bis 2050 wird ein Team von autonomen, humanoiden Robotern den derzeit amtierenden Fußballweltmeister nach den offiziellen FIFA Regeln schlagen.

Eine solche Zielsetzung mag nach dem Stand heutiger Technologien utopisch erscheinen, dient aber als Richtungsweiser dazu, gemeinsame Zwischenziele zu setzen. Zu diesem Zweck wurden mittlerweile fünf Ligen gegründet mit unterschiedlich vereinfachten Regeln und unterschiedlichem Grad an Autonomie der Roboter:

- *Simulation League*: Es spielen 11 gegen 11 Softwareagenten in einer simulierten Umgebung gegeneinander.
- *F180 Small-Size League*: Teams von fünf Robotern von bis zu 15 cm Durchmesser und 18 cm Höhe treten auf einem 5.5 m $\times$ 4 m Feld gegeneinander an.
- *Sony Four-legged Robot League*: 4 Roboter-Hunde der Firma Sony treten pro Team auf einen 4m $\times$ 6m großen Feld gegeneinander an.
- *F2000 Middle-Size League*: Es spielen 4-6 Roboter pro Team mit maximal 60cm Durchmesser auf einem 8 m $\times$ 12 m großem Feld.
- *Humanoid League*: Humanoide Roboter unterschiedlichster Größe treten in Wettkämpfen wie Elfmeterschießen, Laufen oder auch im Fußballspiel, drei gegen drei Spieler auf einem kleinen Feld, gegeneinander an.

In jährlich ausgetragenen internationalen Meisterschaften können Forscher ihre entwickelten Algorithmen und Systeme in direktem Vergleich erproben und sich untereinander austauschen.

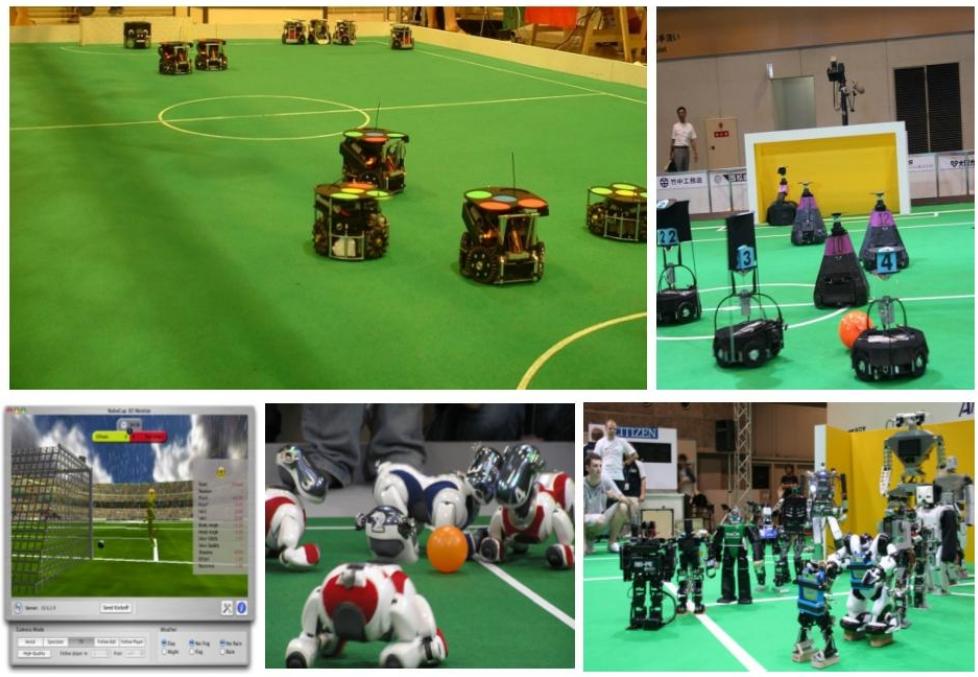


Abbildung 1.1: Die wichtigsten Ligen im RoboCup. Man sieht die Small-Size Liga, Middle-Size Liga, Simulationsliga, die Liga der Roboterhunde, sowie die der Humanoiden.

Die Regeln werden laufend dem aktuellen Stand der Forschung angepasst, so hat sich beispielsweise das Spielfeld der Small-Size Liga von 1999 bis 2004 in seiner Fläche mehr als verfünffacht. Auf diese Weise können Probleme zunächst in einfacheren Umgebungen gelöst, und später auf immer realere Umgebungen übertragen werden.

Natürlich ist es nicht allein das Ziel des RoboCup, intelligente Fußballroboter zu entwickeln, sondern in erster Linie, eine kontrollierte Testumgebung zu schaffen, um die Erforschung zentraler Probleme der KI und Robotik, wie zum Beispiel Objekterkennung, Verhaltenssteuerung, Planung, Koordination im Team, etc. voranzutreiben.

Es gibt eine Vielzahl möglicher technischer Anwendungen, die von Forschungsergebnissen zum Roboterfußball profitieren können. Dazu zählen Haushaltsroboter, Fertigungsautomatisierung, Hilfesysteme für Behinderte, Autopiloten, Warnsysteme für Autofahrer, Fernerkundungsroboter z.B. für den Weltraum und nicht zuletzt intelligentes Spielzeug.

Im Vergleich zum Computerschach ist Roboterfußball ungleich schwerer. Die Umgebung ist in höchstem Grade dynamisch; sowohl die Wahrnehmung der Welt als auch Aktionen, die diese beeinflussen, sind im Gegensatz zu Figuren auf einem Schachbrett kontinuierlich, nicht-deterministisch und fehlerbehaftet. Somit wurde ein würdiger Nachfolger des Computerschachs gefunden, der Forscher aus vielen Bereichen in den kommenden 50 Jahren beschäftigen wird.

Ein wichtiger Aspekt des Roboterfußballs ist die Computer-Vision, das Auge der Roboter. Menschliches Sehen ist ein zentrales Thema, welches in verschiedensten Disziplinen

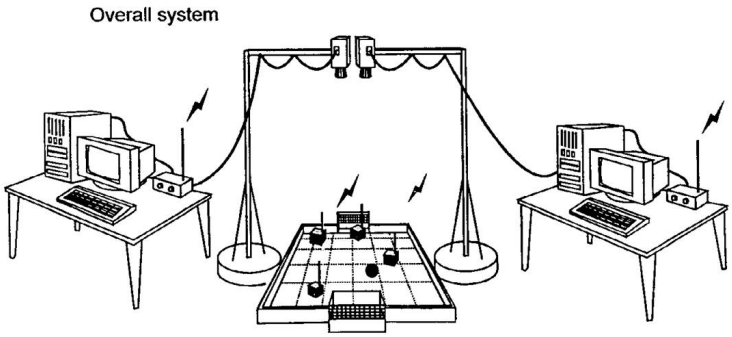


Abbildung 1.2: Schemenhafte Darstellung des allgemeinen Aufbaus in der Small-Size Liga. Kameras hängen über dem Spielfeld und sind an einen externen Rechner angeschlossen. Dieser sendet per Funk Kommandos an die einzelnen Roboter.

der Wissenschaft untersucht wird, und dessen Funktionsweise in großen Teilen immer noch weitgehend unerschlossen ist.

Thema dieser Arbeit war Entwurf und Implementierung eines robusten Echtzeit-Vision-Systems für unser Team die *FU-Fighters* in der RoboCup Small-Size Liga.

Im Folgenden werde ich auf die Small-Size Liga ein wenig genauer eingehen und einen kurzen Überblick der FU-Fighters geben, das Team der Freien Universität Berlin. Kapitel 2 behandelt ein paar Grundlagen, die für das weitere Verständnis der Arbeit hilfreich sind, und in den restlichen Kapiteln beschreibe ich *FU-Vision*, das Computer-Vision-System, das ich für die FU-Fighters entwickelt habe.

1.2 Die RoboCup Small-Size Liga

In der RoboCup Small-Size Liga spielen fünf gegen fünf Roboter von maximal $15cm$ Höhe und $18cm$ Durchmesser auf einem $5.5m \times 4m$ großem Spielfeld. Eine Halbzeit dauert 10 Minuten. Im Unterschied zur Simulationsliga werden in dieser Liga reale Roboter gebaut, die sich in einer nicht simulierten Welt bewegen müssen. Im Gegensatz zur Middle-Size Liga sind diese allerdings nicht vollkommen autonom: Wie in Abbildung 1.2 zu sehen, befestigt jedes Team ein oder mehrere Kameras über dem Spielfeld, die so eine globale Sicht des Geschehens an einen externen Rechner neben dem Feld übertragen. Dieser wertet die Bilder aus und bestimmt für alle Roboter die gewünschten Aktionen, die per Funk an die Roboter gesendet werden.

Die Einbuße an Autonomität ermöglichte, hohe Geschwindigkeiten und Reaktivität bei gleichzeitig besserer Kontrolle zu erreichen, so dass in der Small-Size Liga das dynamischste Spiel im gesamten RoboCup zu sehen ist: Roboter flitzen mit bis zu $3m/s$ über das Feld, passen sich gegenseitig den Ball zu, schießen hohe Freistöße oder mit bis zu $15m/s$ harten Schüssen auf das gegnerische Tor.

Seit den Anfängen 1997 haben sich die Regeln immer mehr realem Fußball angenähert: So gibt es inzwischen direkte und indirekte Freistöße, Eckstöße, Elfmeter, Ein-

würfe, wie auch gelbe und rote Karten. Dabei müssen alle Roboter während der gesamten Spieldauer autonom ohne menschliche Intervention vom Rechner gesteuert werden; Veränderungen dürfen nur in der Halbzeitpause oder dafür vorgesehenen Auszeiten vorgenommen werden.

Im Weiteren werde ich nur auf Regelungen eingehen, die für die Entwicklung eines Vision-Systems von Bedeutung sind, das komplette und jährlich aktuelle Regelwerk ist auf der offiziellen Webseite[1] der RoboCup Organisation zu finden.

Aussehen von Ball und Roboter

Damit die Objekte auf dem Feld leichter im Bild zu finden sind, wird als Spielball ein orangener Golfball verwendet, und Roboter werden auf ihrer Oberseite mit farblichen Markierungen versehen. Pflicht ist dabei der gelbe bzw. blaue Teammarker, der in der Mitte des Roboters anzubringen ist, und anhand dem sich die Zugehörigkeit zum Team erkennen lässt. Zusätzlich zu diesem steht es jeder Mannschaft frei, beliebige weitere Markierungen zu verwenden, sofern sich diese in der Farbe von dem offiziellen Orange, Gelb und Blau ausreichend gut unterscheiden.

Aufhängung der Kamera

Kameras werden an einen Gerüst über dem Spielfeld auf einer Höhe von ca. 4 Metern befestigt. Dabei hängt jedes Team ein oder mehrere Kameras auf, so dass man nicht von einer perfekt zentralen und senkrechten Kamera ausgehen kann – das Vision-System muss mit möglichst beliebiger Aufhängung zurechtkommen.

Beleuchtung

Von anfänglich extra dafür vorgesehenen Strahlern, mit denen das Feld möglichst gleichmäßig auszuleuchten versucht wurde, ist man inzwischen im Reglement übergegangen, die vorhandene Deckenbeleuchtung des jeweiligen Wettbewerbsorts zu verwenden. Es kann bei dem Entwurf eines Vision-System also nicht mehr von einer klar definierten Beleuchtung ausgegangen werden. Zudem wirft das Gerüst zur Befestigung der Kameras harte Schatten auf das Feld, wie in Abbildung 1.1 zu sehen.

1.3 Überblick über die FU-Fighters - Das Small-Size Team der Freien Universität Berlin

Ein lauffähiges System in der Small-Size Liga besteht aus vielen Komponenten, die alle gleichermaßen funktionieren müssen. Dazu gehören ein Computer-Vision-System und *high-level* Steuerungssoftware auf einem externen Rechner, drahtlose Funkübertragung, *low-level* Steuerung auf dem Roboter, Elektronik, wie auch eine leistungsstarke Roboterhardware. Abbildung 1.3 zeigt den Datenfluss zwischen den einzelnen Komponenten. Ich werde nur kurz auf ein paar Komponenten des Systems eingehen, eine aktuelle Beschreibung des Teams findet sich in [9].

[1] www.robocup.org

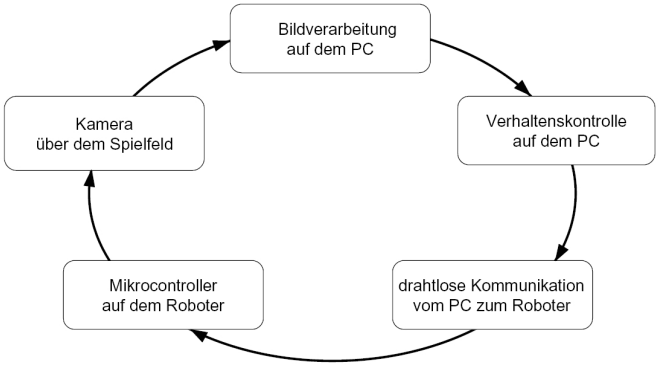


Abbildung 1.3: Darstellung der Systemschleife der FU-Fighters Software. Man sieht die einzelnen Komponenten des Systems und den Datenfluss unter ihnen.

Die Roboterhardware

Der aktuelle Small-Size Roboter verwendet einen omnidirektionalen Antrieb, bestehend aus vier selbst entworfenen Omnirädern, und erreicht eine Maximalgeschwindigkeit von bis zu $3m/s$. Er besitzt einen Magnetschuss, der es ermöglicht, den Ball mit weit mehr als 500-facher Erdbeschleunigung auf bis zu $15m/s$ zu beschleunigen, wie auch einen Hochschuss, der ebenfalls von einer Spule angetrieben wird. Als Energiequelle für den Schuss dienen drei Kondensatoren, die von einer eigenen Schusselektronik in kurzer Zeit geladen werden. Zudem wurde zusätzlich eine Infrarot-Lichtschranke eingebaut, so dass der Schuss nur ausgelöst wird, wenn der Ball sich vor dem Schussmechanismus befindet. Zur besseren Ballführung besitzt der Roboter außerdem einen Dribbelvorrichtung, eine rotierende Rolle, die dem Ball einen *back-spin* verleiht, so dass sich dieser schwer vom Roboter löst.

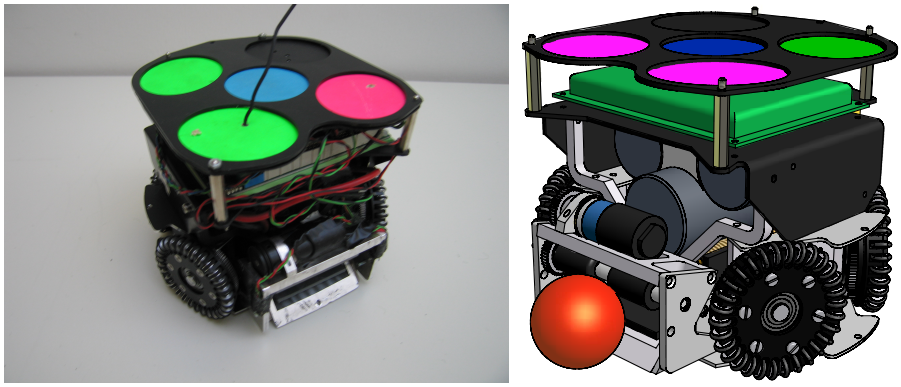


Abbildung 1.4: Die FU-Fighters Roboterhardware 2005. Man sieht die Omniräder, die Elektronik, die Kondensatoren, sowie die Dribbelvorrichtung und den Schussmechanismus.

Die Verhaltenskontrolle

Die Aufgabe der Steuerungssoftware ist, anhand der übertragenen Informationen aus der Computer-Vision, d.h. Position und Orientierung von allen Robotern und dem Ball, die Kommandos für jeden einzelnen Roboter zu bestimmen. Ein solches Kommando enthält den Bewegungsvektor, die Drehrichtung, ob und mit welcher Stärke flach- bzw. hochgeschossen werden und ob der Dribbelmechanismus aktiviert sein soll.

Die Steuerungssoftware der FU-Fighters besteht aus einem komplexen System von Berechnungseinheiten wie Sensoren, Verhalten und Aktoren, die miteinander verbunden sind und Information austauschen. Sensoren beschreiben Charakteristika des momentanen Zustands der Welt, wie z.B. den Winkel zum Ball, das beste Schussziel im gegnerischen Tor, die geschätzte Zeit und Position, bei der ein Roboter den Ball abfangen kann oder die aktuelle Spielsituation, wie Elfmeter oder Freistoß. Sie greifen dabei nicht nur auf Information, die von außen in das System gelangt, sondern auch auf andere Sensoren zu.

Verhalten berechnen, ob sie sich aktivieren, und gegebenenfalls wie und welche Aktoren sie ändern wollen. Dabei können auch mehrere Verhalten gleichzeitig an einem Aktor drehen. Falls dies unerwünscht ist können Hemmungen eingefügt werden, so dass bestimmte Verhalten sich gegenseitig ausschließen.

Über Aktoren können Verhalten miteinander kommunizieren. So kann ein Verhalten *Freistellen* eine geeignete Zielposition setzen, die z.B. vom Pfadplanungsverhalten verwendet wird. Zudem enthalten bestimmte Aktoren die Daten, aus denen letztendlich die Kommandos für jeden Roboter zusammengestellt werden.

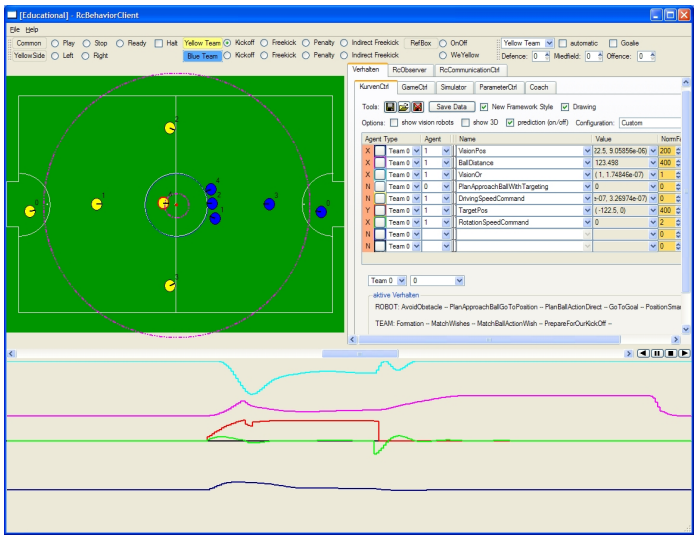


Abbildung 1.5: Screenshot der Verhaltenssoftware. Man sieht eine modellhafte Ansicht des Spielfelds, so wie eine Ansicht, in der Daten aus Sensoren, Aktoren oder Verhalten als Kurven dargestellt werden können. Sämtliche Information wird aufgezeichnet, so dass das Spiel auch im Nachhinein betrachten und analysiert werden kann.

Abbildung 1.5 zeigt einen Screenshot der Verhaltenssoftware. Es werden alle Sensor- und Aktorwerte sowie Verhaltensaktivierungen gespeichert und können auch im Nachhinein betrachtet und analysiert werden.

In einer älteren Version sind diese Berechnungseinheiten hierarchisch in Ebenen angeordnet gewesen, und Informationsfluss über Aktoren fand nur zwischen Ebenen, niemals innerhalb einer Ebene statt. In der gegenwärtigen Version sind sämtliche Sensoren, Aktoren und Verhalten in einem komplexen topologischen Graph angeordnet. Eine umfassende Beschreibung beider Systeme findet sich in [20]. Als Grundlage des Systems diente der *Dual-Dynamics* Ansatz [10].

Die Computer-Vision

Die Entwicklung des Computer-Vision-Systems ist der Bestandteil dieser Arbeit und wird ab Seite 19 im Detail beschrieben. Eine ältere Beschreibung findet sich in [18], [6].

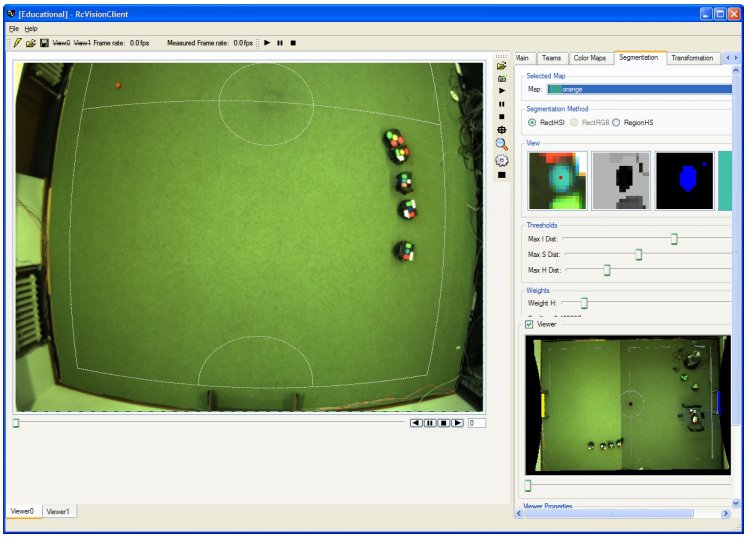


Abbildung 1.6: Screenshot des Computer-Vision-Systems *FU-Vision*.

2 Grundlagen

2.1 Farbräume

Als für den Menschen sichtbares Licht bezeichnet man denjenigen Teil der elektromagnetischen Strahlung, der eine Wellenlänge zwischen 380 und 780 Nanometern besitzt. Licht mit einer Wellenlänge von 380 Nanometern hat dabei eine violette Farbe, Licht mit 780 Nanometern Wellenlänge hat eine rote Farbe. Dazwischen befindet sich das sichtbare Lichtspektrum, bestehend aus theoretisch unendlich vielen Spektralfarben unterschiedlicher Wellenlänge. In der Natur findet sich jedoch normalerweise keine reine Spektralfarbe, sondern Mischungen. Aus beliebiger Mischung einer beliebigen Anzahl von Spektralfarben können also beliebig viele neue Farben entstehen. Weißes Sonnenlicht enthält sämtliche Spektralfarben im sichtbaren Bereich. Eine einzelne Spektralfarbe mit einer ganz bestimmten Wellenlänge lässt sich z.B. mit einem Laser erzeugen.

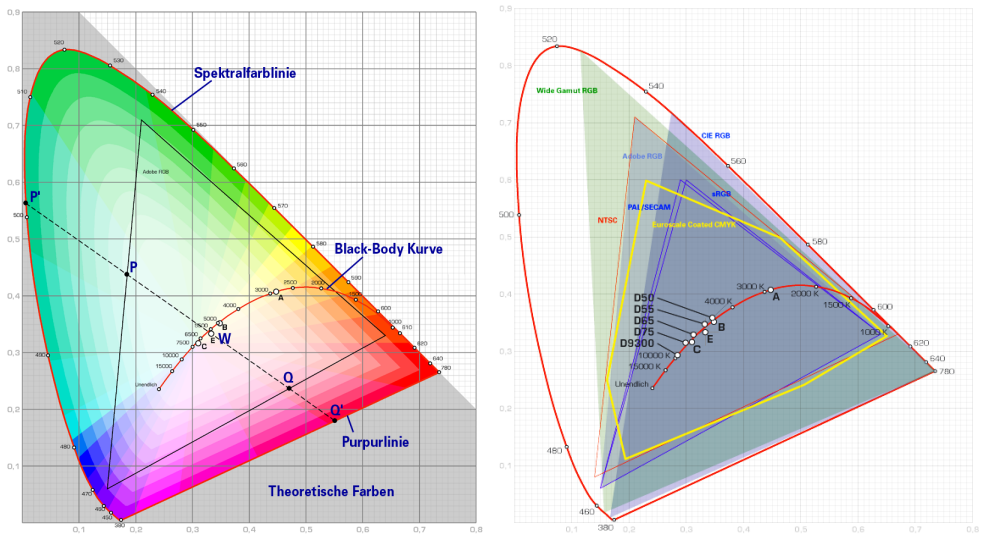


Abbildung 2.1: Die CIE-Normfarbtafel: Alle sichtbaren Farben werden durch die Spektralfarblinie (spektral reine Farben) sowie der Purpurlinie eingefasst. Farbräume bestehen meist aus dreiecksförmigen Unterregionen wie links für den Adobe-RGB Farbraum zu sehen. Im rechten Bild sind weitere RGB-/CMY-Farbmodelle eingezeichnet.

Allerdings können in keinem Computer unendlich viele Farben dargestellt bzw. unterschieden werden. Deshalb werden Farbmodelle verwendet, die einen möglichst großen Bereich der möglichen Farbempfindungen erfassen.

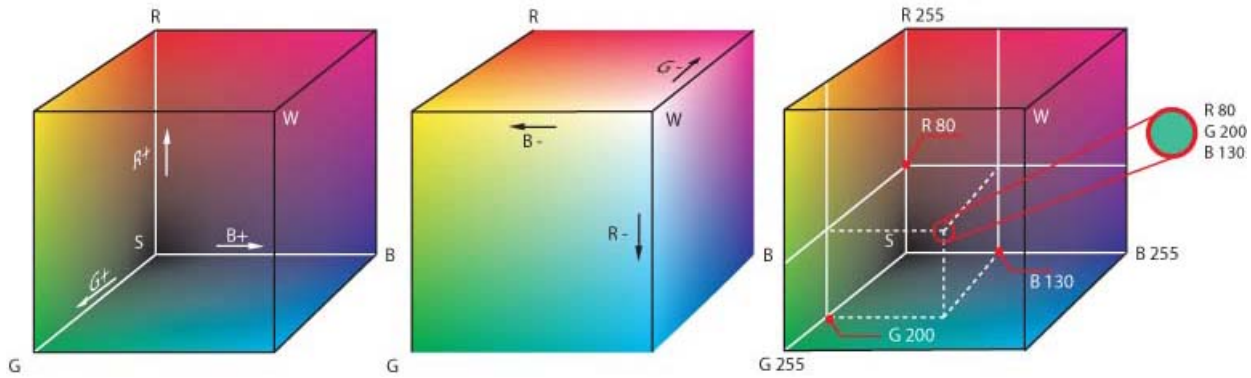


Abbildung 2.2: Der RGB-Würfel mit den Grundachsen Rot, Grün und Blau. Eine Farbe in diesem Raum entspricht einem Punkt im dreidimensionalen kartesischen Raum. In diesem ist auch der euklidische Abstand definiert.

Vom CIE (Commission Internationale de l'Eclairage) gibt es das berühmte CIE-Diagramm, auf dessen Randkurve sich sämtliche Spektrallichter und in dessen Inneren sich sämtliche möglichen Mischungen dieser Spektrallichter befinden, wie in Abbildung 2.1 dargestellt.

Ein Farbmodell könnte nun also z.B. so aussehen, dass man 100 Punkte auf der Randkurve dieses CIE-Diagramms auswählt und eine Farbe als diskretisierte Mischung dieser 100 Punkte darstellt. Ein solches Farbmodell würde einen Großteil des CIE-Farbmodells abdecken. In der Praxis müsste dann allerdings ein Punkt auf dem Bildschirm aus 100 Einzellämpchen zusammengesetzt werden, und in einem Tintenstrahldrucker müssten ebenso viele Farbpatronen eingesetzt werden.

Die nachfolgend beschriebenen Farbmodelle verzichten auf eine Gesamtabdeckung des möglichen Farbraumes, sind dafür jedoch viel einfacher zu beschreiben und in die Praxis umzusetzen.

Das additive RGB-Farbmodell Im RGB-Farbmodell werden sämtliche Farben des RGB-Farbraumes aus den drei Grundfarben Rot, Grün und Blau (RGB) additiv zusammengesetzt. Man verwendet also nur 3 Basisfarben, um durch deren Mischung alle weiteren Farben zu erzeugen. Mischt man alle drei Grundfarben, so erhält man weiß, wird kein Anteil einer Farbe addiert bleibt schwarz übrig. Bei einer Farbtiefe von z.B. 24 Bit, d.h. 8 Bit pro Farbkanal. lassen sich damit 256^3 also ca. 16 Millionen Farben darstellen.

Auf diese Weise wird, wie durch den RGB-Würfel in Abbildung 2.2 graphisch dargestellt, ein dreidimensionaler kartesischer Raum definiert, mit den Koordinatenachsen Rot, Grün und Blau. Im Ursprung befindet sich folglich schwarz, in der gegenüberliegenden Ecke weiß, und auf der verbindenden Diagonalen alle Grauwerte.

Das RGB-Farbmodell lehnt sich an das menschliche Auge an, wo unterschiedliche Rezeptoren auf der Netzhaut auf diese drei Grundfarben reagieren. Nach dem RGB-Farbmodell arbeiten Geräte wie Fernsehapparate, Monitore, Scanner oder Digitalkameras.

Das subtraktive CMY-Farbmodell Wie das RGB-Modell definiert das CMY-Farbmodell einen dreidimensionalen Kartesischen Raum mit den zu Rot, Grün und Blau komplementären Farben Zyan (*cyan*), Magenta(*magenta*) und Gelb (*yellow*) als Grundfarben. Folglich befindet sich im Ursprung weiß, und die Mischung aller drei Grundfarben ergibt schwarz. Dieses Modell wird vor allem im Druckwesen eingesetzt.

Das HSI-Farbmodell Hier steht H für *hue* (Färbung, Tönung), S für *saturation* (Saturierung) und I für *Intensity* (Intensität, Helligkeit).

Dabei gibt die Sättigung an, wie viel weißes Licht der Tönung beigemischt wird. Bei Sättigung Null ist keine Farbe zu sehen, und man erhält somit je nach Intensität Schwarz, Weiß oder Grautöne. Anstelle eines kartesischen Koordinatensystems verwendet das HSI-Modell eine Darstellung in Zylinderkoordinaten. Die Umrechnung von (R, G, B) nach (H, S, I) erfolgt in zwei Schritten: Zunächst wird das RGB-Koordinatensystem gedreht, so dass eine Achse der Raumdiagonalen des Farbwürfels entspricht. Diese wird mit I_1 bezeichnet, die anderen beiden mit M_1 und M_2:

$$\begin{pmatrix} m_1 \\ m_2 \\ i_1 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.1}$$

Im nächsten Schritt werden die kartesischen (m_1, m_2, i_1)-Koordinaten in Zylinderkoordinaten transformiert:

$$\begin{aligned} H &= \arctan\left(\frac{m_1}{m_2}\right) \\ S &= \sqrt{m_1^2 + m_2^2} \\ I &= \sqrt{3} \cdot i_1 \end{aligned}$$

Die umgekehrte Umrechung erfolgt sinngemäß:

$$\begin{aligned} m_1 &= S \cdot \sin H \\ m_2 &= S \cdot \cos H \\ i_1 &= \frac{1}{\sqrt{3}} \end{aligned}$$

und

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = R^t \begin{pmatrix} m_1 \\ m_2 \\ i_1 \end{pmatrix}$$

wobei R^t der transponierten Rotationsmatrix aus Gleichung 2.1 entspricht. Das HSI-Farbmodell wird in der Farbphotographie und in der Videotechnik eingesetzt und eignet sich besonders zur Farbsegmentierung, da der Farbton als eigene Komponente getrennt von Helligkeit und Saturierung betrachtet werden kann.

Eine umfangreiche Beschreibung der genannten und weiterer Farbräume findet sich in [8].

Abbildung 2.3: Drei verschiedene Kameramodelle, die von den FU-Fighters verwendet werden. Die Sony XC 555 (links) ist eine NTSC Kamera, die über ein S-Video Kabel an einen Framegrabber angeschlossen wird. Die anderen beiden sind progressive-scan 1394 Kameras und liefern Bilder im RAW Format. Die AVT Marlin F046c (unten) wird aktuell verwendet und erreicht eine Bildwiederholrate von 53Hz bei einer Auflösung von 780x582.

2.2 Kameras

Das in dieser Arbeit entwickelte System arbeitet mit einer oder zwei Kameras, die auf unterschiedliche Arten an den Rechner angeschlossen werden können. Im Laufe der Zeit wurden verschiedenste Kameramodelle mit unterschiedlichen Eigenschaften eingesetzt. Abbildung 2.3 zeigt drei von uns verwendete Modelle. Bei der Wahl der für die jeweilige Anwendung geeigneten Kamera sind einige Faktoren zu beachten, von denen die zu entwickelnde Software abhängig ist bzw. beeinflusst wird, darunter sind:

- Videoausgang: digital oder analog
- Farbkodierung: RGB, YUV, RAW Format
- *interlaced* oder *progressive-scan* Bildaufnahme
- Bildwiederholrate
- Videoformat (PAL oder NTSC), Auflösung des Bildes

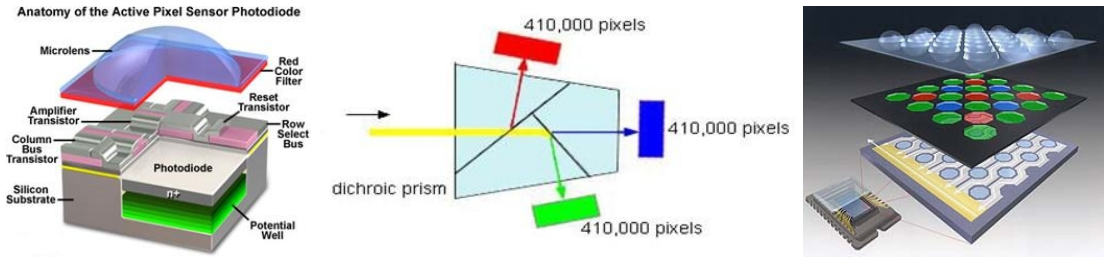


Abbildung 2.4: Links ist eine lichtempfindliche Zelle des CCD-Sensors zu sehen. Durch einfallende Photonen wird Ladung angesammelt. Bei 3 CCD Kameras zerlegt ein Prisma das Licht in seine Grundfarben (Mitte). Bei nur einem CCD wird ein Filterarray verwendet, z.B. im Bayer Muster angeordnet (rechts).

- Schnittstelle zum Rechner: Framegrabber, Firewire
- Gewinde zur Befestigung des Objektivs: C-Mount, CS-Mount
- Größe und Preis

In diesem Abschnitt werde ich auf unterschiedliche Kameraeigenschaften eingehen und welche Probleme sie gegebenenfalls für die Bildauswertung bedeuten können.

2.2.1 1 CCD und 3 CCD Kameras

Moderne Kameras verwenden zur Bildaufnahme CCD Chips (*Charge-coupled Device*), die mit einem Gitter aus lichtempfindlichen Zellen bestückt sind, anhand derer die Helligkeit des einfallenden Lichts gemessen wird.

Während der Belichtung erhöhen die einfallenden Photonen die Ladung an den einzelnen Zellen. Diese Ladungen (engl. 'charge') werden nach der Belichtung ähnlich einer Eimerkette schrittweise verschoben (daher der Wortbestandteil 'coupled'), bis diese einen Ausleseverstärker erreichen. Im Gegensatz zu *progressive-scan* CCD's geben *interlaced* CCD's nur Halbbilder aus, d. h. erst alle ungeraden, dann alle geraden Zeilen.

Dabei können verschiedene negative Effekte auftreten: Wird z.B. eine maximale Ladungsmenge überschritten, gibt die Zelle die überschüssige Ladung an die Nachbarzellen ab. Da diese ebenso nur eine begrenzte Anzahl von Ladungen aufnehmen können, kann sich der Effekt, genannt *Blooming*, abhängig von der Beleuchtungsstärke deutlich ausweiten. Im Bild zeigt sich dies als teilweise weiträumige Überblendungen um besonders helle Bildstellen. Als Maßnahme gegen den Blooming-Effekt können zwischen den Zellen Anti-Blooming-Gates (ABG's) angebracht werden, die den Ladungstransfer verhindern. Da diese allerdings die Größe der Zellen und damit ihre Lichtempfindlichkeit verringern, ist diese Lösung oft nicht praktikabel.

Um den Anteil einer bestimmten Grundfarbe zu ermitteln, werden die Sensorchips mit einem entsprechenden Farbfilter versehen, wie in Abbildung 2.4 links für einen roten Filter dargestellt. 3-CCD Kameras verwenden dazu ein Prisma, welches das einfallende Licht in seine drei Grundfarben zerlegt. Das vollständige Farbbild kann nun Pixel für Pixel aus den drei Bildern für Rot, Grün und Blau kombiniert werden.

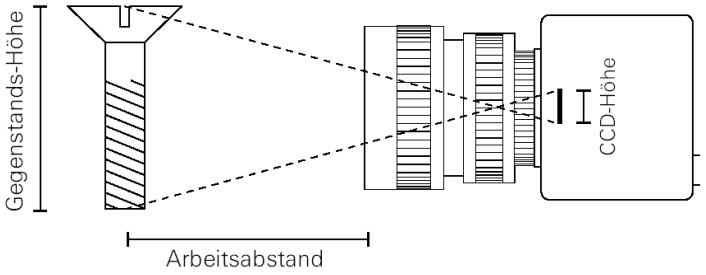


Abbildung 2.5: Graphische Darstellung zur Berechnung der notwendigen Brennweite (Abstand der Linse zum Brennpunkt). Diese hängt ausschließlich von der Größe des CCD und des zu betrachtenden Objekts, wie auch von dem gewünschten Abstand zu diesem ab [2].

Konventionelle Kameras verwenden allerdings nur einen CCD Chip, um ein Farbbild zu erfassen. Dabei werden ähnlich der Retina im menschlichen Auge die einzelnen Zellen des einen Sensorchips mit unterschiedlichen Farbfiltern versehen. Eine weit verbreitete Verteilung ist das Bayer-Muster, in dem es doppelt so viele grüne wie rote oder blaue Pixel gibt. Die fehlenden Farbanteile müssen an jedem Pixel aus der Nachbarschaft extrapoliert werden, um ein vollständiges Farbbild zu erhalten. Diese Aufgabe, genannt *Demosaicing*, wird je nach unterstütztem Übertragungsformat von der Kamera übernommen oder der Bildverarbeitung auf dem Rechner überlassen.

2.2.2 Objektive und Linsen

Ein Objektiv ist ein sammelndes optisches System, das eine reelle optische Abbildung eines Objektes erzeugt. Es können sowohl Linsen als auch spiegelnde Flächen Bestandteile eines Objektivs sein. Man unterscheidet

- Teleobjektiv,
- Normalobjektiv,
- Weitwinkelobjektiv und
- Fischaugenobjektiv.

Normalobjektive erzeugen Bilder, die einen natürlich perspektivischen Eindruck vermitteln. Teleobjektive besitzen eine im Gegensatz zu Normalobjektiven längere Brennweite und damit einen kleineren Blickwinkel. Für unsere Anwendung, bei der inzwischen eine Feldfläche von knapp $22m^2$ bei einer Kamerahöhe von teilweise weniger als drei Metern erfasst werden muss, kommen allerdings nur Weitwinkelobjektive in Frage. Diese haben gegenüber den Normalobjektiven eine kürzere Brennweite. Die oben genannten Kategorien gelten für Festbrennweiten-Objektive. Zoomobjektive mit variabler Brennweite können je nach Brennweiten-Bereich auch mehrere der genannten Kategorien abdecken.

[2] Bildquelle: http://www.1394imaging.com

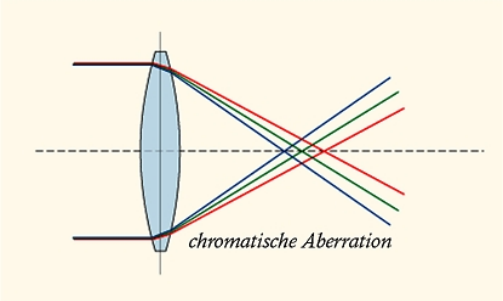


Abbildung 2.6: Chromatische Aberration einer Linse. Man sieht wie Licht der drei Grundfarben Rot, Grün und Blau in unterschiedlichen Punkten entlang der optischen Achse fokussiert werden. Zudem können sie auch innerhalb der Bildebene voneinander abweichen.

Die für eine spezielle Anwendung benötigte Brennweite berechnet sich wie folgt als Minimum der Brennweite in der Breite und in der Höhe, der entsprechende Aufbau ist in Abbildung 2.5 dargestellt:

$$\text{Brennweite der Breite} = \frac{\text{Arbeitsabstand} \cdot \text{CCD Breite}}{\text{Gegenstandsbreite} + \text{CCD Breite}} \tag{2.2}$$

$$\text{Brennweite der Höhe} = \frac{\text{Arbeitsabstand} \cdot \text{CCD Höhe}}{\text{Gegenstandshöhe} + \text{CCD Höhe}} \tag{2.3}$$

Die FU-Fighters verwenden momentan 6mm Weitwinkelobjektive bei einer Kamerahöhe von vier Metern während offizieller Turniere und 4.2mm Objektive im Labor bei einer Deckenhöhe von unter drei Metern.

Derartige Weitwinkelobjektive erzeugen vor allem im Randbereich eine starke radiale Verzerrung. Zudem können optische Systeme Abbildungsfehler wie z.B. durch chromatische Aberration verursachen.

Chromatische Aberration

Die chromatische Aberration (von griech. chroma, die Farbe und lat. ab-errare, wegirren) ist ein Abbildungsfehler optischer Linsen, der von der Wellenlänge bzw. Farbe des Lichts abhängt.

Linsen brechen das Licht in Abhängigkeit von seiner Wellenlänge. Aus diesem Grund wird einfallendes Licht unterschiedlicher Wellenlänge in einer Sammellinse in verschiedenen Punkten fokussiert, und es treten Farbsäume auf. Dabei weichen die drei Grundfarben sowohl entlang der optischen Achse, wie in Abbildung 2.6 zu sehen, als auch innerhalb der Bildebene voneinander ab. Der Effekt tritt besonders an harten Kanten, vor allem an weißen Linien auf, in denen das gesamte Farbspektrum enthalten ist.

In hochwertigen – und entsprechend teuren – optischen Systemen wird dieser Fehler durch Kombination von Linsen aus Gläsern verschiedener Dispersion korrigiert. Werden dabei die am stärksten voneinander abweichenden Grundfarben Rot und Blau zusammengeführt, spricht man von einer achromatischen Korrektur. Wird zudem die Grund-

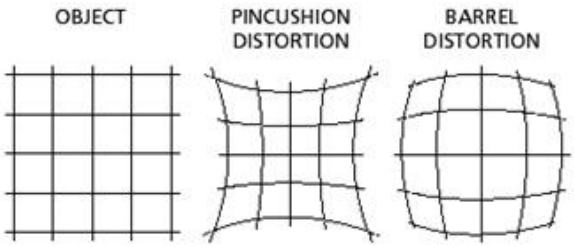


Abbildung 2.7: Beispiel einer kissen- und tonnenförmigen radialen Verzerrung für ein uniformes Gitter.

farbe Grün mit den beiden anderen zusammengelegt, handelt es sich um eine apochromatische Korrektur.

Radiale Verzerrung
Radiale Verzerrung wird vom optischen System der Kamera verursacht. Sie ist in allen Richtungen symmetrisch und nimmt zum Rand hin zu, während im Zentrum der Linse keine Verzerrung vorhanden ist. Es treten kissenförmige und tonnenförmige radiale Verzerrungen auf oder sogar eine Kombination dieser beiden, wie in Abbildung 2.7 dargestellt. Diese sind nicht-linear, so dass Linien als Kurven im Bild erscheinen. Das Vision System muss diese Verzerrung geeignet modellieren, um eine korrekte Abbildungsfunktion von Pixelkoordinaten in Feldkoordinaten zu finden.

3 Problembeschreibung und Bildvorverarbeitung

3.1 Problembeschreibung – Warum ist es schwierig ?

Auch wenn es einem Menschen nicht schwer fallen mag, einen Ball oder Roboter auf dem Spielfeld zu verfolgen, sieht sich doch eine Maschine mit einer Vielzahl von Problemen konfrontiert. Zunächst gilt es 11 sehr schnell bewegende Objekte gleichzeitig zu verfolgen. Aktuelle Roboter in der Small-Size Liga fahren mit bis zu 3m/s über das Feld und schießen den Ball mit bis zu 15m/s.

Zur Auswertung benutzen wir momentan zwei Kameras mit einer Auflösung von 780 × 582 Pixel bei 53Hz. Das entspricht der immensen Datenrate von 140 MB, die pro Sekunde verarbeitet werden muss. Dabei darf der Rechner niemals voll ausgelastet werden, um anderen Programmen, wie der Verhaltenssteuerung, noch genügend Rechenzeit zu geben.

Inhomogene Lichtverhältnisse stellen eine große Schwierigkeit dar: Es gibt sowohl Schatten von Robotern und umstehenden Zuschauern, als auch starke Reflektionen durch die Beleuchtung, so dass die Objekte kaum noch ein konstantes Erscheinungsbild aufweisen. Dabei variiert nicht nur die Helligkeit – wie man zunächst annehmen könnte, sondern auch die Saturierung und der *hue* einer Farbe, d.h. die „Knalligkeit“ und der Farbwert selbst. So kann ein gelber Teammarker eines Roboters, bewegt man ihn über das Feld, Farben von einem blassen, hellen Gelb bis zu einem dreckigen Braunton annehmen, oder ein Pink auch mal wie Rot oder Orange erscheinen. Auch enthält das Bild an vielen Stellen Fehlfarben, die in Wirklichkeit gar nicht vorhanden sind. Dafür gibt es mehrere Ursachen:

Chromatische Aberration Wie unter 2.2.2 beschrieben, brechen Linsen den Rot-, Grün- und Blauanteil des eingehenden Lichts unterschiedlich, so dass der Fokalpunkt sowohl im Abstand zur Linse, als auch in der Bildebene abweichen kann. Dies bewirkt zum Beispiel, dass weiße Linien, aus der Nähe betrachtet, im Bild aus vielen verschiedenen Farben bestehen, die vor allem in radialer Richtung variieren.

Demosaicing 1 CCD Kameras nehmen in jedem Pixel immer nur einen Anteil (Rot, Grün oder Blau) der Farbe wahr, so dass das vollständige Farbtripel aus seinen Nachbarn interpoliert werden muss. Auf uniformen Flächen resultiert so die korrekte Farbe, aber an harten Kanten werden Farbanteile von verschiedenen Farben interpoliert, und es entstehen Fehlfarben im Bild. Besonders tritt dieser Effekt wiederum an den weißen Feldlinien auf, aber auch an den Rändern von Farbmarkierungen.

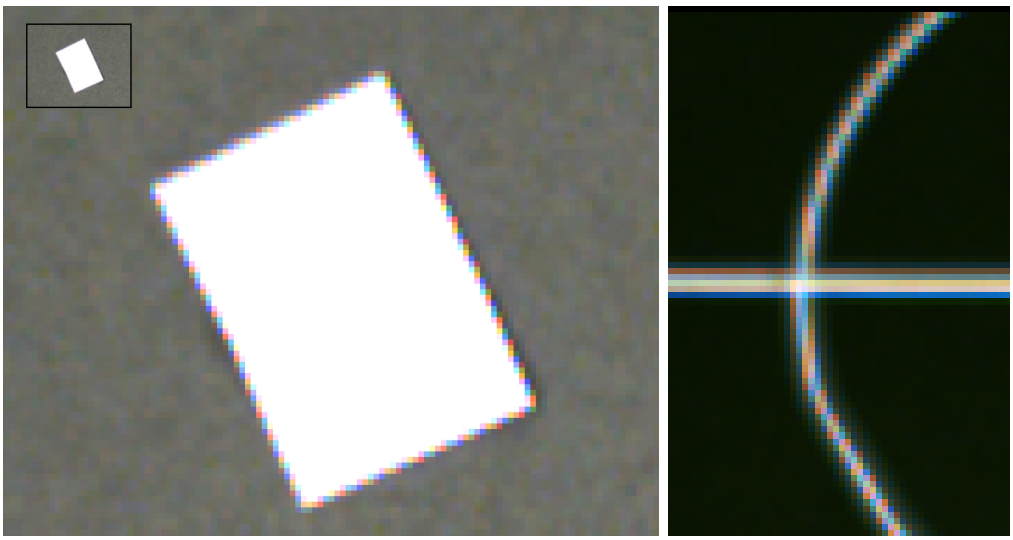


Abbildung 3.1: Durch chromatische Aberration und Demosaicing des Bayer-Filters entstehen Fehlfarben an weißen Linien.

Blooming Durch das einfallende Licht sammelt sich in den einzelnen Zellen des CCD Chips Ladung an. Wenn diese einen Wert überschreitet, kann sie auf benachbarte Zellen „überschwappen“. Im Bild bewirkt dieser Effekt, dass Farben auf ihre Nachbarschaft abstrahlen und auf diese Weise benachbarte Farbmarkierungen ineinander verlaufen können. So entsteht zum Beispiel zwischen einem gelben und einem pinken Marker oft ein orangener Übergang. Der gleiche Effekt bewirkt, dass in sehr hellen Spots ein weiter Bereich von Weiß vollkommen überstrahlt wird und somit die Farbinformation völlig verloren geht.

Neben diesen Bildfehlern können Kameras noch andere Probleme verursachen:

Interlacing Kameras ohne *progressive-scan* lesen zunächst alle ungeraden Zeilen des Bildes und darauf die geraden Zeilen. Objekte können sich in der Zwischenzeit bewegt haben, so dass ein *interlacing* Effekt auftritt, der die bewegten Objekt kammartig auseinander zieht.

Verschmierungen Während der Shutter der Kamera geöffnet ist, integriert der Chip das einfallende Licht. Die Shutterzeit kann nicht beliebig klein gehalten werden, da sonst zu wenig Licht eintritt, so dass sehr schnelle Objekte, abhängig von ihrer Geschwindigkeit, leicht bis stark verschmieren können. Dies führt bei unseren Robotern nahe ihrer Maximalgeschwindigkeit zum Verschmelzen von zwei bis drei der Farbmarkierungen. Der Ball hingegen, der mit einer wesentlich höheren Geschwindigkeit geschossen wird, erscheint als ein transparenter Schweif im Bild, da er sich nur einen Bruchteil der Shutterzeit an einem Fleck befindet und den Rest der Zeit der Hintergrund, also das grüne Feld, zu sehen ist.

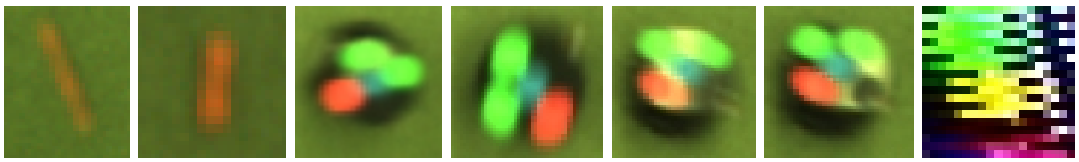


Abbildung 3.2: Verschiedene Artefakte sind zu sehen wie Verschmieren bei harten Schüssen und schnellen Robotern, Spiegelung des Lichts auf halbwegs glänzenden Oberflächen und *Interlacing*.

Abbildung 3.3: Beide Bilder zeigen die Entstehung falscher Farben. Rechts sieht man wie der blaue Marker die anderen überstrahlt, so dass an Übergängen Farbmischungen erzeugt werden. Der gelbe Teammarker hat aufgrund des *Blooming* – Effekts an Sättigung eingebüßt.

Verzerrung Oft steht nur eine Deckenhöhe von knapp 3 Metern zur Verfügung, um eine Spielfeldfläche von 5×4 Metern abzudecken. Darum ist man auf stark verzerrende Objektive mit weitem Öffnungswinkel angewiesen. Linien erscheinen als Kurven im Bild, Kreise als Ellipsen, und Farbmarkierungen können im Randbereich auf wenige Pixel schrumpfen. Zudem werden die Bereiche, in denen der Ball von Robotern verdeckt wird, stark vergrößert.

Es müssen folglich nicht immer alle Objekte sichtbar sein. Roboter können den Ball seitlich verdecken, aber auch ihrerseits verdeckt werden, z.B. wenn der Schiedsrichter ins Spiel eingreifen muss. Die Farbe, Form und Anzahl der Markierungen variieren ausserdem von Team zu Team, das Vision-System muss sich daher schnell an das unterschiedliche Aussehen der verschiedenen Gegner anpassen können.

Sind nun Ball und Roboter im Bild gefunden, müssen ihre Position und Orientierung von Pixelkoordinaten im Bild in Weltkoordinaten transformiert werden, damit sie z.B. von der Steuerungssoftware verwendet werden können.

Schließlich stelle ich mir die Anforderung, ein möglichst variables und unabhängiges System zu entwerfen. Es soll nicht auf spezialisierter Hardware wie FPGAs oder DSPs aufsetzen und auch mit Standard-Kameras auf einem Standard-PC mit durchschnittlich geringer Rechenlast laufen. Die Regeln im RoboCup werden Jahr für Jahr geändert und dabei mehr an die „reale“ Welt angepasst. So ist z.B. die Größe des Feldes von anfänglich

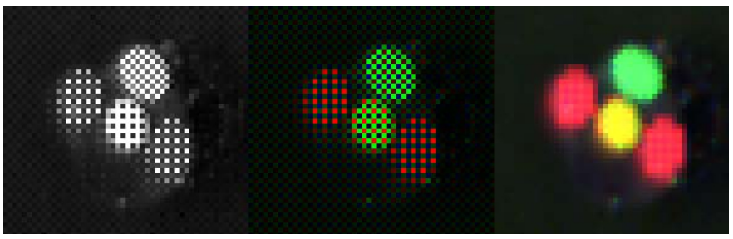


Abbildung 3.4: Links die Rohdaten, wie sie von der Kamera übertragen werden, in der Mitte entsprechend dem Bayer Muster eingefärbt und rechts mit linearer Interpolation für alle Bildpunkte.

$1.52m \times 2.74m$ inzwischen auf $5m \times 4m$ angewachsen. Ein variables Vision-System muss sich daher möglichst leicht an künftige Änderungen anpassen lassen können.

3.2 Bilderfassung und -vorverarbeitung

Unser System bietet mehrere Möglichkeiten, Kameras an den Rechner anzuschließen, zudem können zu Testzwecken auch Bilder oder Videos als Quelle dienen.

Anschluss über Framegrabber Hierbei werden die Kameras analog über Framegrabberkarten mit dem Rechner verbunden. Der Einbau mehrer Karten kann teilweise Probleme bereiten, da nicht jeder Treiber für eine mehrfache Verwendung ausgelegt ist. Um höhere Frameraten als 30 Hz zu erzielen, sind spezielle und teurere Framegrabberkarten notwendig.

FireWire (IEEE-1394) Kameras Eine weitere Möglichkeit ist, Kameras über Firewire an den Rechner anzuschließen. Vorteil dabei ist, dass beinahe jeder moderne Rechner mit Firewire ausgestattet ist. Um hohe Frameraten bei gleichzeitig hoher Auflösung erzielen zu können, müssen die Kameras an getrennten Controllern angeschlossen werden, da ein gemeinsam verwendeter Bus durch seine Bandbreite die Framerate zu sehr begrenzt. Ebenso müssen die Bilder im RAW Format übertragen werden, d.h. entsprechend dem Bayer-Muster gibt es pro Pixel nur einen Farbanteil. Auf diese weise senkt sich die Datenrate im Verhältnis zu vollen RGB-Bildern um ein Drittel.

Letztere Möglichkeit wirft zwei Probleme auf: Bilder im RAW Format müssen zunächst in RGB Bilder konvertiert werden. Außerdem bieten Firewire Kameras – gerade im Wissenschaftsbereich – nicht immer die Möglichkeit eines Weißabgleichs.

3.2.1 Demosaicing

Einfach CCD Kameras nehmen Farbbilder mittels eines vorgeschaltenen Farbfilter-Arrays auf (color filter array, CFA) (siehe 2.2). Die Filter sind entsprechend dem Bayer Mosaik angeordnet, so dass an jedem Pixel im Bild nur ein Farbanteil vorhanden ist. Demosaicing nennt man das Problem aus einem solchen Bild, die eigentlichen RGB

Werte an jedem Pixel zurück zu rechnen. Ein einfacher Ansatz ist, die fehlende Farbinformation bilinear aus den direkten Nachbarn zu interpolieren [13],[14],[11]. Wie unter 3.1 gesehen, kann es dabei aber gerade an harten Kanten zu Fehlfarben kommen. Daher gibt es eine Vielzahl von Algorithmen die solche Fehlinterpolationen zu verhindern suchen.

Die grundlegende Idee dabei ist Korrelation zwischen den einzelnen RGB-Kanälen zu verwerten. Der *constant-hue* [14] Ansatz von Freeman geht davon aus, dass sich der Farbwert von Pixel zu Pixel nicht stark ändert (siehe HSI Farbraum in 2.1). Dabei wird zunächst der Grün-Kanal bilinear interpoliert und bei den anderen Kanälen die Interpolation so gewichtet, dass der Hue, hier als R/G bzw. B/G Ratio definiert, möglichst konstant bleibt.

Gradienten-basierte Verfahren schätzen die Richtung von Kanten, und versuchen, nicht über Kanten hinweg zu interpolieren [14]. Noch bessere Qualität wird durch iterative Methoden wie den Filter von Kimmel erzielt [11].

Der große Nachteil dieser Verfahren ist ihre hohe Komplexität: Die teuersten berechnen an die 480 Operationen pro Pixel und greifen weit über die direkte Nachbarschaft hinaus. Bei einer Framerate von 50 Hz stehen dem gesamten System maximal 18 ms pro Frame zur Verfügung. Zudem ist es für die Steuerungssoftware wichtig, die Latenzzeit des Gesamtsystems möglichst niedrig zu halten.

Daher ist eine billigere Methode, die leicht zu optimieren ist, trotz der entstehenden Artefakte zu favorisieren.

In diesem Sinne habe ich eine noch einfachere Methode implementiert: Dabei werden jeweils vier benachbarte Pixel zu einem zusammengefasst, wobei Rot und Blau direkt übernommen und die beiden Grünen diagonal interpoliert werden.

3.2.2 Software-Weißabgleich

Es wurde ein Algorithmus zum Weißabgleich implementiert, der von der *gray-world-assumption* [3] ausgeht. Dabei wird in einem iterativen Prozess jeweils ein Skalierungsfaktor für Rot- und Blauanteil bestimmt, so dass die Farben über das gesamte Bild gemittelt annähernd Grau ergeben.

4 Detektion und Tracking von Farbmarken

4.1 Tracking - Verfolgung von bewegten Objekten

Wie unter 3.1 geschildert haben wir es mit sehr schnell bewegenden Objekten zu tun. Daher ist für einen Roboter, der in einer so interaktiven Umgebung agieren soll – sei es um seine eigenen Bewegungen zu koordinieren, auf andere Roboter zu reagieren oder einen Pass abzufangen – eine schnelle Wahrnehmung essentiell. Es wird sofort klar, dass ein Vision-System mit gleichzeitig hoher Auflösung und Framerate sich unmöglich jedes Mal mit aufwendigen Methoden das ganze Bild anschauen kann.

Aber betrachten wir uns zunächst eine ähnliche Situation beim Menschen: Ein Zuschauer eines Tennisspiels schaut sich nicht jedes Mal den gesamten Platz an, um den Ball zu finden, er versucht einfach, ihn im Auge zu behalten. Dabei fällt es ihm leichter einen langsamen Ball zu verfolgen, während er ihn bei sehr harten Schlägen teilweise bis ganz aus dem Auge verliert und eben doch größere Bereiche durchsuchen muss.

Nach diesem Prinzip habe ich einen Algorithmus entwickelt, der die Positionen von Objekten wie zum Beispiel Ball oder Farbmarkierungen von Robotern vorhersagt und nur einen kleinen Bereich herum betrachtet. Die Größe dieses Suchrahmens passt sich dabei dynamisch an die gegebene Situation an.

4.1.1 Positionsvorhersage

Die Positionsvorhersage hat die Aufgabe die Positionen der zu verfolgenden Objekte im aktuellen Frame möglichst genau zu schätzen, so dass nur möglichst kleine Bereiche um diese Schätzungen durchsucht werden müssen, um das jeweilige Objekt zu finden.

Ich berechne die vorhergesagte Position linear anhand der beiden vorherigen Positionen des Objekts. Dabei wird die Reibung der Oberfläche durch einen abschwächenden Faktor modelliert. In früheren Versionen, wo das Spielfeld noch mit Wänden umrandet war, wurden ebenfalls Reflexionen des Balls an den Wänden in der Vorhersage berücksichtigt.

Gerade für die farbigen Markierungen der Roboter sind wesentlich genauere und komplexere Methoden denkbar. So könnte man mittels eines physikalischen Modells des Roboters seine zukünftige Position und Orientierung vorhersagen, und anhand eines Robotermodells über die Anordnung der Farbmarkierungen auf deren vorhergesagte Positionen schließen. Der Vorteil dieser Methode wird bei einem rotierenden Roboter leicht ersichtlich: Während mithilfe eines physikalischen Modells die Markierungen wirklich auf einer Kreisbahn vorhergesagt werden, wird diese im simplen Verfahren nur stückweise linear approximiert. Allerdings muss ein solches physikalisches Modell eines Roboters analytisch bestimmt oder wie zum Beispiel unter [1] gelernt werden. Zudem

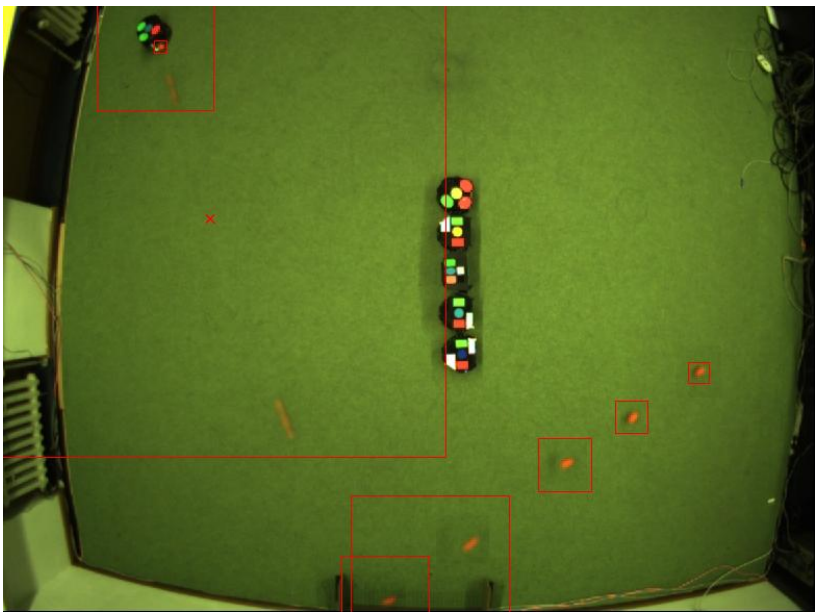


Abbildung 4.1: Wir sehen eine Schusssequenz eines hart geschossenen Balls (es wurden dabei nicht alle Frames in das Bild eingezeichnet). Zur Veranschauung sind die Suchrahmen eingezeichnet, die in einer derartigen Bildfolge zu erwarten sind. Zu Beginn des Schuss ist die vorhergesagte Position (rotes Kreuz), auf der der Suchrahmen zentriert wird, weit entfernt von der wirklichen Position und der Rahmen muss stark anwachsen. Später kann der Ball wieder präziser vorhergesagt werden und der Rahmen schrumpft wieder auf seine Minimalgröße.

sind die Änderungen zwischen zwei Frames bei einer hohen Framerate meistens nicht sehr groß, so dass die lineare Vorhersage gute Ergebnisse liefert.

Nun stellt sich die Frage, wie groß der Suchrahmen sein soll, in dem man den Ball oder eine farbige Markierung der Roboter um die vorhergesagte Position herum suchen soll.

4.1.2 Variable Suchrahmen

Wie beim Tenniszuschauer sind auch hier die vorhergesagten Positionen bei sich sehr schnell bewegenden Objekten nicht sehr genau, so dass größere Bereiche betrachtet werden müssen. Bei sehr langsamen oder stehenden Objekten hingegen werden die Vorhersagen sehr genau und ein sehr kleiner Suchrahmen genügt.

Daher habe ich einen Algorithmus entwickelt, der die Größe des Suchrahmens dynamisch anpasst. Finden wir das Objekt in dem gewählten Suchrahmen, dann suchen wir es im nächsten Kamerabild mit einem etwas kleineren Rahmen. Ist die Suche allerdings erfolglos, so vergrößern wir den Rahmen auf das Vierfache und suchen erneut, bis eine maximale Größe erreicht ist (z.B. wenn das gesamte Bild oder das Spielfeld abgedeckt ist). Neben dieser maximalen Größe wird der Rahmen auch durch eine minimale Größe

beschränkt, in die das Objekt gerade noch hineinpasst. Die Teile des Suchrahmens, die außerhalb des Feldes liegen, werden nicht durchsucht.

Die durchschnittliche Größe des Suchrahmens während eines Spiels liegt nahe seinem Minimalwert. Nur bei schnellen Objekten ist ein größerer Rahmen notwendig, wie bei einem geschossenen Ball oder wenn ein Objekt nicht richtig gefunden wird, z.B. aufgrund schlechter Lichtverhältnisse oder Verdeckungen.

Wenn zum Beispiel der Ball verdeckt wird oder verdeckt über das Feld gedribbelt wird und an ganz anderer Stelle wieder auftaucht, muss sein Suchrahmen mehrere Male vergrößert werden, bis die Suche erfolgreich oder seine Maximalgrenze erreicht ist. Dabei werden Teile des Bildes mehrfach durchsucht, da die schon durchsuchten kleineren Rahmen in den größeren vollkommen enthalten sind. Um dieses mehrfache Durchsuchen zu vermeiden, kann der neue vergrößerte Suchrahmen in fünf Bereiche zerlegt werden, die separat durchsucht werden, wobei der alte Suchrahmen ausgespart wird. Dies ist allerdings für die grundsätzliche Funktionsweise des Verfahrens nicht von Belang, sondern dient ausschließlich der Beschleunigung.

Algorithm 4.1 Tracking mit variablem Suchrahmen(SR)

```
1: p ← BerechneVorhersageDesObjekts()
2: repeat
3:    Zentriere SR auf p
4:    Suche Objekt im SR
5:    if gefunden then
6:       SR.size ← max(SR.size − δ_Decrease, SR_Min)
7:    else
8:       SR.size ← min(SR.size ∗ δ_Increase, SR_Max)
9:    end if
10: until (SR maximal ∨ Objekt gefunden)
11: return gefunden
```

4.2 Modell von Ball und Robotern

Um ein Objekt im Bild finden bzw. verfolgen zu können, muss man sich zunächst für eine geeignete Repräsentation im Rechner entscheiden. Anhand dieses Modells wird während der Suche bestimmt, ob und wie gut ein Objekt im Bild dem gesuchten Objekt gleicht. Die entsprechenden Kriterien sollten daher so gewählt werden, dass diese Entscheidung möglichst gut getroffen werden kann. Ein solches Modell kann anfänglich entweder manuell (z.B. durch anklicken mit der Maus durch den Benutzer) oder durch automatische Kalibrierung initialisiert werden. Bei erfolgreicher Suche wird das Modell weiter angepasst, so dass es möglichst gut dem aktuellen Zustand entspricht.

Ballmodell Der Ball wie auch die Markierungen der Roboter erscheinen als bewegte farbige Punkte im Bild. Allerdings variiert die Farbe aufgrund von Beleuchtung und Reflektionen vor allem sehr stark in der Helligkeit. Teams verwenden runde, elliptische,

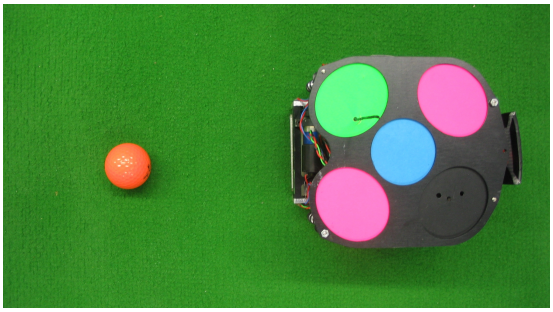


Abbildung 4.2: Großaufnahme von Ball und Roboter. Ball und Marker erscheinen als farbige kompakte Flächen im Bild. Der Roboter besitzt neben dem Teammarker noch zusätzliche Marker in kreisförmiger Anordnung.

wie auch rechteckige Markierungen für ihre Roboter. Zudem wird die Form durch Effekte wie das *Interlacing* oder die teilweise sehr starke Verzerrung der Kamera beeinflusst. Die Form scheint daher kein geeignetes Kriterium zu sein. Allerdings weisen die Marker meist eine kompakte Form auf, anhand derer sie sich zum Beispiel von den länglichen Farbverfälschungen an den weißen Linien unterscheiden lassen. Daher modelliere ich Ball bzw. die Farbmarken durch folgende Kriterien:

- Position und Geschwindigkeit
- Farbe im HSI-Raum
- Größe in Pixeln
- Kompaktheit

Robotermodell Neben dem gelben oder blauen Teammarker werden noch zusätzliche Markierungen für die Orientierung und Identifizierung der Roboter verwendet. Ihre Anzahl und geometrische Anordnung variieren dabei von Team zu Team. Daher habe ich mehrere Modelle entworfen, die dieses Spektrum möglichst gut abdecken sollen. Welches Modell für welchen Gegner das beste ist, wird vom Benutzer vor dem Spiel manuell bestimmt, könnte aber zukünftig auch automatisch entschieden werden. Auf die einzelnen Modelle werde ich auf Seite 48 genauer eingehen, allgemein bestehen sie aus folgenden Kriterien:

- Position und Geschwindigkeit
- Höhe des Roboters
- Menge von Markierungen mit Modell
- Modell der geometrischen Anordnung
- `[optional]` Orientierungs- und Identifizierungsmodell

4.3 Verfolgen von Farbmarken

Um den Ball oder eine Farbmarke zu verfolgen, braucht man nun eine Methode, sie anhand ihres Modells in ihrem jeweiligen Suchrahmen zu finden. Dabei reicht es nicht einfach nur, die am besten passende Stelle im Suchrahmen zu bestimmen, sondern es muss hart entschieden werden, ob das Objekt wirklich an der gegebenen Stelle vorhanden ist oder nicht, so dass im negativen Fall der Rahmen vergrößert werden kann.

Auch hier gilt es hohe Priorität auf die Rechenzeit zu legen. Gerade beim Ball, der häufig verdeckt ist, kann der Suchrahmen sehr groß werden. Es ist also verboten mit aufwendigen Methoden den gesamten Suchrahmen zu untersuchen.

Aus diesem Grund habe ich meinen Algorithmus in zwei Stufen unterteilt. Im ersten Schritt wird der gesamte Rahmen mit einer weniger aufwendigen Methode betrachtet und eine oder mehrere in Frage kommende Stellen ausfindig gemacht, an denen ich im zweiten Schritt mit einer aufwendigeren Methode versuche die Existenz des gesuchten Objekts zu verifizieren.

4.3.1 Schritt 1 – Schnelles finden von Hypothesen

Pixel mit ähnlicher Farbe zum Modell sind ein guter Hinweis auf die Existenz der gesuchten Farbmarkierung. Ein erster Ansatz ist folglich für jeden Pixel im Suchrahmen die RGB-Distanz[1] zum Modell zu berechnen und den ähnlichsten als Hypothese zu verwenden. Die Schwäche dieser Methode wird sofort klar: Ein einzelnes Pixel, welches zum Beispiel durch Farbverfälschungen an einer Linie entstanden ist, kann zu Unrecht das Interesse auf sich lenken und somit das Finden des eigentlichen Objektes verhindern. Einerseits wird dieses Problem dadurch abgeschwächt, dass die Suchrahmen im Allgemeinen klein sind, so dass potentielle Gefahrenzonen gar nicht enthalten sind, andererseits helfen Mechanismen wie Pixelmaskierung und das Entfernen gefundener Objekte im Bild, auf die ich auf Seite 38 bzw. 55 eingehen werde, solche Fehlhypothesen zu verhindern. Als weitere Möglichkeit können mehrere Hypothesen aufrechterhalten werden, die einen gewissen Mindestabstand aufweisen, so dass sie nicht auf demselben Objekt liegen.

Um das Verfahren noch zusätzlich zu beschleunigen, werden Hypothesen nur untersucht, sofern sie keine zu große RGB-Distanz zum Modell aufweisen.

Außerdem wird bei sehr großen Suchrahmen eine reduzierte Auflösung verwendet. Dazu betrachtet sich die Methode sowohl horizontal als auch vertikal nur jedes zweite Pixel, so dass die Auflösung auf ein viertel reduziert wird. Dabei werden abwechselnd einmal die geraden und einmal die ungeraden Zeilen bzw. Spalten ausgespart, so dass mehrere Aufrufe jedes Mal ein anderes Viertel betrachten.

4.3.2 Schritt 2 – Verifizieren von Hypothesen: Segmentierung

Nachdem in Schritt 1 nur anhand der Farbe im RGB Raum eine Vorauswahl getroffen wurde, wird eine solche Hypothese nun mithilfe aller Kriterien des Modells in einer klei-

[1] Euklidischer Abstand im RGB-Raum. Das teure Wurzelziehen ist hier allerdings unnötig und wird ausgespart

nen Region untersucht. Diese Region kann zum Beispiel ein festes Rechteck (Segmentierrahmen), oder auch eine dynamisch wachsende Region sein. Der Segmentier-Algorithmus entscheidet nun für jedes Pixel der Region, ob es zum gesuchten Marker gehören könnte (*Farbklassifizierung*) und berechnet aus den positiv klassifizierten Pixeln die *Qualität* wie auch sub-pixel-genau seine Position als Schwerpunkt, d.h. als gemittelte Summe.

Mithilfe der Qualität kann nun das aufrufende Modul (zum Beispiel Ball- oder Robotersuche) entscheiden, ob das gesuchte Objekt gefunden wurde und sein Modell entsprechend *anpassen*. Dabei adaptieren die Module zur Robotersuche erst dann das Modell eines ihrer Marker, wenn auch der gesamte Roboter zufrieden stellend gefunden wurde. Im Folgenden werde ich auf die einzelnen Schritte genauer eingehen.

4.3.2.1 Farbklassifizierung

Die Farbklassifizierung hat die Aufgabe für eine gegebene Farbe zu entscheiden, zu welchem Objekt bzw. in unserem Fall ob sie zu dem gesuchten Objekt gehören könnte. Oftmals wird für diese Zwecke eine feste Lookup-Tabelle benutzt, die für jede Farbe die entsprechende Zugehörigkeitsinformation speichert. In einem solchen Fall ist die gesamte Tabelle Teil des Modells und muss meistens aufwendig manuell kalibriert werden. In geeigneten Fällen kann hierfür auch eine automatische Kalibrierung verwendet werden, wie z.B. in [7].

Solche Lookup-Tabellen sind zudem meist sehr groß ($\approx$ 16MB) und schwer in Echtzeit anzupassen. Ich fordere von meinem System aber eine hohe Adaptionsfähigkeit, d.h. sich möglichst schnell an Veränderungen anzupassen. Daher werde ich die Klassifizierung stattdessen anhand einer einfachen Farbdistanz vornehmen.

Wie unter 3.1 beschrieben, kann das Aussehen der Farbmarken in Abhängigkeit der Position auf dem Feld sehr stark variieren. Dabei schwankt der Farbwert weniger als seine Saturierung, und am meisten die Intensität. Im Gegensatz zum simplen RGB-Abstand ist es folglich sinnvoll diese Komponenten getrennt zu betrachten und unterschiedlich stark zu gewichten. Ich klassifiziere die Pixel daher im HSI-Raum nach folgenden Kriterien:

Die Intensität darf absolut nicht zu stark abweichen,

$$|I - I_{Modell}| < I_{Max} \tag{4.1}$$

die Saturierung darf relativ gesehen nicht zu klein werden,

$$S > S_{relativMin} * S_{Modell} \tag{4.2}$$

und der Farbwert(Hue) muss ausreichende Ähnlichkeit aufweisen

$$\Delta_H(H, H_{Modell}) < H_{Max} \tag{4.3}$$

wobei Δ_H, dem Abstand im Winkel entspricht, d.h.

$$\Delta_H(h_1, h_2) = \min(d, 360^\circ - d)$$

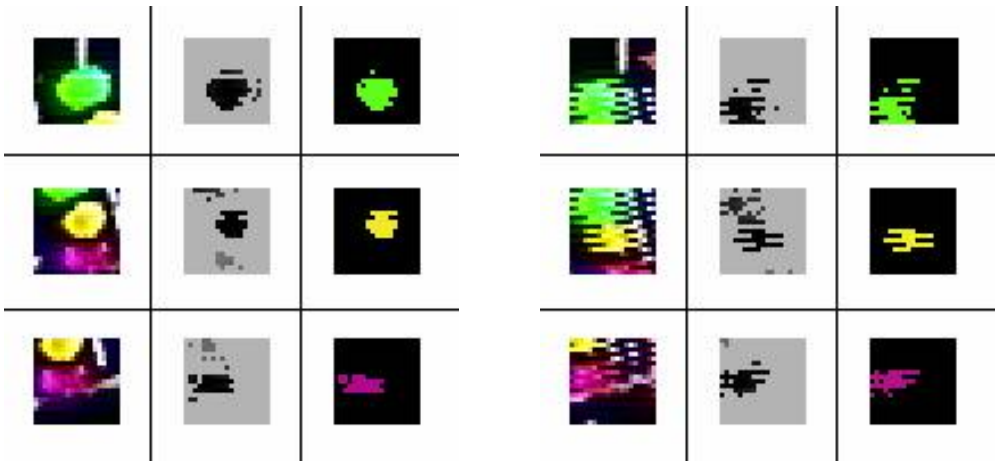


Abbildung 4.3: Das linke Bild zeigt für drei Marker eines Roboters (Spalte 1) die jeweils noch positiv klassifizierten Pixel nach Beschränkung durch Helligkeits- und Saturierungskriterium (Spalte 2), und zusätzlich des Farbwerts (Spalte 3). Im rechten Bild sieht man dasselbe Szenario bei einem bewegten Roboter. Man erkennt, dass es sich bei der Aufnahme um keine *progressive-scan* Kamera handelt, da ein starker *interlacing*-Effekt auftritt.

mit $d = |h_1 - h_2|$.

Figur 4.3 zeigt die jeweils noch positiv klassifizierten Pixel nach Anwenden der einzelnen Bedingungen. I_{Max}, $S_{relativMin}$ und H_{Max} sind dabei Parameter, mit denen der Benutzer die Farbdistanz-Funktion an die gegebene Situation anpassen kann. Durch die Erweiterung auf Farbkarten, die Farbmarken in Abhängigkeit ihrer Position auf dem Feld modellieren, genügen in nahezu allen Fällen die Standardparameter, und eine Anpassung durch den Benutzer ist nicht notwendig.

Das Aufwendige dieser Methode ist die Umrechnung aller Pixel der Region aus dem RGB-Raum in den HSI-Raum. Wie auf Seite 11 gesehen, enthält sie rechenaufwendige Funktionen wie den Arcus-Tangens oder die Quadratwurzel. Um diese zu umgehen habe ich eine Lookup-Tabelle entworfen, mit dessen Hilfe nur wenige billige Operationen für die Transformation notwendig sind.

Anhand der positiv klassifizierten Pixel P_+ kann ich nun die Kompaktheit des möglichen Markers kontrollieren und seine Qualität, d.h. die Ähnlichkeit zum Modell berechnen. Seine Position P_{center} wird auf den Schwerpunkt von P_+ gesetzt.

4.3.2.2 Maß für Kompaktheit

Um längliche Farbverfälschungen an Linien und Kanten auszuschließen, habe ich Kompaktheit als Kriterium ins Modell aufgenommen.

Setzen wir die Fläche eines Pixels auf eins, so besitzt ein Kreis K_+ zentriert auf P_{center} mit der Fläche aller positiv klassifizierten Pixel P_+ den Radius

$$r = \sqrt{|P_+| \cdot \pi}$$

Als Maß für die Kompaktheit Q_C verwende ich nun den prozentualen Anteil an Pixeln aus P_+, die auch in K_+ liegen, d.h.

$$Q_C = \frac{|P_+ \cap K_+|}{|P_+|}$$

Auf diese Weise können nun Objekte, die einen gewünschten Mindestgrad an Kompaktheit Θ unterschreiten, leicht zurückgewiesen werden.

Algorithm 4.2 Teste Kompaktheit

```
1:  if Θ > 0 then
2:     S = ∅
3:     r² = |P+| · π
4:     for all p ∈ P+ do
5:        if ‖p − P_center‖² ≤ r² then
6:           S = S ∪ {p}
7:        end if
8:     end for
9:     Q_C = |S|/|P+|
10:    if Q_C < Θ then
11:       return FALSE
12:    end if
13:    return TRUE
14: end if
```

4.3.2.3 Qualitätsfunktion

Neben der Kompaktheit brauchen wir auch noch ein allgemeines Maß für die Ähnlichkeit zum Modell, anhand dessen z.B. Ball- oder Robotersuche das Objekt akzeptieren oder zurückweisen kann. Zudem ist die Stärke der anschließenden Anpassung des Modells von dieser Qualität abhängig. Hierfür eignet sich wiederum der HSI-Raum. Zudem geht auch die Größe, d.h. die Anzahl der gefundenen Pixel $|P_+|$ in die Qualität ein.

Sofern $|P_+| \neq 0$ (ansonsten ist die Qualität $Q = 0$), berechne ich dazu den durchschnittlichen Hue H_{Avg}, wie auch die Saturierung S_{Avg} von P_+. Ähnlich wie bei der Klassifizierung erhalte ich

$$\delta_S = |S_{Modell} - S_{Avg}|$$

$$\delta_{Hue} = \Delta(H_{Avgas}, H_{Model})$$

für ihre Distanz zum Modell und zusätzlich

$$\delta_{Size} = \frac{|Size_{Modell} - |P_+||}{Size_{Modell}}$$

für die der Größe des Objekts. Bei letzteren normiere ich mit der Größe des Modells, so dass Unterschiede nur relativ bestraft werden. So stellt eine Differenz von 5 Pixeln bei

einem 10 Pixel großen Objekt einen erheblichen Unterschied dar, nicht aber bei einem 100 Pixel großen Objekt.

Die Gesamtdistanz wird nun als gewichtete Summe berechnet,

$$\delta_{Total} = W_{Total} \cdot (W_H \cdot \delta_{Hue} + W_S \cdot \delta_S + W_{Size} \cdot \delta_{Size})$$

und schließlich die Qualität wie folgt:

$$Q = \frac{1}{1 + \delta_{Total}}.$$

Dabei stellen die Gewichte wiederum Parameter dar, mit denen der Benutzer die Erkennung beeinflussen kann. Typischerweise ist hier der Hue besonders stark gewichtet. Wie schon bei den Parametern der Klassifizierung sind mit Einführung von Farbkarten die Standardwerte in nahezu allen Fällen hinreichend gut, um Farbmarker oder Ball überall auf dem Feld korrekt zu segmentieren, und eine Anpassung durch den Benutzer ist nicht notwendig.

4.3.2.4 Anpassen des Modells

Während ein Ball oder Roboter sich über das Feld bewegt und von dem Vision-System gefunden und verfolgt wird, wird sein Modell bei erfolgreicher Suche in Abhängigkeit zu seiner Qualität ständig angepasst. Auf diese Weise kann ein Objekt sich von sehr hellen Bereichen in sehr dunkle Bereiche bewegen, ohne verloren zu gehen, solange der Übergang nicht zu hart ist oder für die eingestellte Adaptionsrate zu schnell vonstatten geht, so dass sich das Modell nicht ausreichend anpassen kann.

Bei der Anpassung z.B. des Modells einer Farbmarke wird das alte Modell M_{Old} mit der aktuellen Messung, d.h. der aktuell gefundenen Instanz des Markers $I_{Current}$, fusioniert.

$$M_{New} = \lambda \cdot I_{Current} + (1 - \lambda) \cdot M_{Old}$$

Die Adaptionsrate λ ist dabei nicht für alle Komponenten gleich. So möchte man z.B. die Position in der Regel direkt übernehmen ($\lambda_{pos} = 1$), während die Farbinformation und Größe etwas vorsichtiger angepasst wird.

Hohe gegen niedrige Adaptionsrate Die Wahl der geeigneten Adaptionsrate ist nicht vollkommen trivial. Höhere Raten helfen, härtere Kanten schneller zu überqueren (z.B. von einem hellen Spot in einen harten Schatten), können aber leichter zu Überanpassung auf falsche Farben führen. Wie unter 3.1 beschrieben, führen Effekte wie Blooming oder Verschmierung teilweise zu glatten Farbübergängen zwischen zwei benachbarten Farbmarken (gelb und pink haben z.B. oft einen orangenen Übergang).

Qualitätsabhängige Adaptionsrate Mehr Sicherheit erreicht man durch eine qualitätsabhängige Adaptionsrate. Je höher die Qualität, desto sicherer wurde das richtige Objekt gefunden und desto ungefährlicher ist eine höhere Adaptionsrate. Hingegen bei niedriger Qualität ist eine vorsichtigere Adaptionsrate zu bevorzugen. Wie schon vorher erwähnt,

wird das Objekt erst ab einer Mindestqualität überhaupt als gefunden erachtet und auch nur dann angepasst.

Absolute Adaptionsgrenzen Um eine Anpassung auf gänzlich falsche Farben zu unterbinden, lassen sich zudem absolute Maximalgrenzen definieren. Dies bedeutet, dass z.B. bei der Anpassung der Farbe eines Modells niemals ein fester Maximalabstand zu einer mittleren Farbe des Modells überschritten werden darf.

Man könnte fälschlicher Weise annehmen, dass solche Maximalgrenzen doch schon während der Segmentierung realisiert werden können, indem die Schranken dort restriktiver gehalten werden. Um die Position eines pinken Markers möglichst präzise bestimmen zu können, möchte man allerdings sämtliche Pixel segmentieren, also auch die teilweise orange erscheinenden Pixel an Rändern oder Reflektionen.

Diese Probleme werden durch den Einsatz von Farbkarten wesentlich entschärft.

4.3.2.5 Wahl der zu segmentierenden Region

Nachdem in Schritt 1 Hypothesen gefunden wurden, werden diese in Schritt 2 in einer kleinen Region genauer betrachtet. Ich habe zwei Arten von Regionen implementiert: ein festes Rechteck und eine dynamisch wachsende Region. Beide Methoden haben Vor- und Nachteile.

Rechteck-Scan Hierbei wird ein fester Segmentierrahmen an der Stelle der Hypothese zentriert. Dieser sollte den gesamten Farbmarker enthalten, da sonst nicht alle Pixel des Markers segmentiert werden können, wodurch die Berechnung seines Schwerpunkts ungenauer wird und schließlich die Position und Orientierung von Ball und Robotern verrauscht. Da eine Hypothese auch am Rand eines Markers liegen kann, muss der Rahmen folglich mindestens viermal so groß sein, um diesen in jedem Falle ganz zu enthalten. Bei elliptischen oder allgemein weniger kompakten Markierungen sogar noch mehr. Auf der anderen Seite, darf dieser auch nicht zu groß sein, da er sonst einen Teil eines benachbarten Markers gleicher Farbe mitenthalten kann, und dieser die Segmentierung verfälscht. Da die Größe eines Markers bei stark verzerrenden Kameras sehr stark variiert, sollte die Größe des Segmentierrahmens abhängig von der des aktuellen Modells gewählt werden.

Wachsende Region In diesem Fall wird beginnend an der Stelle der Hypothese $|P_+|$ als zusammenhängende Region gewachsen (*region growing*). Hierbei begrenze ich das Wachstum durch ein maximales Rechteck, das aber im Gegensatz zum Segmentierrahmen recht groß sein darf, da beim *region growing* nur Pixel des zu segmentierenden Objektes angeschaut werden. Allerdings birgt dies Schwierigkeiten bei Kameras mit interlacing, wenn ein Marker derart auseinander gezogen wird, dass er nicht mehr perfekt zusammenhängend ist. Ein Marker kann auch durch Reflektionen und Blooming zerteilt werden.

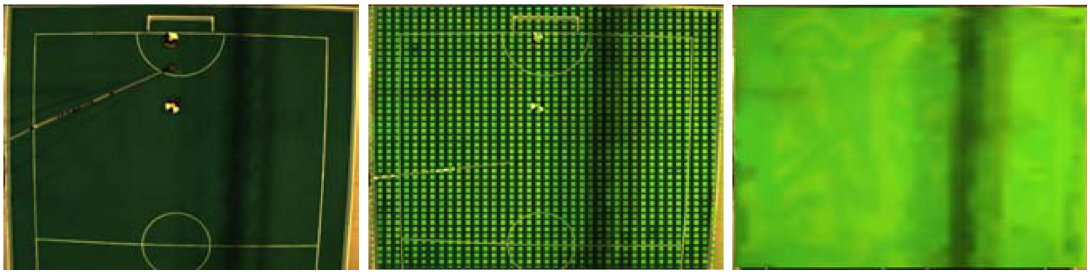


Abbildung 4.4: Man sieht eine Farbkarte für die Farbe Grün eines Markers. Gerade bei einem sehr ungleichmäßig beleuchtetem Feld mit harten Schatten (links) zeigt sich die Mächtigkeit von Farbkarten. Es werden ortsabhängige Prototypen des Marker für ein Gitter von Knotenpunkten gespeichert (mittig), und Zwischenwerte werden interpoliert (rechts).

4.4 Adaptive Farbkarten

Das System, wie bisher beschrieben, verwendet pro Marker einen Modell-Prototyp, anhand dessen dieser Marker gesucht wird und der bei erfolgreicher Suche adaptiert wird. Auf diese Weise lassen sich sowohl zeitliche Veränderungen als auch räumliche Unterschiede auf dem Feld bewältigen, sofern die Übergänge nicht zu hart sind. Zudem wächst der Aufwand durch den Benutzer je ungleichmäßiger die Beleuchtung und damit das Aussehen von Ball und Robotern, da die Standard-Parameter bei Klassifizierungs- und Qualitätsfunktion nicht mehr genügen und angepasst werden müssen.

Allerdings hat sich unter 3.1 gezeigt, dass sich die Farbe eines Markers, bewegt man ihn über das Feld, nicht nur in der Helligkeit, sondern auch stark in Saturierung und im Farbwert unterscheiden kann. Die größte Schwierigkeit dabei entsteht, wenn ein Marker global gefunden werden muss, d.h. ohne ihn kurz zuvor in der Nähe gesehen zu haben. Selbst wenn lokal keine harten Übergänge im Bild existierten, sind die Unterschiede global meist groß. Eine geeignete Klassifizierungs- und Qualitätsfunktion muss folglich einerseits global sehr tolerant sein, andererseits lokal möglichst restriktiv.

Um dieses Problem zu lösen, verwende ich anstelle eines einzigen Prototyps viele ortsabhängige Prototypen, die in einer uniformen Karte gleichmäßig über dem Feld verteilt sind, siehe Abbildung 4.4. Diese Farbkarte speichert in jedem Prototyp sowohl die Farbe in RGB als auch die Größe in Pixeln, und modelliert so das Aussehen eines Markers in Abhängigkeit seiner Position. Um das Aussehen an einer bestimmten Stelle zu erfragen, werden die jeweils vier benachbarten Prototypen in der Karte bilinear interpoliert. Die Auflösung der Karte kann abhängig von der Situation gewählt werden. Bei sehr ungleichmäßiger Beleuchtung, harten Schatten und Unstetigkeiten ist eine höhere Auflösung notwendig (z.B. 20×20 - 40×40), während bei diffusen Licht ohne harte Schatten auch niedrige Auflösungen genügen. Dabei teilen sich alle Marker einer Farbe die gleiche Karte. Falls unsere Roboter also z.B. gelbe, grüne und pinke Markierungen verwenden und der Gegner blaue und rosane, so werden wir je nachdem mit welchem Modell wir die Roboter verfolgen wollen (siehe Robotermodelle auf Seite 48) zu allen oder einigen von diesen jeweils eine Farbkarte erstellen.

4.4.1 Suchen mit Farbkarten

Um die Farbkarten nun auch zu verwenden, müssen ein paar, wenn auch geringfügige, Änderungen an dem bisherigen Suchalgorithmus vorgenommen werden.

Verifizieren von Hypothesen Anstelle den einzelnen Prototyp zum Verifizieren zu verwenden, wird nun zunächst an der Stelle der Hypothese das Aussehen des Markers in der Karte erfragt.

Finden von Hypothesen In diesem Schritt wurde zu jedem Pixel im Suchrahmen die RGB-Distanz zum Modell, d.h. zu dem einen Prototyp berechnet. Um dieses Prinzip auf Farbkarten zu erweitern, zerteile ich den Suchrahmen in Blöcke einer bestimmten Mindest- und Maximalgröße. Für jeden Block erfrage ich nun die durchschnittliche Farbe in der Karte und verwende diese innerhalb dieses Blocks als Referenzfarbe für die RGB-Distanz. Der einfachste Ansatz, an jedem Pixel die interpolierte Farbe aus der Karte zu holen, ist nicht akzeptabel, da in diesem Schritt des Algorithmus nur möglichst billige Methoden verwendet werden dürfen, siehe 4.3.1. Es bleibt die Frage, wie die Blockgröße zu wählen ist. Es ist klar, dass größere Blöcke die Auswirkung der Karte vermindern, während kleinere Blockgrößen höhere Rechenzeit bedeuten. Die feste Standardgröße von maximal 50^2 Pixeln pro Block hat für verschiedene Feldgrößen, Kameras, und Kameraauflösungen gute Dienste geleistet, und musste bisher niemals angepasst werden. Zukünftig könnte diese aber auch dynamisch bestimmt werden z.B. in Abhängigkeit zur Auflösung oder Regelmäßigkeit der Farbkarte.

4.4.2 Adaptation

Wie auch in der 1-Prototyp-Version wird das Modell und damit auch die Farbkarte bei erfolgreicher Suche an den aktuellen Zustand angepasst, wie unter 4.3.2.4 beschrieben. Allerdings wird die Karte nur lokal adaptiert an der Stelle des gefundenen Markers. Dabei passe ich nur die vier direkten Nachbarn an, wobei die Gewichtung zwischen diesen wie auch beim Erfragen eines Wertes aus der Karte bilinear bestimmt wird. Bei der eigentlichen Anpassung verwende ich *qualitätsabhängige Adaption* wie auch *absolute Adaptionsgrenzen*, wobei als mittleres Modell ein automatisch aus der Karte berechneter Mittelwert dient.

Zusätzlich hat der Benutzer noch die Möglichkeit eine allgemeine Adaptionsrichtlinie festzulegen, so kann diese stufenlos von *blockiert* bis *maximale Anpassung* gestellt werden.

4.4.3 Initialisierung

Wie wird nun eine solche Karte erstellt? Einerseites kann sie manuell bearbeitet, aber auch automatisch während des Spiels adaptiert werden.

Manuelle Manipulation Dies geschieht z.B. durch Anklicken eines Markers mit der Maus. Dabei wird mithilfe eines kleinen Gaussfilters eine Referenzfarbe bestimmt, zu

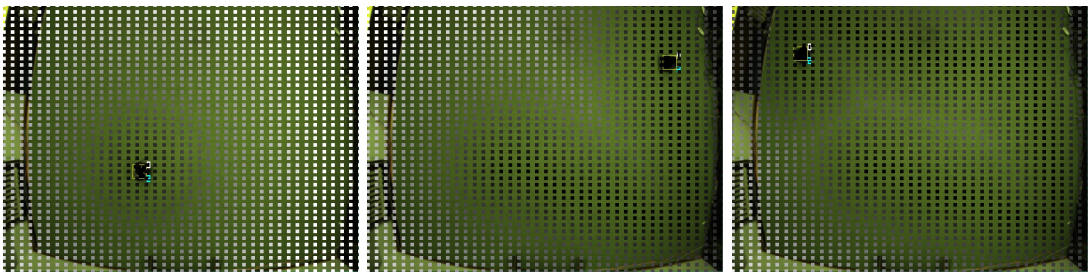


Abbildung 4.5: Man sieht die Distanzkarte einer Farbkarte während der Initialisierungsphase. Sie speichert die kleinste Entfernung, aus der die Farbkarte an dieser Stelle bisher adaptiert wurde. Je dunkler desto geringer diese Distanz. Während der Roboter über das Feld fährt und verfolgt wird, wird die Farbkarte angepasst und die Distanzkarte aktualisiert.

der – ähnlich wie bei der Klassifizierung (S. 30) – der angeklickte Marker segmentiert wird und somit seine Durchschnittsfarbe und Größe in Pixeln bestimmt werden können. Mehrmaliges Anklicken desselben oder verschiedener Marker gleicher Farbe führen zur Mischung von Farbe und Größe. Dieses so bestimmte Modell kann nun benutzt werden, um die Karte entweder uniform zu initialisieren oder wie mit einem Pinsel an ausgewählten Stellen, wie z.B. dunklen Bereichen am Rand oder bei harten Schatten, zu bearbeiten.

Dieses Verfahren ist allerdings – je nach Beleuchtung – verhältnismäßig zeitaufwendig. Schlauer ist es, die Adaptionsfähigkeit der Karte auszunutzen. Dabei werden die Marker des Roboters an einer Stelle einmal angeklickt, und so die zugehörigen Farbkarten uniform initialisiert. Während der Roboter nun über das Feld fährt, werden die Karten automatisch adaptiert. Zusätzlich habe ich ein verbessertes Verfahren implementiert, mit dem in einer Initialisierungsphase die Karten gelernt werden.

Initialisierungsphase Im Unterschied zur normalen Adaption können hierbei nicht nur die direkt benachbarten Prototypen, sondern jedes Mal die gesamte Karte angepasst werden. Dabei werden mit maximaler Adaptionsrate alle diejenigen Prototypen angepasst, die nicht bereits aus geringerer Distanz adaptiert wurden. Um dies zu gewährleisten, wird zunächst zu den zu lernenden Farbkarten jeweils eine Distanzkarte gleicher Auflösung erzeugt, die zu jedem Prototyp die Distanz speichert aus welcher er zuletzt angepasst wurde. Zu Beginn werden diese Karten mit ∞ initialisiert. Betrachtet man eine solche Distanzkarte als Grauwertbild, so markieren die hellen Stellen schlecht initialisierte Bereiche, während die dunklen Stellen bereits besucht sind, siehe 4.5.

Die Anpassung selbst kann für alle Prototypen gleich mit der aktuellen Messung vorgenommen werden. Dies kann zudem noch beschleunigt werden, indem zum Beispiel, wie unter [5] beschrieben, anhand schon bekannter Bereiche und der lokalen Helligkeit des Hintergrunds die jeweilige Farbe und Größe jedes einzelnen Prototyps geschätzt werden. Dabei wird versucht die Korrelation zwischen Farbe des Markers und dem Hintergrund zu extrahieren.

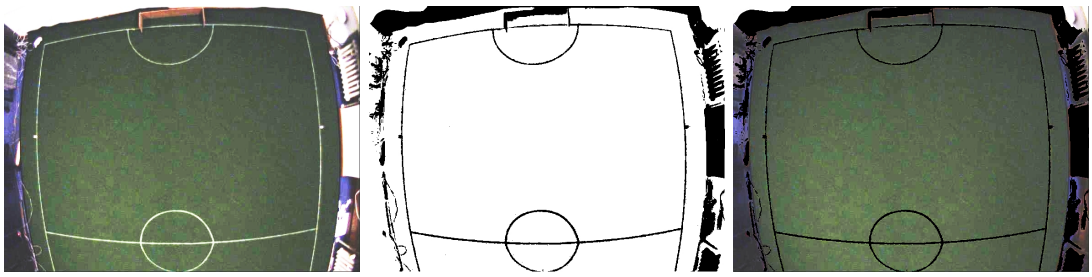


Abbildung 4.6: Links sieht man ein Bild vom Feld mit weißen Linien, die Fehlfarben enthalten, wie in Abb. 3.1 gezeigt. Mithilfe der automatisch erstellten Pixelmaske (Mitte) werden kritischen Bereiche von der Suche ausgeschlossen (rechts). Zudem können in Pixelmasken Bereiche manuell verboten werden.

4.4.4 Pixelmaskierung

Neben der *bounding-box* lässt sich zu jeder Farbkarte eine Pixelmaske definieren. Während die *bounding-box* die Suche auf einen rechteckigen Rahmen beschränkt, indem dieser den variablen Suchrahmen im Tracking Algorithmus als maximale Begrenzung dient, kann die Pixelmaske einzelne Pixel von der Suche ausschließen. Sie dient vor allem dazu, das Problem der Farbverfälschungen an weißen Linien zu lösen, kann aber benutzt werden, um beliebige Bereiche im Bild auszusparen (z.B. für den Ball im Übergangsbereich bei Mehrkamerasystemen, siehe 6.2.3).

Manipulieren der Pixelmaske Ähnlich wie bei den Farbkarten, können Pixelmasken manuell oder automatisch bearbeitet werden. So gibt es die Möglichkeit, wie mit einem Pinsel oder Radiergummi einstellbarer Strichstärke im Bild zu malen, und auf diese Weise Bereiche zu verbieten oder wieder freizugeben. Für spezielle Bereiche wie die Linien, das Feldäußere oder – wie später erläutert – den Übergangsbereich zwischen zwei Kameras habe ich automatische Methoden entworfen.

Automatisches Maskieren der Feldlinien Die Linien unterliegen wie die Farbmarker ebenfalls starken Helligkeitsschwankungen, daher entscheide ich anhand der lokalen Hintergrundhelligkeit, ob ein Pixel zu einer Linie gehört und in der Pixelmaske verboten wird. Hierzu erstelle ich eine niedriger aufgelöste Helligkeitskarte (ca. 20×20), für die ich an jedem Gitterpunkt die lokale Helligkeit berechne. Dazu ermittle ich zunächst in einem größeren lokalen Fenster um den Gitterpunkt eine mittlere Helligkeit. Anhand dieser wird in einem kleineren lokalen Fenster der Vordergrund ausgeschlossen und die lokale Hintergrundhelligkeit aus den restlichen Pixeln bestimmt. Der Zugriff auf diese Helligkeitskarte erfolgt, wie auch schon bei den Farbkarten, per bilinearer Interpolation der benachbarten Gitterpunkte.

Suchen mit Pixelmasken Neben dem Austanzen gefundener Objekte dient die Verwendung von Pixelmasken vor allem, falsche Hypothesen an den weißen Linien zu verhindern, die sonst das Finden des eigentlichen Markers vereiteln könnten. Dazu wird in Schritt 1

des Suchalgorithmus (siehe 4.3.1) allen maskierten Pixeln eine unendlich große Distanz zugeordnet. Während der Verifizierung einer Hypothese in Schritt 2 werden die Pixelmasken hingegen nicht verwendet, so dass auch Marker, die sich momentan teilweise im Bild auf einer Linie befinden, vollkommen segmentiert werden können. Die Pixelmaske markiert folglich nur Bereiche, in denen keine Hypothesen gefunden werden dürfen.

4.4.5 Diskussion und Ausblick

Mithilfe des Mechanismus der *Adaptiven Farbkarten* kann das System robust sowohl harte räumliche Unterschiede, wie harte Schatten und helle Spots durch ungleichmäßige Beleuchtung, als auch langsame zeitliche Veränderungen bewältigen. Allerdings müssen sich die Objekte über das Feld bewegen, damit die Karte die zeitliche Veränderung wahrnehmen kann.

Zukünftig könnte die Karte automatisch bestimmen, in welchen Bereichen sie nicht mehr aktuell oder fehlerhaft ist, indem sie misst, an welchen Stellen Roboter schlecht oder gar nicht mehr gefunden werden. Auf diese Weise könnte jedem Prototyp der Karte noch eine Konfidenz zugeteilt werden. An Stellen mit schlechter Konfidenz könnte nun vorgreifender adaptiert oder neue Schätzungen wie während der Initialisierungsphase berechnet werden. Zudem könnte sie den Hintergrund betrachten, um an beliebigen Stellen Veränderung in der Beleuchtung wahrzunehmen.

Auch nicht-statische nicht-uniforme Datenstrukturen wie Quadtrees, KD-Trees oder AMR-Gitter, die ihre lokale Auflösung dynamisch an die gegebene Unregelmäßigkeit des Feldes anpassen können, erscheinen geeignet. Allerdings bleibt es zu testen, inwieweit sich diese Alternativen negativ auf die Rechenlast auswirken.

5 Ball- und Robotersuche

In diesem Kapitel bespreche ich, wie einzelne Objekte, insbesondere Ball und Roboter, auf einem Spielfeld effizient verfolgt werden können.

5.1 Ballverfolgung

Mithilfe der beschriebenen Methoden ist es uns nun möglich, einen Ball im Bild zu verfolgen. Dazu müssen wir lediglich angeben, mit welcher Farbkarte der Ball gesucht werden soll. Da der Ball sehr häufig verdeckt ist, kann der variable Suchrahmen sehr groß werden und oft das gesamte Spielfeld abdecken. Hierbei erzielen die unter 4.3.1 erwähnten Beschleunigungsverfahren beim Finden geeigneter Hypothesen eine starke Verminderung der Rechenlast.

Dennoch gilt es, zwei Dinge zu beachten: Zunächst suche ich den Ball erst, nachdem die eigenen und gegnerischen Roboter gesucht wurden. Dies hat einen entscheidenden Vorteil: Viele Teams verwenden Ball-verwandte Farben wie rot oder pink, die von der Ballsegmentierung in jedem Fall zurückgewiesen werden müssten. Zudem können durch Demosaicing oder chromatische Abberation Fehlfarben entstehen, so auch orangene Pixel an Rändern von pinken Markern. Noch katastrophaler wirken sich orange Übergänge aus, die durch Blooming zwischen einem gelben und pinken Marker erzeugt werden. Hat man allerdings die Positionen der einzelnen Roboter bereits in der Hand, so lässt es sich leicht sicherstellen, dass der Ball nicht auf einem Roboter gefunden wird. Wie später beschrieben (siehe 5.2.4), erziele ich diesen Effekt, indem ich bereits erkannte Objekte aus dem Bild entferne.

5.1.1 Tracken geschossener Bälle

Eine weitere Schwierigkeit stellen geschossene Bälle dar. Normale Schussstärken sind dabei immer noch gut zu verfolgen. Allerdings erreichen geschossene Bälle inzwischen Geschwindigkeiten von bis zu $15 m/s$ und erscheinen dabei als semi-transparenter Schweif im Bild, dessen Aussehen mit dem Ball kaum noch etwas gemein hat. Ebenso können Hochschüsse – obgleich wesentlich langsamer geschossen – den Ball in ein vollkommen anderes Licht rücken. So ist dieser aufgrund der Höhe plötzlich wesentlich größer und weist auch farblich ein anderes Bild auf. Diese Information lässt sich natürlich nicht in der Farbkarte finden und soll auch gar nicht dort gespeichert werden, da der Ball wenige Momente später wieder mit normaler Geschwindigkeit rollt und sein bekanntes Aussehen aufweist. Die Farbkarte modelliert sozusagen den Ball unter nicht-hart geschossenen Bedingungen. Aber gerade ein hart geschossener Ball ist besonders torgefährlich, so dass eine möglichst frühe und gute Erkennung dem eigenen Torwart einen erheblichen Vorteil verschafft.

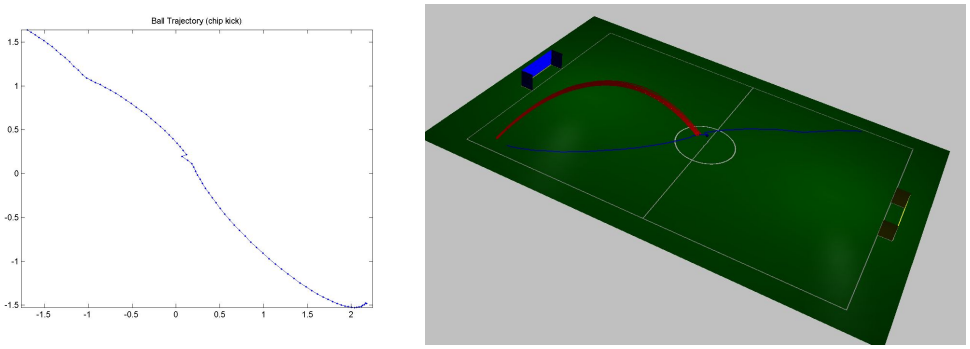


Abbildung 5.1: Links sieht man die vom Vision-System wahrgenommene Positionstrajektorie eines hochgeschossenen Balles. Diese verläuft von unten rechts nach oben links. Das rechte Bild, Screenshot aus dem Computer-Vision-Software, zeigt die rekonstruierte Flugbahn.

Um mit diesen speziellen Situationen auch umgehen zu können, verwende ich zusätzlich zur Farbkarte einen weiteren Prototyp, der mit hoher Adaptionsrate und sehr weiten Adaptionsgrenzen angepasst wird. Diesem wird allerdings zusätzlich zu seiner Qualität nur während einer solchen Schusssequenz vertraut, d.h.

- Zu Beginn der Sequenz hatte die Farbkarte den Ball im vorherigen Frame noch gefunden.
- Der erkannte Ball muss ein Schuss sein, d.h. eine gradlinige minimale Geschwindigkeit aufweisen.
- Eine Schusssequenz ist von kurzer Dauer, d.h. spätestens nach einer bestimmten Zeit wird dem Prototyp nicht mehr vertraut, da ein Schuss zu Ende sein müsste.
- Zwischen zwei Sequenzen muss die Farbkarte den Ball gefunden haben.

5.1.2 Erkennen von Hochschüssen

Immer mehr Teams verwenden in ihren Robotern einen Hochschuss, mit dem sie den Ball über die Verteidigung und den Torwart spielen können. Hochschüsse sind besonders torgefährlich, da sie aufgrund der Perspektive der Kamera zunächst vom Vision-System als ein auf dem Spielfeld in einer Kurve rollender Ball wahrgenommen werden. Anhand dieser scheint sich der Ball zunächst gar nicht in Richtung Tor zu bewegen und die Gefahr wird zu spät erkannt.

Aus diesem Grund habe ich in meinem System eine Erkennung von Hochschüssen integriert, wie sie in [15] im Detail beschrieben wird. Dabei wird nicht nur die Anwesenheit eines Hochschusses, sondern der exakte räumliche Parabelflug des Balls mit Auftreffpunkt und Auftreffzeit berechnet. Das Besondere dieser Methode ist, dass es mit einer einzigen Kamera auskommt, es ist keine Stereovision notwendig. In dem Verfahren wird

ein Punkt in Kamerakoordinaten auf der Flugbahn zum Zeitpunkt t mit

$$(x, y, z) = \left(x_0 + tv_x, y_0 + tv_y, z_0 + tv_z - \frac{1}{2}gt^2\right)$$

beschrieben, wobei (x_0, y_0, z_0) und $\vec{v} = (v_x, v_y, v_z)$ die korrekte Position und Geschwindigkeit des Balls zum Beginn der Messung ist, g entspricht der Gravitationskonstante. Für eine vom Vision-System beobachtete Kurve von m Punkten $(x'_1, y'_1), \ldots, (x'_m, y'_m)$ auf dem Feld, gilt folgender Zusammenhang

$$(\frac{x'_i}{z_c}, \frac{y'_i}{z_c}) = \left(\frac{x_0 + t_i\, v_x}{z_0 + t_i v_z - \frac{1}{2}gt_i^2}, \frac{y_0 + t_i\, v_y}{z_0 + t_i v_z - \frac{1}{2}gt_i^2}\right) \tag{5.1}$$

da die Projektion in die Kameraebene (z=0) für den von der Vision auf das Feld projizierten Ball und dem echten in der Luft gleich ist. $-z_c$ entspricht dabei der Höhe der Kamera. Aus (5.1) können nun 2 lineare Gleichungen mit 6 Unbekannten gewonnen werden, so dass im Minimalfall drei Punkte zur Lösung des Gleichungssystems genügen. Für bessere Genauigkeit können beliebig viele Punkte verwendet werden, indem das überbestimmte Gleichungssystem

$$D(z_0, v_z, x_0, v_x, y_0, v_y)^{\mathrm{T}} = d$$

mittels der Pseudoinversen $D^+ = (D^T D)^{-1} D^T$ von D gelöst wird. D entspricht dabei der $2m \times 6$ Datenmatrix und d einem $2m$-dimensionalen Vektor, die aus Gleichung 5.1 extrahiert werden.

In unserem System genügen 10 Punkte für eine gute Näherung, ab 20 Punkten erreicht die Erkennung eine Genauigkeit des Auftreffpunktes unter einem Zentimeter. Abbildung 5.1 zeigt einen Screenshot aus der Vision-Software. Die gezeigten Parabelflüge eines Hochschusses verwendeten bei der Erkennung 10 bis 20 Punkte, wobei die berechneten Auftreffpunkte sehr nah beieinander liegen. Abbildung 5.2 zeigt einen Hochschuss, wie er von einer Kamera beobachtet wird.

Ich habe das Verfahren hier nur kurz angerissen, eine genaue Beschreibung findet sich unter [15].

5.2 Robotersuche

Um einen Roboter im Bild zu finden, werden zunächst – je nach verwendetem Modell – eine oder mehrere seiner farbigen Markierungen in ihren variablen Suchrahmen gesucht. Dies geschieht mithilfe des unter 4.1-4.4 erläuterten Algorithmus. Sobald nun alle Markierungen mit ausreichend guter Qualität gefunden wurden, wird deren geometrische Anordnung zueinander mit der des Modells verglichen. Ähnlich wie beim Verfolgen von Markierungen gilt ein Roboter nur als gefunden, falls seine Qualität ausreichend gut ist. Diese setzt sich aus den Qualitäten der einzelnen Marker und ihrer geometrischen

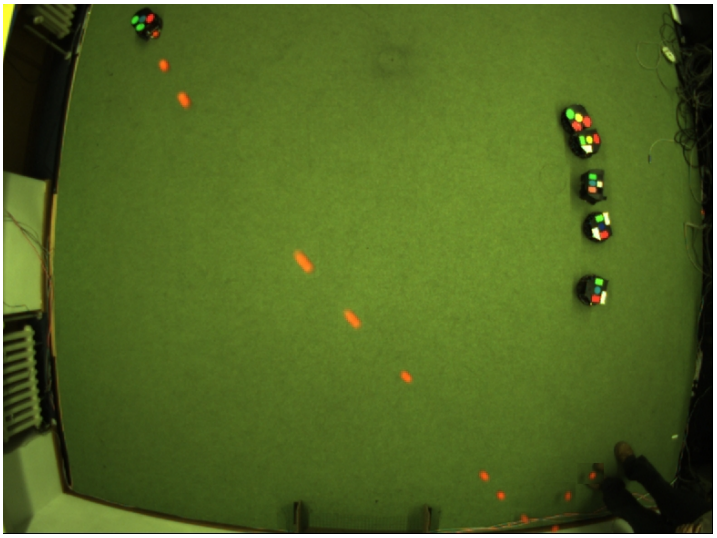


Abbildung 5.2: Zu sehen ist eine Hochschusssequenz (es wurden nicht alle Frames eingezeichnet). In diesem Fall fliegt der Ball direkt unter der Kamera durch, weshalb keine Kurve zu sehen ist. Dennoch kann anhand der Abstände zueinander die korrekte Parabel ermittelt werden.

Anordnung zusammen:

$$Q_{Roboter} = \sum Q_{Marker} + Q_{Geometrie}$$

Anhand der Geometrie des Modells kann nun aus den Positionen der gefundenen Markierungen die Position des Roboters berechnet werden. Zusätzlich werden oftmals weitere Module verwendet, die nun entweder anhand der bereits gefundenen Marker oder mittels weiterer Untersuchung des Bildes um die Position des Roboters herum die Orientierung oder Identifizierung des Roboters bestimmen. Solche optionalen Orientierungs- oder Identifizierungsmodelle können zudem als zusätzliche Verifizierung des gefundenen Roboters dienen. So scheint ein Roboter, auf den sein Orientierungs- oder Identifizierungsmodell nicht passt, nicht korrekt zu sein und kann zurückgewiesen werden. Überdies kann oftmals auch anhand der zusätzlich extrahierten Information die Positionsbestimmung des Roboters verbessert werden.

Ist nun ein Roboter gefunden, so werden die Modelle seiner Marker und damit die zugehörigen Farbkarten, wie unter 4.3.2.4 bzw. 4.4.2 angepasst. Entscheidend hierbei ist, dass ein Marker erst adaptiert wird, wenn auch der Roboter zufrieden stellend gefunden wurde. Zudem verwende ich eine qualitätsabhängige Adaption bezüglich der Qualität des Roboters, nicht der des Markers selbst. Auf diese Weise kann ein weniger gut gefundener Marker von den besser gefundenen profitieren.

Allerdings stellen sich zwei Probleme: Die Roboter eines Teams (oftmals auch beider Teams) verwenden Markierungen gleicher Farbe, wie in Abbildung 5.4 zu sehen. So befinden sich in der Summe teilweise an die 15 korrekte grüne Marker auf die Roboter beider Teams verteilt. Aus diesem Grund kann es passieren, dass die in ihren variablen

Suchrahmen gefundenen Marker zu einem Roboter in Wirklichkeit auf unterschiedlichen Robotern liegen, und somit dieser nicht gefunden werden kann, da seine Geometrie nicht erfüllt wird. Ebenso können mehrere gesuchte gleichfarbige Marker auf ein- und demselben Marker eines Roboters gefunden werden. Diese Effekte treten vor allem bei großen Suchrahmen auf, die mehrere Roboter enthalten können, während dies bei kleinen Rahmen eher unwahrscheinlich ist. Zudem bedeuten viele große Suchrahmen eine erhebliche Rechenlast, so dass für das Durchsuchen größerer Bereiche aus diesen beiden Gründen eine schlauere Methode sinnvoll erscheint.

In diesem Sinne habe ich die Suche in eine lokale Suche, die auf kleine Suchrahmen beschränkt ist, und eine globale Suche unterteilt, die alle Roboter, die von der lokalen Suche verloren wurden, global im Bild mit aufwendigerer Methode wieder findet.

5.2.1 Lokale Suche

Während der lokalen Suche sind die Suchrahmen der einzelnen Marker auf eine bestimmte Größe beschränkt. Um die Wahrscheinlichkeit zu minimieren, dass einzelne Marker auf andere Roboter überspringen können, verwende ich zusätzliche Mechanismen: Erstens lassen sich solche einzelnen vertauschten Marker anhand von Redundanz im Modell erkennen und korrigieren, wie in den jeweiligen Modellen unter 5.2.3 beschrieben. Zweitens hilft es, die lokale Suche der Roboter zu synchronisieren: Dazu werden alle Marker aller Roboter zunächst in ihren kleinsten Suchrahmen gesucht. Nun werden die Suchrahmen aller nicht gefundenen Marker vergrößert und erneut gesucht, bis die maximale Suchrahmengröße erreicht ist. Dabei wird ein Roboter, sobald er gefunden ist, aus dem Bild entfernt, wie unter 5.2.4 beschrieben wird. Auf diese Weise ist es für einen Marker, der in einem größeren Rahmen gesucht werden muss, unwahrscheinlicher überzuspringen, da die meisten Marker bereits vorher in den kleineren Suchrahmen gefunden und aus dem Bild entfernt wurden.

5.2.2 Globale Suche

Da die Suchrahmen während der lokalen Suche beschränkt sind, kann es passieren, dass ein oder mehrere Roboter verloren gehen. Vor allem zu Beginn, wenn ein neuer Roboter das Feld betritt, ist seine Position der lokalen Suche noch nicht bekannt, und er muss global gesucht werden. Es können auch falsche Hypothesen der einzelnen Marker während der lokalen Suche dazu führen, dass Roboter verloren gehen. Dies ist natürlich nicht der Normalfall, so dass die globale Suche – abgesehen von Verdeckungen durch den Menschen, wie z.B. den Schiedsrichter, der den Ball positioniert – nur in den seltensten Fällen ausgelöst werden muss. Aus diesem Grund ist es möglich, eine aufwendigere Methode zu verwenden, die mit hoher Sicherheit alle verlorenen Roboter findet, und unanfällig für Rauschen oder Fehlfarben ist.

Dazu berechne ich in einem ersten Schritt zu jeder Farbkarte eine Liste von Markern, die zu dieser Karte gehören. In einem zweiten Schritt werden aus diesen Listen geeignete Kombinationen von Markern gefunden, die die geometrische Anordnung des Modells und gegebenenfalls Orientierungs- und Identifizierungsmodell erfüllen. Ist nun die Qualität

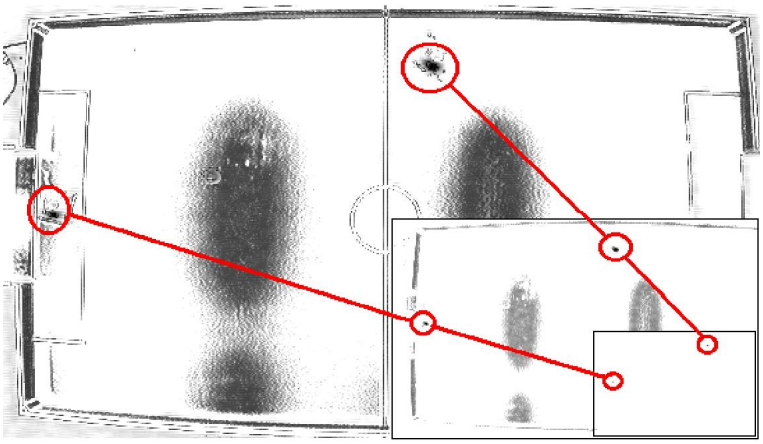


Abbildung 5.3: Man sieht das Differenzbild zur Farbkarte eines Markers und das einfache und zweifache Subsample, bei denen jeweils vier benachbarte Pixel zusammengefasst werden. Während im Differenzbild Rauschen enthalten ist, sind die beiden Marker auf den zwei Robotern klar identifizierbar.

des so gefundenen Roboters ausreichend, wird im letzen Schritt der zugehörige verlorene Roboter bestimmt und seine lokale Suche reinitialisiert.

Schritt 1 – Berechne Liste von Markern Hierfür wird zunächst zu jeder Farbkarte des Robotermodells ein Differenzbild mittels RGB-Distanz berechnet. Dabei wird wiederum für einzelne Blöcke eine neue Referenzfarbe aus der Karte geholt, wie unter 4.4.1 beschrieben. Um einerseits den Suchraum zu verkleinern und andererseits Rauschen und Fehlfarben einzelner Pixel zu entfernen, wird dieses Differenzbild nun zweimal in der Auflösung reduziert, wie in Abbildung 5.3 zu sehen. Dabei werden jeweils vier benachbarte Pixel aufsummiert und zu einem zusammengefasst. Auf diese Weise werden einzelne gute Pixel gegenüber echten Markern bestraft. In diesem zweifachen *Subsample* werden nun die N besten Pixel bestimmt. Diese müssen dabei einen minimalen Abstand zueinander einhalten, so dass keine zwei auf ein und demselben Marker liegen können. An diesen N Stellen wird nun die genauere Position im einfachen Subsample und schließlich im Originalbild bestimmt, und dort der Marker wie unter 4.3.2 verifiziert. Alle Marker mit ausreichend guter Qualität werden nun sortiert in einer Liste zu der entsprechenden Farbkarte gespeichert.

Schritt 2 – Kombinieren der Marker Aus Schritt 1 erhalte ich nun zu jeder Farbkarte eine Liste möglicher Marker sortiert nach ihrer Qualität. Aus diesen bestimme ich nun Kombinationen, die der geometrischen Anordnung des Modells entsprechen und teste - falls vorhanden, ob das Orientierungs- bzw. Identifizierungsmodell die Existenz des Roboters bestätigt.

Schritt 3 – Reinitialisierung der lokalen Suche Falls nun ein Roboter mit ausreichend guter Qualität gefunden wird, muss dieser dem richtigen verlorenen Roboter zugeordnet

werden. Dies geschieht entweder anhand der Position, so dass die Gesamtsumme der Abstände minimiert wird, oder falls ein Identifizierungsmodell vorliegt, direkt anhand der Identität des Roboters. Die lokale Suche kann nun reinitialisiert werden, indem die entsprechenden Felder des Modells (Position und Geschwindigkeit des Roboters und der einzelnen Marker, Farbkarten, etc.) auf die übliche Weise angepasst werden.

Schritt 2 und 3 werden nun solange wiederholt bis entweder kein Roboter mehr verloren ist, oder kein ausreichend guter Roboter gefunden wird.

Algorithm 5.1 Globale Suche von X verlorenen Robotern eines Modells M

1: **for all** Farbkarten des Modells M **do**
2: $\Delta_{Bild} \leftarrow$ BerechneRGBDifferenzZurFarbkarte(Bild)
3: ${\Delta_{Bild}}^{s} \leftarrow$ Subsample(Δ_{Bild})
4: ${\Delta_{Bild}}^{ss} \leftarrow$ Subsample(${\Delta_{Bild}}^{s}$)
5: Finde die N minimalen Positionen in ${\Delta_{Bild}}^{ss}$
6: Finde die entsprechenden minimalen Positionen in ${\Delta_{Bild}}^{s}$ und Δ_{Bild}
7: Versuche die Existenz eines Markers an diesen Stellen zu verifizieren
8: Merke Liste aller korrekt verifizierten Marker sortiert nach Qualität
9: **end for**
10: **while** Anzahl verlorener Roboter > 0 **do**
11: Suche Kombination von Markern, die die geometrische Anordnung des Modells und gegebenenfalls Orientierungs- und Identifizierungsmodell erfüllen
12: Reinitialisiere die lokale Suche des zugehörigen Roboters
13: **end while**

5.2.2.1 Stabile Framerate

Wie schon erwähnt, ist die globale Suche verhältnismäßig rechenintensiv, so dass bei einer hohen Framerate die Zeit eines Frames nicht zur Berechnung ausreicht und das nächste Frame verloren geht. Bei Aufruf der globalen Suche in jedem Frame bedeutete dies bei einer Framerate von z.B. 50Hz, dass jedes zweite Frame verloren geht und die Rate auf 25Hz singt. Auch wenn der Aufruf einer globalen Suche im Allgemeinen äußerst selten notwendig ist, gibt es dennoch Situationen, in denen sie jeden Frame aufgerufen werden würde. So z.B. wenn ein eigener oder gegnerischer Roboter durch zu starkes Bremsen umfällt, oder aufgrund schlechter Befestigung sich Teile eines Covers (z.B. Marker) lösen. Dies klingt zwar eher ungewöhnlich, aber es passiert in der Tat immer wieder. Zudem können natürlich auch schlechte Kalibrierung oder plötzliche Veränderungen der Lichtverhältnisse in bestimmten Bereichen des Feldes dazu führen, dass Roboter dort schlecht gefunden werden und die globale Suche öfter als gewöhnlich aufgerufen wird.

Wenn nun in solchen Situationen die Framerate einbrechen sollte und sich damit die Kontrolle der Roboter – sei es auch nur kurzzeitig – erheblich verschlechtert, kann dies schnell ein Gegentor bedeuten und somit den Sieg des Spiels kosten. Zudem erhöht sich die Wahrscheinlichkeit bei stark verringerter Framerate, dass schnell bewegende Objekte von der lokalen Suche verloren werden können. Ich erwarte von meinem System daher in allen Situationen eine weitgehend stabile Framerate zu erhalten. In diesem Sinne habe

ich einen Mechanismus entworfen, der dafür sorgt, dass die globale Suche nicht zu oft aufgerufen wird, so dass die Framerate stabil bleibt.

Ich stelle folgendes fest: Objekte, die gerade lokal verloren wurden, sollen möglichst sofort d.h. noch im selben Frame global gesucht werden. Hingegen für Objekte, die auch schon seit längerem global nicht gefunden werden konnten, steigt die Wahrscheinlichkeit, dass sie erneut nicht gefunden werden, und es lohnt nicht in jedem Frame mit hohem Aufwand nach ihnen zu suchen.

Zu diesem Zweck definiere ich eine Schranke Θ. Eine globale Suche darf nun erst initiiert werden, wenn mindestens Θ viele Frames seit dem letzten Aufruf vergangen sind. Dabei wird Θ um ein δ erhöht, wenn eine globale Suche erfolglos aufgerufen wurde (d.h. es konnten nicht alle verlorenen Roboter wieder gefunden werden). Bei erfolgreicher globaler Suche wird es um δ verringert. Dabei bleibt Θ stets zwischen Null und einer oberen Schranke GS_{Max}, um unnötig hohe Wartezeiten zwischen zwei globalen Suchen zu vermeiden. Anhand der Parameter δ und GS_{Max} lässt sich so das worst-case Verhalten des Algorithmus regulieren und an die gegebene Rechenleistung und gewünschte Framerate anpassen.

5.2.3 Robotermodelle

Die grobe Struktur der Robotersuche habe ich nun umrissen, allerdings bin ich noch nicht im Detail auf das jeweilige Robotermodell eingegangen, das verwendet werden kann, d.h. wie viele Marker werden verfolgt, wie ist ihre geometrischen Anordnungen zueinander, welche Orientierungs- und Identifizierungsmodelle gibt es und wie werden diese verifiziert bzw. berechnet.

Roboter in der Small-Size Liga sind stets durch einen gelben bzw. blauen zentralen Teammarker gekennzeichnet, sowie durch weitere optionale Markierungen, die für die Orientierung und Identifizierung der Roboter verwendet werden. Ihre Anzahl und geometrische Anordnung variieren dabei von Team zu Team. Daher habe ich mehrere Modelle entworfen, die dieses Spektrum möglichst gut abdecken sollen. Im Folgenden werde ich auf die einzelnen Modelle, die ich implementiert habe genauer eingehen.

5.2.3.1 1/2/3-Marker-Modelle

1-Marker-Modell Das denkbar einfachste Modell. Hierbei wird nur der Teammarker getrackt, daher kann es immer angewandt werden, da dieser bei jedem Roboter vorhanden sein muss. Aufgrund mangelnder Redundanz bietet es allerdings keine zusätzliche Sicherheit: Da nur ein Marker verfolgt wird, genügt eine falsche Hypothese für diesen Marker, um den Roboter in der lokalen Suche zu verlieren. Zudem kann anhand des einen Markers keine Orientierung berechnet werden, daher eignet sich dieses Modell nur für gegnerische Roboter. Allerdings kann es mit einem Orientierungsmodell kombiniert werden, welches um den Teammarker zusätzliche Information aus dem Bild extrahiert und so einerseits die Orientierung berechnet, andererseits höhere Sicherheit erlangt werden kann. Unterschiedliche Orientierungsmodelle werde ich später besprechen.

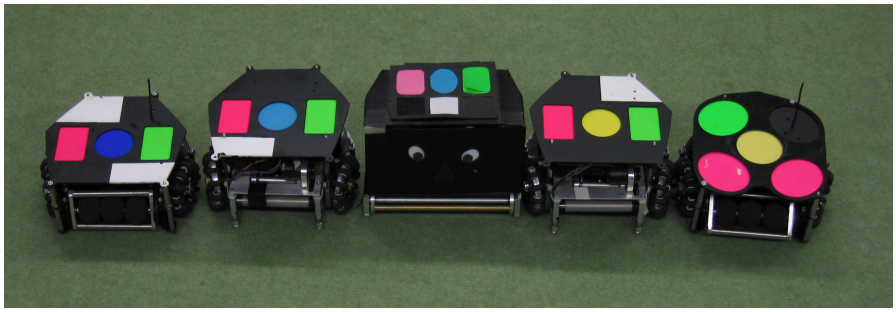


Abbildung 5.4: Fünf Roboter aus mehreren Generationen der FU-Fighters. Die ersten vier von links verwenden das 3-Marker-Modell mit Schwarz-Weiß Kodierung zur Identifizierung. Der Mittlere verwendete hierfür drei Bits auf einer Seite, die anderen vier Bits, je 2 auf einer Seite. Der rechte Roboter besitzt das FU2004-Modell. Allerdings könnten alle auch mit dem 1-Marker oder 2-Marker Modell verfolgt werden.

2-Marker-Modell Hierbei werden zwei Marker unterschiedlicher Farbe verfolgt. Diese haben einen festen Abstand δ zueinander, der während der Suche bis auf eine gewisse Toleranz T_δ erhalten bleiben muss. Der Abstand kann entweder fest eingegeben oder anhand Anklickens des Roboters ermittelt werden. Das Kontrollieren der geometrische Anordnung während der Suche, d.h. in diesem Fall den Abstand der Marker zueinander, sollte nicht im Pixelraum erfolgen, da aufgrund starker Verzerrung der Pixelabstand erheblich schwankt. Stattdessen werden die Positionen der Marker zuvor in Weltkoordinaten transformiert. Wie die entsprechende Transformationsfunktion, die sowohl starke Verzerrung, projektive Transformation, als auch die Höhe der Roboter berücksichtigt, berechnet werden kann, erkläre ich in Kapitel 6. Die Position des Roboters ergibt sich als gewichtetes Mittel beider transformierter Marker ${M_1}^t$ und ${M_2}^t$,

$$Pos = \gamma * {M_1}^t + (1 - \gamma) * {M_2}^t$$

die Orientierung ergibt sich aus der Gerade durch die Marker, gedreht um einen einstellbaren Winkel.

$$Or = \overline{{M_1}^t {M_2}^t} \text{ gedreht um } \alpha$$

γ, α, δ und T_δ sind Parameter des Modells.

3-Marker-Modell Dieses Modell besteht aus drei equidistanten kollinearen Markern unterschiedlicher Farbe. Die Berechnung von Position und Orientierung, sowie die Verifizierung der geometrischen Anordnung verlaufen analog zum 2-Marker-Modell. Abgesehen davon, dass drei Marker grundsätzlich mehr Sicherheit bieten als zwei oder einer, um auf die Existenz eines Roboters zu schließen, enthält das Modell außerdem Redundanz, die ausgenutzt werden kann: So kann ein Marker, der z.B. aufgrund einer fehlerhaften Hypothese nicht gefunden werden konnte oder der auf einen anderen Roboter übergesprungen ist, erkannt werden und seine korrekte Position aus den anderen beiden Markern bestimmt werden. Es genügt also, wenn während des Tracking zwei der drei

Abbildung 5.5: Mehrere Roboter mit 3-Marker-Modell und 4-Bit Schwarz-Weiß Kodierung. Alle Roboter sind nach links ausgerichtet. Das niederwertigste Bit der Kodierung ist vom Roboter aus gesehen vorne links, das höchstwertigste hinten rechts. Von links nach rechts sind folgende ID's zu sehen: 2, 7, 11, 1, 10, 13, 12, 5.

Marker gefunden werden, da die Position des dritten sofort bestimmt, und seine Existenz an dieser Stelle mit der üblichen Methode (siehe 4.3.2) verifiziert werden kann.

5.2.3.2 Orientierungs- und Identifizierungsmodelle

Wie schon erwähnt, biete ich in meinem System die Möglichkeit die beschriebenen Robotermodelle, die zum Verfolgen der Roboter verwendet werden, mit einem Orientierungs- und Identifizierungsmodell zu kombinieren. Diese ermöglichen nicht nur die Orientierung und Identität des jeweils gefundenen Roboters zu berechnen, sondern erlauben es auch die Genauigkeit der Positionsbestimmung durch zusätzliche Information zu verbessern und dienen als weiteres Kriterium zur Verifizierung.

Anhand der gefundenen Identität des Roboters, kann dieser, falls er auf einen anderen Roboter übergesprungen ist, wieder zurückgetauscht und seinem korrekten Modell zugeordnet werden. Aufgrund höherer Konfidenz akzeptiere ich dabei nur richtige Permutationen, d.h. z.B. $1 \rightarrow 4 \rightarrow 5 \rightarrow 1$ wäre eine korrekte Vertauschung, $1 \rightarrow 3, 3 \rightarrow 3$ hingegen nicht. Dazu finde ich in der Menge der falsch zugeordneten Identitäten Kreise, die ich zyklisch zurücktausche.

In meiner Implementation habe ich darauf geachtet, dass neue Modelle sehr leicht hinzugefügt werden können. Momentan muss dazu nur ein Interface mit ein paar wenigen Methoden implementiert und dieses mit einem Registrierungsbefehl dem System bekannt gegeben werden. Später könnten solche Modelle auch als DLL´s automatisch geladen werden, um eine erneute Kompilierung des Gesamtsystems zu vermeiden.

Im Folgenden werde ich einige Modelle beschreiben, die ich implementiert habe und die für unsere eigenen oder Roboter anderer Teams eingesetzt wurden.

5.2.3.3 3Bit/4Bit Schwarz-Weiß Kodierung

Für die ersten Generationen unserer Roboter verwendete ich das 3-Marker-Modell kombiniert mit einer 3-Bit bzw. einer 4-Bit Schwarz-Weiß Kodierung zur Identifizierung. Wie in Abbildung 5.5 zu sehen, sind dabei parallel zu den drei kollinearen Farbmarkierungen schwarze und weiße Flächen angeordnet, die jeweils ein Bit der Identität darstellen. Nachdem ein Roboter gefunden wurde, kann ausgehend von den drei Markern an diesen Stellen nach der Existenz von Schwarz oder Weiß geschaut und daraus die Identität des Roboters bestimmt werden. Es lassen sich damit maximal 8 bzw. 16 verschiedene ID's darstellen, allerdings vermeide ich einfarbige ID's, d.h. Null und 8 bzw. 16, so dass

immer mindestens eine schwarze und eine weiße Fläche vorhanden ist. Die einzelnen Schritte im Detail verlaufen wie folgt:

- Ich berechne die zentralen Positionen der einzelnen Flächen anhand der Marker in Weltkoordinaten und transformiere sie zurück in den Pixelraum.
- Mithilfe eines kleinen Gaussfilters berechne ich für jede Fläche die Helligkeit I. Die hellste wird als weiß, die dunkelste als schwarz klassifiziert.
- Die restlichen Flächen werden anhand eines Schwellenwerts Θ klassifiziert. Dazu verwende ich die mittlere Helligkeit, d.h. $\Theta = (I_{Max} + I_{Min})/2$

Um eine Aussage zu treffen, wie sicher die korrekte Identität gefunden wurde, akzeptiere ich diese nur, falls bestimmte Konfidenzkriterien erfüllt sind:

- Die minimale (schwarz) und maximale (weiß) Filterantwort müssen ausreichenden Abstand zueinander besitzen, d.h. $|I_{Max} - I_{Min}| > \delta$.
- Da ich nach uniformen Flächen schaue, erwarte ich einen weitgehend uniformen Filter, d.h. die Varianz der Helligkeit aller Pixel innerhalb eines Filters darf nicht zu groß sein. Dies könnte z.B. geschehen, wenn eine Fläche nicht zentral, sondern am Rand getroffen wird und somit die Filterantwort verfälscht und daher nicht vertrauenswürdig ist.
- Der relative Abstand einer Filterantwort F zum Schwellenwert muss ausreichend groß sein, so dass die Fläche sicher zu Schwarz oder Weiß zugeordnet werden kann, d.h.
$$\frac{|F - \Theta|}{|I_{Max} - I_{Min}|} > \gamma$$
- Es muss dieselbe Identität über mehrere Frames erkannt werden, so dass einzelne falsche Ausreißer zurückgewiesen werden.

Auf diese Weise kann das Erkennen falscher Identitäten vermieden werden. Falls nun eine dieser Kriterien verletzt wird, ist die Identifizierung fehlgeschlagen und die zuletzt erkannte Identität bleibt erhalten.

Dieses Verfahren wurde erfolgreich auf mehreren Turnieren eingesetzt. Verbesserte Modelle auf Basis weiterer Farbmarkierungen, die ohne derartige Konfidenzkriterien auskommen, beschreibe ich im folgenden Abschnitt.

5.2.3.4 Modelle mit radialem Scan

Viele Teams verwenden zusätzlich zum Teammarker weitere Markierungen, die auf einer Kreisbahn um den Teammarker angeordnet sind. Anhand dieser können sowohl Orientierung als auch Identität eines Roboters bestimmt werden. Unterschiedliche Teams setzten dabei drei, vier und mehr zusätzliche Marker ein, die zudem jeweils zwei oder drei Farben annehmen. Dabei können aus Gründen der Symmetrie nicht immer alle Kombinationen an Farben benutzt werden. Andere Teams, wie z.B. 5dpo aus Portugal,

verwenden anstelle von farbigen Markern schwarze und weiße Flächen, die kreisförmig um den Teammarker angeordnet sind.

Um solche Modelle mit höherer Sicherheit und größerem Informationsgehalt zu tracken als ausschließlich mittels des Teammarkers, kombiniere ich, wie schon vorher erwähnt, das 1-Marker-Modell mit einem geeigneten Orientierungs- und Identifizierungsmodell. Um nun die kreisförmig angeordneten Marker zu extrahieren, verwende ich in diesen Modellen ein Verfahren mit radialem Scan um den Teammarker herum. Dabei gehe ich wie folgt vor:

- Zunächst bestimme ich Kreispunkte mit festem Radius r um den Teammarker herum mit einem maximalem Abstand zueinander von ca. $\delta = 7mm - 1cm$. Da Kreise aufgrund der starken Verzerrung und der projektiven Verschiebung der Kamera nicht als solche im Bild erscheinen, berechne ich die Kreispunkte in Weltkoordinaten und konvertiere sie mithilfe der Transformationsfunktion, die in Kapitel 6 beschrieben wird, zurück in den Pixelraum. Dabei werden doppelt vorkommende Pixel entfernt.

- Nun ermittle ich mit einem kleinen 3x3 Gaussfilter an diesen Pixeln die Farbe im Bild und speichere sie sortiert nach Qualität, d.h. nach RGB-Abstand zur Farbkarte. Dabei werden zu stark abweichende Pixel sofort ausgefiltert.

- Diese Liste gehe ich der Qualität nach durch und versuche an jeder Stelle einen Marker wie gewöhnlich zu verifizieren. Dabei kommen Pixel mit zu geringem Winkel zu bereits erkannten Markern nicht mehr in Frage und werden übergangen, so dass keine Markierungen doppelt erkannt werden.

- Die letzten beiden Schritte werden für alle vom Modell verwendeten Farbkarten durchgeführt. Schließlich werden alle so erkannten Marker des Roboters aller Karten in einer Liste nach Winkel sortiert abgespeichert. Dabei speichere ich zu jedem Marker seine berechnete Position, Farbe und Qualität und die zugehörige Farbkarte.

Im Folgenden bespreche ich einige Modelle, die ich implementiert habe und die von diesem Verfahren Gebrauch machen.

Cornell-Modell Dieses Modell wurde von der Cornell Universität eingeführt. Wie in Abbildung 5.6 zu sehen, verwendet es vier radiale Marker mit drei möglichen Farben. Allerdings bleiben aufgrund von Symmetrie von den 3^4 Kombinationsmöglichkeiten nur 15 übrig, die gegenüber Rotation eindeutig sind. Hierbei wird sowohl die Orientierung, als auch die Identität des Roboters allein aus der Farbkombination berechnet. Dazu speichere ich zu jeder Kombination inklusive ihrer zyklischen Verschiebungen die Position des Markers links vorne und die zugehörige ID in einer Hashtabelle.

Damit kann nun die Orientierung, wie auch die Position des Roboters mit erhöhter Genauigkeit aus allen fünf Markern bestimmt werden. Wie üblich werden alle Marker zur Berechnung der Qualität des Roboters herangezogen. Falls nun zu viele oder zu wenig Marker während des radialen Scan-Verfahrens erkannt wurden, scheint kein korrekter Roboter gefunden worden zu sein, und er wird zurückgewiesen. Wie auch bei

den folgenden Modellen lässt sich die Identifizierung einstellen, dass alle möglichen ID's (hier also 15) akzeptiert werden, oder restriktiv nur die, die tatsächlich auf dem Feld vorhanden sind und der Software bekannt gegeben wurden. Nachteil dieses Modell ist, dass vier verschiedene Farben (inklusive Teamfarbe) zu kalibrieren sind.

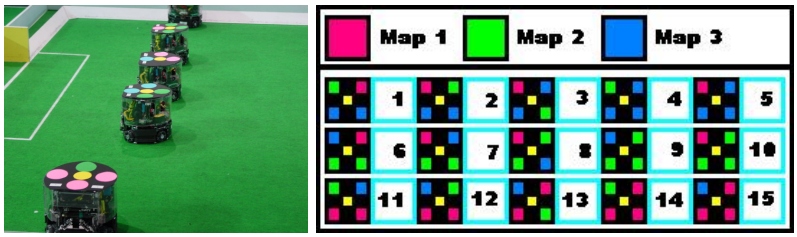


Abbildung 5.6: Das Modell des Teams *BigRed* der Cornell Universität. Es verwendet vier zusätzliche symmetrisch angeordnete Marker mit drei möglichen Farben und ermöglicht 15 verschiedene ID's zu unterscheiden.

CMU-Modell Dieses Modell wurde von der Carnegie Mellon Universität eingeführt [2] und wird von einigen Teams benutzt. Wie in Abbildung 5.7 zu sehen, werden vier radiale Marker mit zwei möglichen Farben verwendet, die zudem gegenüber Rotation eine eindeutige Anordnung aufweisen. Damit lassen sich $4^2 = 16$ eindeutige ID's darstellen. Die Berechnung der ID erfolg analog zur Cornell Kodierung per Zugriff auf eine Hashtabelle. Die Ausrichtung des Roboters kann hier anhand des maximalen Öffnungswinkels bestimmt werden, und so wiederum Orientierung und Position aus allen fünf Markern berechnet werden.

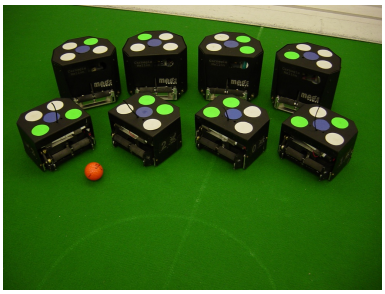


Abbildung 5.7: Das Modell des Teams der Carnegie Mellon University. Es verwendet vier zusätzliche asymmetrisch angeordnete Marker mit zwei möglichen Farben und ermöglicht 16 verschiedene ID's zu unterscheiden.

FU2004-Modell Dieses Modell verwende ich seit 2004 für unser Team die FU-Fighters. Es besteht aus drei radialen Markern mit zwei möglichen Farben, die wie im CMU-Modell eine zur Rotation eindeutige Anordnung besitzen. Damit lassen sich insgesamt 8 verschiedene ID's darstellen. Die ID wird analog den beiden Vorgängern per Hashtabelle ermittelt, sowie die Ausrichtung wiederum relativ zum maximalen Öffnungswinkel

bestimmt werden kann. Im Gegensatz zum Cornell-Modell müssen nur drei Farben kalibriert werden und im Kontrast zum CMU-Modell sind hier keine zwei Marker dicht bei einander, was bei starken Verzerrungen im Randbereich einen kleinen Vorteil verschafft.

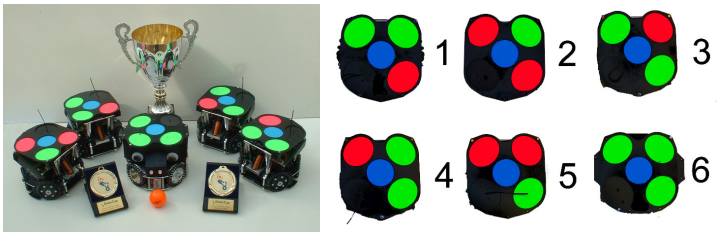


Abbildung 5.8: Das Modell der FU-Fighters seit 2004. Es verwendet drei zusätzliche asymmetrisch angeordnete Marker mit zwei möglichen Farben und ermöglicht 8 verschiedene ID's zu unterscheiden.

Zusätzlich habe ich diesem Modell weitere Robustheit verliehen. Dazu kombiniere ich nicht wie im Standardfall das Orientierungsmodell mit dem 1-Marker-Modell, sondern verwende alle Marker für das Tracking. Wie viele Marker zu welcher Farbkarte für einen bestimmten Roboter getrackt werden müssen, lässt sich dabei leicht anhand seiner ID ermitteln. Nun kann ähnlich dem 3-Marker-Modell die Redundanz der Kodierung ausgenutzt werden. So kann die Position eines nicht gefundenen Teammarkers aus zwei beliebigen radialen Markern des Roboters berechnet werden. In der Diagonalen ist die Position dabei eindeutig, ansonsten gibt es zwei mögliche Positionen mit korrektem Abstand, die wie üblich verifiziert werden können, siehe Abbildung 5.9. Die Abstandsberechnungen werden dabei wie immer in Weltkoordinaten vorgenommen.

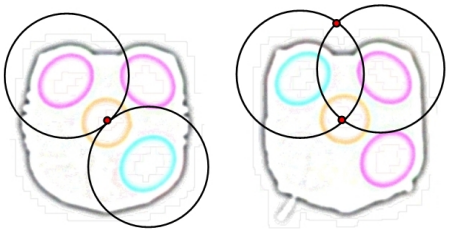


Abbildung 5.9: Redundanz des FU2004 Modells. Falls beim Tracking beide diagonalen Marker gefunden wurden, kann die Position des Teammarkers eindeutig bestimmt werden, ansonsten gibt es zwei symmetrische Möglichkeiten.

5dpo-Modell Dieses Modell wurde von dem Team 5dpo aus Portugal eingeführt, und wird in ähnlicher Form auch von anderen Mannschaften benutzt. Es verwendet im Gegensatz zu den bisherigen Modellen keine zusätzlichen Marker, sondern schwarze und weiße Flächen, die kreisförmig angeordnet sind. Nach einem radialen Scan, wie in Abbildung 5.10 rechts in rot zu sehen, kann anhand der Abtastkurve sowohl ID und Orientierung des Roboters bestimmt werden. In meiner Implementierung berechne ich diese Information allerdings bisher noch nicht, sondern verwende den Scan nur um die Existenz von

schwarzen und weißen Flächen zu verifizieren und so gegenüber dem Teammarker mehr Konfidenz zu erhalten.

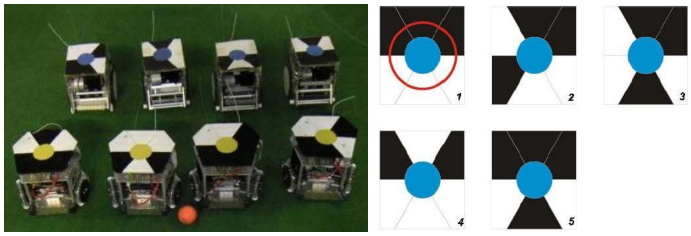


Abbildung 5.10: Das Modell des Teams 5dpo aus Portugal. Es verwendet radial angeordnete schwarz-weiße Flächen. Anhand der Abtastkurve kann ID und Orientierung bestimmt werden.

5.2.4 Entfernen gefundener Roboter aus dem Bild

Wie schon mehrfach erwähnt, werden Roboter, sobald sie gefunden wurden aus dem Bild entfernt. Dies hat den Vorteil, dass einerseits der Ball, der erst nach den Robotern gesucht wird, nicht auf einem Ball-ähnlichen Marker gefunden werden kann. Andererseits hilft es während der Robotersuche, das Überspringen von Markern auf andere Roboter oder das Mehrfachfinden einzelner Marker zu vermeiden.

Ich habe zwei verschiedene Möglichkeiten implementiert, einen Roboter aus dem Bild zu entfernen, wie in Abbildung 5.11 zu sehen. Einmal kann dieser als ganzes kreisförmig schwarz gemahlt werden. Diese Methode hat den Vorteil, dass sämtliche gefährlichen Farben auf dem Roboter entfernt werden können, ohne diese explizit im Modell verfolgen zu müssen. Der Nachteil ist, dass kaum Roboter vollkommen rund sind, sondern gerade vorne, wo der Ball geführt wird, eine Einbuchtung haben. Ein Kreis kann folglich auch Teile des Balls entfernen und so die Ballverdeckung erhöhen.

Die zweite Methode deckt nur die gefundenen Marker mit schwarzen Kreisen ab, folglich werden nur die notwendigen Bereiche entfernt. Allerdings sollte das Modell alle gefährlichen Marker erfassen.

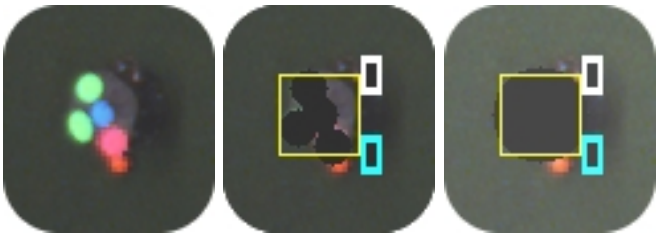


Abbildung 5.11: Man sieht zwei Möglichkeiten einen gefundenen Roboter (links) aus dem Bild zu entfernen. Entweder durch Entfernen des gesamten Roboters (rechts), oder der einzelnen Marker (Mitte).

Der Radius kann dabei in beiden Methoden vom Benutzer eingestellt werden. Um teure Wurzelberechnungen zu umgehen, wird der Kreis mit dem gewählten Radius einmal als Muster vorberechnet und kann dadurch mit billigen Operationen an der entsprechenden Stelle ins Bild kopiert werden.

Eine Schwierigkeit ist, dass Roboter aufgrund der Verzerrung weder als perfekte Kreise im Bild erscheinen, noch auf dem Feld überall die gleiche Größe im Bild aufweisen. Aus diesem Grund wird am Rand tendenziell ein wenig zuviel ausgeschnitten, während es in der Mitte gerade ausreicht. Um dies zu verbessern, könnten Muster zu mehreren Radien vorberechnet werden und der geeignete anhand der Position im Bild und der Transformationsfunktion ausgewählt werden. Um den Roboter korrekt auszustanzen, müsste der Kreis hingegen in Weltkoordinaten berechnet und in den Pixelraum transformiert werden. Dies ist allerdings rechenaufwendig und nicht notwendig, da schon die Methode mit festem Radius ausreichend gut funktioniert.

6 Kalibrierung und Mehrkamerasysteme

In diesem Kapitel beschreibe ich Methoden, mit denen eine Kamera über dem Spielfeld kalibriert werden kann. Zudem werden Systeme mit mehreren Kameras bei zunehmender Feldgröße und damit erhöhter Ballverdeckung immer wichtiger. Diese benötigen jedoch spezielle Lösungen, auf die ich in diesem Kapitel eingehen werde.

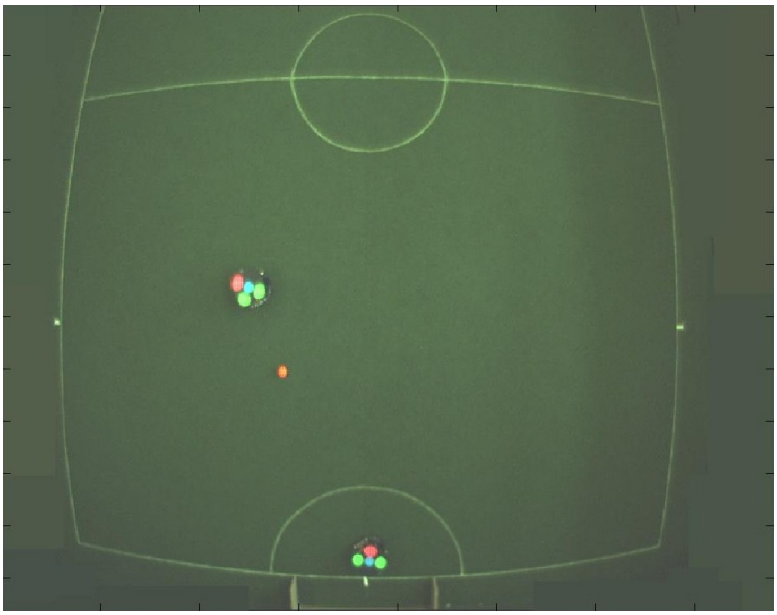


Abbildung 6.1: Tonnenförmige radial verzerrtes Bild des Spielfelds, aufgenommen mit einem 4.2mm Objektiv bei einer Höhe von 2.55 Metern. Die geraden Linien sind stark gekrümmt und Kreise erscheinen ellipsenförmig im Bild.

6.1 Kalibrieren einer Kamera über dem Spielfeld

Bis hierhin habe ich beschrieben, wie sich Ball und Roboter in einer Bildsequenz robust finden und verfolgen lassen. Allerdings müssen Position und Orientierung aller Objekte von Pixelkoordinaten in das Koordinatensystem des Spielfelds transformiert werden, bevor sie von der Steuerungssoftware verwendet werden können. Dieses hat seinen Ursprung in der Spielfeldmitte und die Distanzen entsprechen denen der Wirklichkeit. Zudem wird diese Transformation $T_{P \to F}$ (Pixel nach Feld), wie auch die entsprechende Umkehrfunktion $T_{F \to P}$ an mehreren Stellen des Suchalgorithmus benötigt, wie zuvor beschrieben.

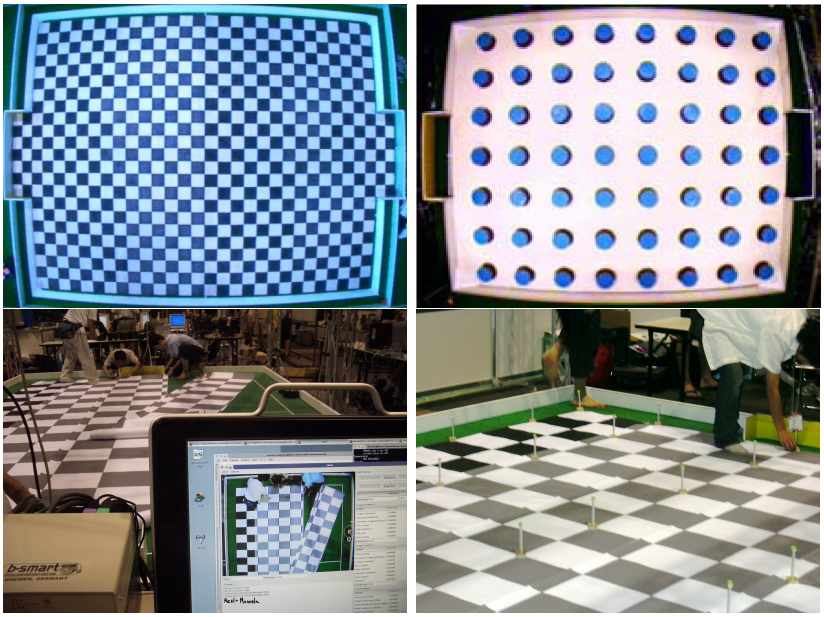


Abbildung 6.2: Aufwendige Methode zur Generierung von Kalibrierungspunkten. Dazu wird ein Teppich mit einer speziellen Musterung auf dem gesamten Feld ausgebreitet. Manche Teams verwenden sogar zusätzliche erhöhte Objekte, um auch die Ebene auf Roboterhöhe zu kalibrieren.

Wie Abbildung 6.1 zeigt, ist das Feld, wie es im Kamerabild zu sehen ist, stark verzerrt. Dafür gibt es zwei grundlegende Ursachen:

Aufhängung der Kamera Das Koordinatensystem der Kamera ist weder zentriert noch muss es achsenparallel zu dem des Spielfelds sein. Nicht nur, dass dies unmöglich zu garantieren ist, oftmals hängen Teams ihre Kameras absichtlich in einem schrägen Winkel weiter nach außen, um die Ballverdeckung vor den Toren zu minimieren.

Radiale Verzerrung der Linse Bei einer Deckenhöhe von oftmals nur 3m und einer Spielfeldfläche von knapp $22m^2$ ist man gezwungen Linsen mit weitem Öffnungswinkel und starker radialer Verzerrung zu verwenden. Wie in Abbildung 6.1 zu sehen, erscheinen gerade Linien als Kurven und Kreise als ellipsenartige Formen im Bild.

Eine weit verbreitete Entzerrungsfunktion ist die Methode von Tsai [19], die die Kamerafunktion durch einen Satz von intrinsischen und extrinsischen Parametern beschreibt. Allerdings hat dieses Verfahren zwei entscheidende Nachteile: Einerseits müssen die intrinsischen Parameter teilweise vorher bekannt sein, andererseits benötigt die Methode eine Vielzahl von Kalibrierungspunkten, die zudem das Feld möglichst gleichmäßig abdecken sollten. Um solche Kalibrierungspunkte zu generieren, verwendeten einige Teams in der Vergangenheit Teppiche mit spezieller Musterung. Um auch die Ebene auf Roboterhöhe zu kalibrieren, gingen manche sogar soweit, eine Vielzahl von Objekten auf

dem Feld zu positionieren, siehe Abbildung 6.2. Ein solches Verfahren ist offensichtlich äußerst zeitaufwendig – eine Kalibrierung dauerte bis zu 40 Minuten – und ab einer bestimmten Feldgröße nicht akzeptabel. Zudem passiert es öfters während eines Turniers, dass sich durch äußere Einflüsse die Aufhängung der Kamera verändert. Dennoch wird diese Methode immer noch von wenigen Teams verwendet.

Aus diesem Grund habe ich mich für Methoden entschieden, die in kurzer Zeit zu kalibrieren sind, und mit möglichst wenig Kalibrierungspunkten auskommen. Ein erster Ansatz verwendet dabei eine biquadratische Interpolation von 9 Punkten, um sowohl die radiale Verzerrung, als auch die projektive Transformation der Kamera zu approximieren. In einem verbesserten zweistufigen Verfahren wird zunächst getrennt einmal die radiale Verzerrungsfunktion der Linse bestimmt. Mithilfe dieser genügt es, während eines Turniers nur noch die projektive Transformation der Kamera zu ermitteln.

Im Folgenden werde ich auf beide Verfahren näher eingehen.

6.1.1 Ein erster Ansatz

In einem ersten Verfahren verwendete ich, wie in Abbildung 6.3 zu sehen, nur die 8 Randpunkte des Feldes, sowie die Feldmitte. Der Vorteil dabei ist einerseits, dass diese ohne zusätzlichen Aufwand eindeutig im Bild zu bestimmen sind. Andererseits sind sie daher für den Menschen im Gegensatz zu parametrisierten Transformationsfunktionen, dessen Parametern intuitiv keine Bedeutung zuzuordnen ist, leicht einstellbar. Um nun einen Punkt in das Feldkoordinatensystem zu transformieren, werden diese 9 Punkte wie folgt hier am Beispiel der x-Koordinate biquadratisch interpoliert:

Vorberechnung Wir betrachten die 9 Kalibrierungspunkte als ein Gitter bestehend aus drei Punktereihen mit je drei Punkten. Für jede der drei Punktreihen werden nun zwei quadratische Kurven vorberechnet:

- $X_{Pixel} \rightarrow Y_{Pixel}$, als waagerechte rote Kurven im Bild markert.
- $X_{Pixel} \rightarrow X_{Feld}$.

Transformation eines Punktes P Anhand der x-Koordinate von P werden mittels der beiden vorberechneten Kurven für jede der drei Punktereihen sowohl die zugehörigen Y_{Pixel} und X_{Feld} berechnet. Durch diese drei Punkte der Form (Y_{Pixel}, X_{Feld}) wird nun wieder eine quadratische Funktion gelegt, anhand der mithilfe der Y-Koordinate von P, der transformierte x-Wert abgelesen werden kann.

Für den y-Wert verläuft die Berechnung analog. Die Methode, wie bisher beschrieben, geht allerdings davon aus, dass sich der zu transformierende Punkt in der Feldebene befindet. Um nun die Position eines Roboters zu transformieren, verwende ich das simple Prinzip des Strahlensatzes, wie in Abbildung 6.3 zu sehen. Dabei wird zunächst die transformierte Position P_0 für einen Roboter der Höhe Null berechnet. Nun kann mithilfe des Abstands D zum Lotpunkt der Kamera L_C, d.h. $D = ||P_0 - L_C||$, der korrekte Abstand

$$d = D \cdot \frac{H - h}{H}$$

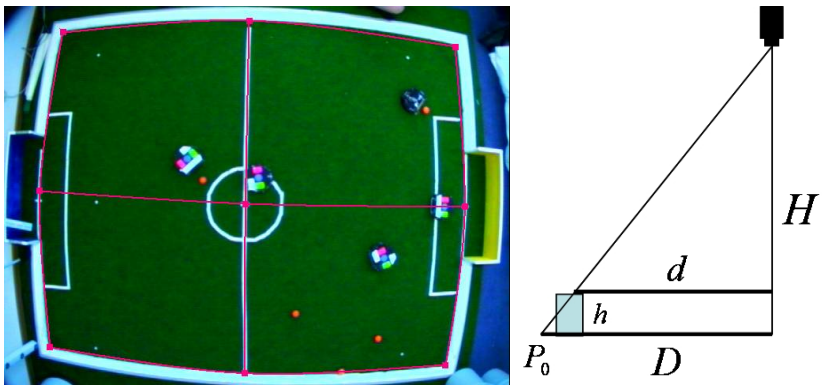


Abbildung 6.3: Transformation mittels biquadratischer Interpolation von 9 Punkten (links). Durch jeweils drei Punkte wird eine Quadratische Funktion gelegt. Rechts sieht man, wie mithilfe des Strahlensatzes die Höhe der Roboter in der Transformation berücksichtigt werden kann.

berechnet werden, wobei H die Höhe der Kamera und h die des Roboters ist. Es ist folglich notwendig sowohl die Höhe als auch den Lotpunkt der Kamera zu kennen. Dies kann einerseits per Hand gemessen werden, andererseits lässt sich anhand der projektiven Transformation die 3D Kameraposition und damit Lotpunkt und Höhe automatisch berechnen. Dazu ist es allerdings notwendig, die radiale Verzerrung und die projektive Transformation getrennt zu behandeln, wie es in einer verbesserten Methode im nächsten Abschnitt beschrieben wird.

Die Transformation mittels biquadratischer Interpolation wurde mehrfach erfolgreich auf Turnieren verwendet, bei zunehmender Feldgröße und geringer Deckenhöhe konnte sie allerdings die benötigten stark verzerrenden Linsen nicht mehr ausreichend gut modellieren.

6.1.2 Kalibrieren in zwei Schritten

In diesem Verfahren wird zunächst für das verwendete Objektiv die Verzerrungs- bzw. Entzerrungsfunktion der Linse bestimmt. Der Vorteil dabei ist, dass dies nur einmal im Vorhinein ermittelt werden muss. Während der Kalibrierung auf einem Turnier genügt es, die projektive Transformation, d.h. Translation und Rotation der Kamera zum Feld, zu bestimmen. Allerdings verändern sich bei Objektiven mit eingebautem Zoom je nach Zoomfaktor die optischen Eigenschaften des Systems und damit die Verzerrung. Die FU-Fighters verwenden 4.2mm bzw. 6mm Objektive ohne eingebauten Zoom. Das Verfahren wurde schon in einem internen Report detailliert beschrieben [17], daher werde ich es hier nur grob skizzieren.

6.1.2.1 Eliminieren der radialen Verzerrung

Die radiale Verzerrung z.B. einer sphärischen Linse ist in alle Richtungen symmetrisch und nimmt zum Rand hin zu, während sie im Zentrum der Linse gleich Null ist. Um

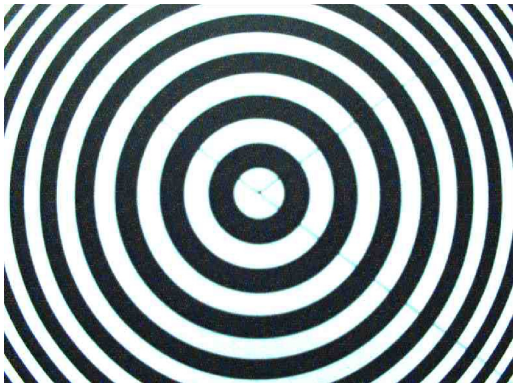


Abbildung 6.4: Verwendetes Muster, anhand dessen die radiale Verzerrung des Objektivs in einer Vorstufe einmalig ermittelt werden kann.

nun ein Bild radial zu entzerren muss dieses an jedem Pixel in Abhängigkeit zu seinem Abstand zum Mittelpunkt gestreckt (*Barrel distortion*) bzw. gestaucht werden (*pin-cushion distortion*). Wir modellieren sowohl Verzerrungs- (f^+), als auch Entzerrungsfunktion (f^-) mithilfe eines Polynoms dritten Grades, d.h.

$$f^+(d) = a_3 d^3 + a_2 + d^2 + a_1 d$$

wobei d dem Abstand zum Mittelpunkt entspricht und der konstante Term a_0 entfällt, da im Zentrum der Linse keine Verzerrung vorhanden ist. Mithilfe nichtlinearer Regression können nun aus einer Menge von Punkten der Form (d^-, d^+), d.h. entzerrter und verzerrter Abstand, die Koeffizienten von f^+ bestimmt werden. Entsprechend verfährt man für f^-.

Zur Generierung der Kalibrierungspunkte verwenden wir ein Muster bestehend aus konzentrischen Kreisen, dessen Abstand zueinander bekannt ist (siehe Abbildung 6.4). Dabei werden strahlenförmig vom Zentrum ausgehend alle Übergänge gefunden. Allerdings muss die Kamera möglichst perfekt mittig und senkrecht auf das Muster gerichtet sein.

In Abbildung 6.5 sehen wir die ermittelte Verzerrungs- und Entzerrungsfunktion für ein 4.2mm Objektiv.

6.1.2.2 Projektive Transformation

Im Folgenden gehen wir davon aus, dass die radiale Verzerrung im Bild bereits eliminiert wurde und nur noch die projektive Transformation bestimmt werden muss. Der Vorteil dieses Verfahrens ist, dass diese Abbildung von bereits radial entzerrten Koordinaten in Spielfeldkoordinaten analytisch berechnet und aus dieser die genaue Rotation und Translation der Kamera bestimmt werden kann.

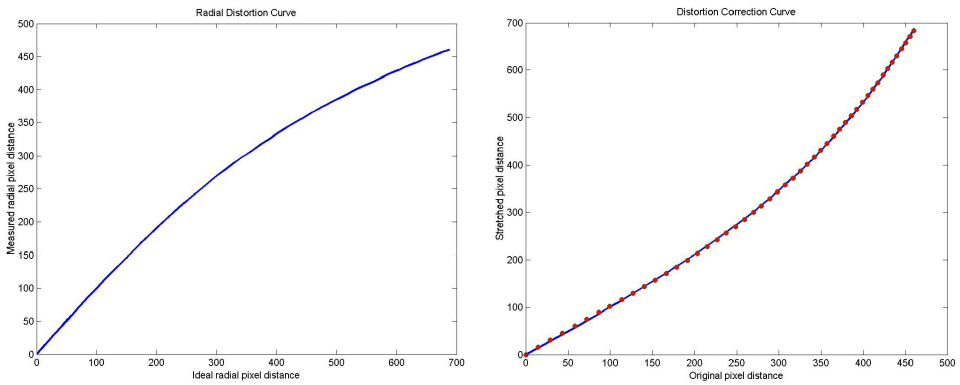


Abbildung 6.5: Anhand der Kalibrierungspunkte aus dem Muster kann die Verzerrungsfunktion (links) wie auch die Entzerrungsfunktion als Polynom dritten Grades mithilfe nichtlinearer Regression bestimmt werden.

In [17] wird die Umkehrabbildung, d.h. die eines Punktes (x, y) auf der Spielfeldebene in die Projektionsebene der Kamera (x'', y''), wie folgt beschrieben:

$$\begin{pmatrix} x'' \\ y'' \\ 1 \end{pmatrix} = f_1(H(x, y, 1)^t) = f_1 \left[\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \right]$$

wobei $f_1((x, y, z)^t) = (x/z, y/z, 1)^t$ Punkte perspektivisch in die Kameraebene bei $z = 1$ projiziert. Die Transformationsmatrix H transformiert einen Punkt $(x, y, 1)$ in homogenen Koordinaten des Spielfelds in das Koordinatensystem der Kamera. Es gilt

$$H = \lambda \cdot \begin{pmatrix} r_1 & r_2 & -Rt \end{pmatrix}$$

wobei r_1, r_2 den beiden ersten Spalten der Rotationsmatrix R und t dem Translationsvektor entsprechen. λ ist ein Skalierungsfaktor, da jedes Vielfache der eigentlichen Transformationsmatrix die obere Gleichung ebenso erfüllt.

Um von Kamerakoordinaten in Feldkoordinaten zu transformieren, kann einfach die inverse Matrix $G = H^{-1}$ verwendet werden, so dass die Gleichung

$$(x, y, 1)^t = G(x'', y'', 1)^t$$

erfüllt wird.

Zur Bestimmung der Koeffizienten $h_{11}, \ldots, h_{33}$ der Matrix H genügen vier Punkte in Feldkoordinaten, die auf vier Punkte im entzerrten Kamerakoordinatensystem abgebildet werden sollen. Wir verwenden dazu die Eckpunkte des Spielfelds bzw. die einer Spielfeldhälfte, falls das System mit zwei Kameras betrieben wird.

Anhand der Matrix H kann die Rotation R und die Translation t extrahiert werden. Dazu wird zunächst der Skalierungsfaktor $\lambda = |h_1|/|r_1|$ und damit die korrigierte Transformationsmatrix $H' = H/\lambda$ berechnet, wobei h_1 der ersten Spalte von H entspricht.

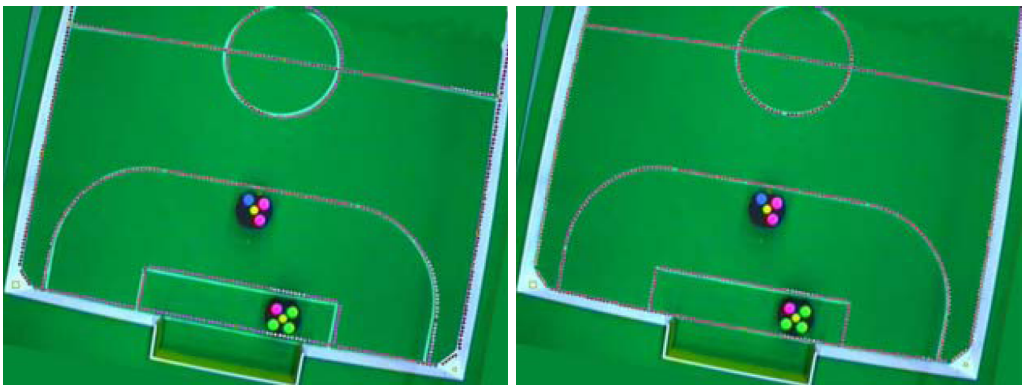


Abbildung 6.6: Man sieht links die Transformation nach dem Initialisierungsschritt, rechts nach der Optimierung.

Aus H' lässt sich nun $r_1 = h'_1$ und $r_2 = h'_2$ ablesen und daraus $r_3 = r_1 \times r_2$ berechnen. Damit ergibt sich für die Translation $t = -R^{-1}h'_3$.

Das genaue Verfahren wird im Detail unter [17] beschrieben.

6.1.2.3 Diskussion

Wie gerade beschrieben, genügt es nach Entfernen der radialen Verzerrung, eine projektive Transformation zu bestimmen, die ausschließlich die Aufhängung der Kamera modelliert. Allerdings kann die radiale Entzerrungsfunktion ebenso mit der biquadratischen Transformationsmethode kombiniert werden. Dadurch lassen sich bessere Ergebnisse erzielen, falls die radiale Verzerrung nicht perfekt bestimmt wurde. Wie unter 6.1.2.1 erläutert, benötigt das momentan von uns eingesetzte Verfahren der konzentrischen Kreise eine zentrierte und senkrecht ausgerichtete Kamera. Dies ist allerdings schwer zu garantieren. Zudem muss eine Linse nicht unbedingt zentriert auf dem Chip der Kamera angebracht sein.

6.1.3 Automatische Kalibrierung

Zusätzlich habe ich eine Methode entworfen, die die Parameter der Transformation automatisch findet. Dazu werden zunächst die Feldlinien als Punktemenge im Bild gefunden. Nun können diese Punkte mit der aktuellen Transformationsmethode in den Feldkoordinatenraum transformiert und mit einem Modell des Spielfelds verglichen werden. Dazu wird für die verwendete Transformation eine Qualität berechnet, wie gut die Feldlinien im Bild unter der gegebenen Transformation zum Modell passen. Anhand dieser Qualitätsfunktion können die Parameter der Transformationsfunktion mit Optimierungsverfahren, wie Gradientenabstieg, optimiert werden. Dazu wird in einem ersten Schritt die Transformation anhand einer bestimmten Feldregion geeignet initialisiert. Abbildung 6.6 zeigt die Transformation vor und nach dem Optimierungsschritt.

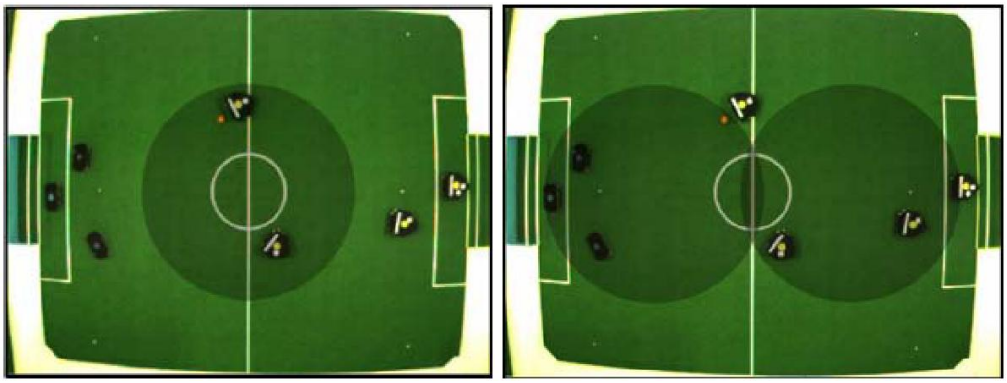


Abbildung 6.7: Ballüberdeckung mit einer (links) oder zwei (rechts) Kameras für das 2003 Spielfeld. In den dunkelgrünen Bereichen ist der Ball stets sichtbar, auch wenn ein Roboter mit voller Höhe von 15cm in Richtung der Kamera vor diesem steht. 2004 hat sich die Spielfeldfläche zudem verdoppelt, dementsprechend ist auch die Ballüberdeckung erheblich gestiegen. Bilder aus [4]

Das Verfahren wurde veröffentlicht und ist ausführlich beschrieben unter [5]. Diese Methode, kombiniert mit den vorher beschriebenen Verfahren, ermöglicht eine Kamerakalibrierung per Knopfdruck.

6.2 Mehrkamerasysteme

Seit Beginn von RoboCup hat sich die Fläche des Spielfelds mehr als verfünffacht, so dass Objekte im Bild, wie Farbmarker oder der Ball, wesentlich kleiner erscheinen. Zudem sind Objektive mit großem Öffnungswinkel und starker Verzerrung notwendig, so dass das Problem der Ballverdeckung immens gestiegen ist: Wie in Abbildung 6.7 zu sehen, kann dieser annähernd überall auf dem Feld durch einen Roboter verdeckt werden. Aus diesen Gründen verwenden inzwischen sämtliche Teams Systeme mit zwei oder teilweise mehr Kameras.

Das System, wie bisher beschrieben, verwendet eine Kamera und wurde auf mehreren Turnieren erfolgreich eingesetzt. Bei Verwendung von zwei Kameras richte ich jeweils eine auf je eine Spielfeldhälfte, wobei sich diese partiellen Ansichten im Übergangsbereich überschneiden (siehe Abbildung 6.8), so dass sich ein Roboter zu jeder Zeit vollständig in mindestens einer der Ansichten befindet. Die Bildaufnahme der Kameras wird dabei entweder durch einen Kamera-internen oder einen externen Trigger synchronisiert.

Die Suche nach Ball und Robotern in solch einer partiellen Ansicht funktioniert grundsätzlich genauso, wie in den vorherigen Kapiteln beschrieben, allerdings können im Gegensatz zu einer globalen Sicht Objekte die partielle Ansicht verlassen und sollten daher nicht gesucht werden. Aus den Ergebnissen der einzelnen Ansichten muss ein globales Weltbild erstellt werden. Dabei müssen die Roboter im Übergangsbereich von einer zur anderen Sicht geeignet interpoliert werden, um einen glatten Übergang zu gewährleisten. Das globale Weltbild besteht aus

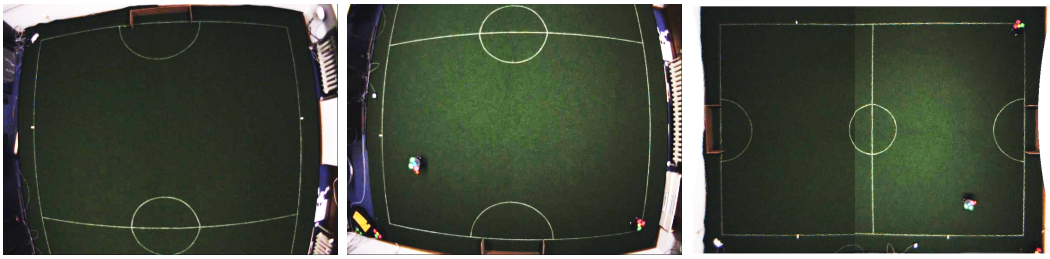


Abbildung 6.8: Die beiden linken Bilder zeigen die lokalen Sichten zweier Kameras über dem Spielfeld. Die Sichten überlappen in der Mitte, so dass ein Roboter immer in mindestens einer Sicht vollständig zu sehen ist. Das rechte Bild zeigt, wie in der 3D Ansicht der Vision-Software die Texturen beider Kameras in Echtzeit mithilfe der Transformationsfunktion in das Spielfeldkoordinatensystem gelegt werden. Dies dient lediglich der Visualisierung.

- Position, Orientierung und Geschwindigkeit aller Roboter in Weltkoordinaten.
- Position und Geschwindigkeit aller zur Verfolgung verwendeten Marker aller Roboter in Weltkoordinaten. Diese werden dazu benutzt, bei Eintritt in die andere Ansicht diese zu reinitialisieren, um eine globale Suche zu vermeiden.
- Position und Geschwindigkeit des Balls in Weltkoordinaten als auch zusätzliche Informationen, wie die Flugkurve mit Auftreffpunkt und Auftreffzeit im Falle eines Hochschusses.

Der grobe Aufbau des Algorithmus mit zwei Kameras sieht wie folgt aus:

- Bestimme anhand des globalen Weltbilds, welches Objekt in welcher partiellen Ansicht sichtbar sein müsste.
- Suche die entsprechenden Objekte in den partiellen Sichten mit den gewohnten Methoden. Da die Kameras synchronisiert sind, liefern sie ihre Bilder gleichzeitig an den Rechner. Dieser bearbeitet die Bilder in zwei getrennten Threads.
- Ist die Suche in beiden Ansichten abgeschlossen, so wird aus den gefundenen Objekten in beiden Sichten das neue Weltbild fusioniert.

Im Folgenden werde ich auf die einzelnen Schritte genauer eingehen.

6.2.1 Suchen in partiellen Ansichten

Wie vorher die globale Sicht hat nun jede partielle Sicht unter anderem ihre eigene Transformationsfunktion, Modelle von Ball und Robotern und eine Menge von Farbkarten, die während der Suche verwendet werden. Um dem Benutzer die doppelte Arbeit zu ersparen, werden dabei allgemeine Einstellung automatisch konsistent gehalten, z.B. welches Robotermodell für das eigene oder gegnerische Team verwendet werden soll oder mit welchen Farbkarten diese arbeiten sollen. Spezifische Einstellungen hingegen, wie z.B. Parameter der Segmentierung, sind für beide Sichten getrennt einstellbar.

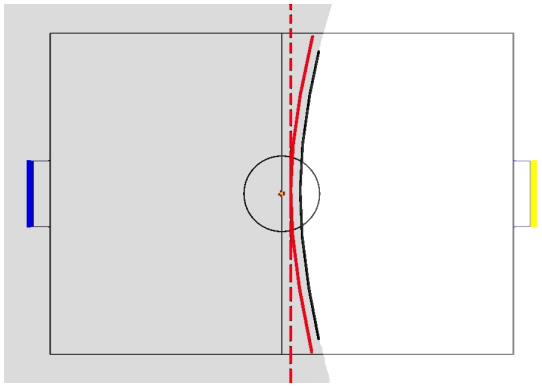


Abbildung 6.9: Die schwarze Kurve begrenzt den eingegrauten Bereich, in dem ein Roboter mit maximaler Höhe und Durchmesser gerade noch vollständig in der linken Kamera zu sehen ist. Die rote Kurve markiert diese Grenze für Ball. Anhand der gestrichelten Tangente einer Sichtbarkeitsgrenze kann eine schnelle Vorentscheidung getroffen werden.

Die Suche nach Ball und Robotern in solch einer partiellen Ansicht funktioniert grundsätzlich genauso, wie zuvor für eine globale Sicht mit einer Kamera beschrieben. Dazu bestimme ich für alle Objekte, in welchen partiellen Ansichten sie vollständig sichtbar sind, und markiere sie für die Suche. Objekte, die zuvor auch nicht im globalen Weltbild, d.h. in keiner der beiden partiellen Sichten, gefunden wurden, werden in beiden gesucht.

Welches Objekts ist in welcher Ansicht vollständig sichtbar? Die Grenze, bis zu der ein Roboter noch vollständig im Kamerabild zu sehen ist, ist wie in Abbildung 6.9 dargestellt, eine Kurve, welche sich automatisch anhand der Transformationsfunktion berechnen lässt. Dazu transformiere ich den Rand des Kamerabildes, der an die Feldmitte grenzt, für einige Punkte mithilfe der Transformationsfunktion in Weltkoordinaten für die maximale Höhe eines Roboters, d.h. 15cm. Im Beispiel aus Abbildung 6.8 ist dies also links der untere Bildrand und rechts der obere Bildrand. Die entstehende Kurve markiert die Grenze, bis zu der ein Roboter maximaler Höhe bis zu seinem Mittelpunkt noch im Bild ist. Daher wird diese noch um den maximalen Radius eines Roboters, d.h. 9cm, verschoben.

Für den Ball verfahre ich aus folgendem Grund nicht analog: Da Roboter, die nur zum Teil in einer Sicht zu sehen sind, nicht gesucht und damit mögliche ballähnliche Marker auf ihnen nicht aus dem Bild entfernt werden können, möchte ich den Ball in diesen Bereichen gar nicht erst suchen. Dazu verschiebe ich die schwarze Sichtbarkeitsgrenze erneut um den maximalen Radius eines Roboters, da ein Roboter außerhalb dieser Grenze nicht mehr als um seinen Radius über diese hinausragen kann.

Um zu berechnen, ob ein Objekt in einer Ansicht sichtbar ist, muss nur getestet werden, ob seine aktuelle Position im globalen Weltbild auf der richtigen Seite der zugehörigen Sichtbarkeitsgrenze liegt. Da ich diese Kurve mit Strecken approximiere, entspricht dies einer Anzahl von left-of-Tests. Um dies zu beschleunigen, kann anhand der Tangente eine Vorentscheidung getroffen werden.

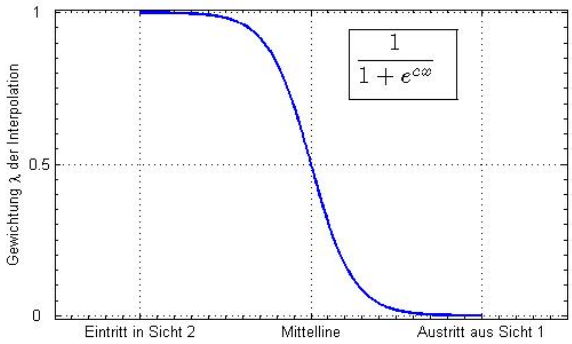


Abbildung 6.10: Gewichtung λ der Interpolation.

Wechsel von einer in die andere Sicht Da sich beide Sichten im Übergangsbereich überlappen, ist ein Objekt bei Eintritt in eine partielle Ansicht in der anderen sichtbar und wurde gefunden. Um eine teure globale Suche für das neue Objekt zu vermeiden, verwende ich das globale Weltbild, um die lokale Suche zu initialisieren. Dazu transformiere ich z.B. Position und Orientierung des eintretenden Roboters, sowie die Position der von seinem Modell verfolgten Marker in Pixelkoordinaten der betroffenen Ansicht und setzte die entsprechenden Felder des Modells.

6.2.2 Erstellen eines globalen Weltbilds

Nachdem in beiden partiellen Ansichten die Suche abgeschlossen ist, müssen die Daten aus beiden Sichten geeignet zu einem globalen Weltbild fusioniert werden. Dieses wird einerseits von der Steuerungssoftware, andererseits von der Suche selbst verwendet, wie im vorherigen Abschnitt beschrieben. Dazu werden zunächst anhand der Position Paarungen von Robotern ermittelt, die in beiden Sichten gefunden wurden, und darauf alle Roboter wiederum anhand der Position dem globalen Weltmodell zugeordnet. Falls das Robotermodell mit einem Identifizierungsmodell kombiniert ist, wird diese Zuordnung zusätzlich anhand der gefundenen Identitäten der Roboter vorgenommen. Dabei müssen zunächst mögliche Vertauschungen korrigiert werden, an denen Roboter aus beiden Sichten beteiligt sind, und die daher nicht schon in den einzelnen Sichten behoben werden konnten. Ich korrigiere diese Vertauschungen auf die gleiche Weise, wie schon unter 5.2.3.2 beschrieben.

Roboter im Übergangsbereich sind in beiden Sichten zu sehen. Die fusionierte Position und Orientierung von Robotern bzw. Ball im Weltmodell berechne ich daher als gewichtetes Mittel der einzelnen Sichten.

$$Pos = \lambda \cdot Pos_1 + (1 - \lambda) \cdot Pos_2$$

Dies erhöht nicht nur die Genauigkeit in diesem Bereich, sondern ermöglicht es auch Unstetigkeiten beim Eintritt und Austritt aus dem Übergangsbereich zu vermeiden. Diese wären zum Beispiel für die Steuerungssoftware oder die Ball- und Robotervorher-

sage (siehe Kapitel 7) problematisch. Die Gewichtung λ berechne ich, wie in Abbildung 6.10 zu sehen, mithilfe einer Sigmoidfunktion, da diese in den kritischen Bereichen eine sehr flache Steigung aufweist.

6.2.3 Pixelmaskierung im Übergangsbereich

Wie in den vorherigen Abschnitten beschrieben, können Roboter, die nur teilweise im Bild zu sehen sind, nicht gefunden, und ihre Marker daher auch nicht aus dem Bild entfernt werden. Aus diesem Grund wurde mithilfe einer Sichtbarkeitsgrenze ein Ball in diesem kritischen Bereich nicht als sichtbar in dieser Ansicht erklärt. Allerdings könnten, wenn der Ball in der Ansicht sichtbar ist, jederzeit während der Suche falsche Hypothesen auf solchen nicht entfernten ballähnlichen Markern gefunden werden. Um dies zu verhindern, verwende ich die zu solchen Zwecken eingeführten Pixelmasken (siehe 4.4.4). Dabei wird automatisch jedes Pixel, dessen Weltkoordinaten auf der falschen Seite der Sichtbarkeitsgrenze des Balls liegen in der Pixelmaske des Balls verboten.

6.3 *Client-Server*-Architektur

In de FU-Fighters System wird das Vision-System zusammen mit allen anderen Komponenten, wie Steuerungs- und Kommunikationssoftware in eine Applikation gelinkt. Dies hat den Vorteil, dass das Gesamtsystem an einem Rechner leicht von einer Person bedient werden kann. Viele andere Teams verwenden hingegen auf Grund hoher Rechenlast mehrere Rechner – teilweise sogar zwei für das Vision-System (einen Rechner pro Kamera) und einen für die Steuerungssoftware.

Um das Vision-System anderen leicht zugänglich zu machen, habe ich die Funktionalität eingebaut, dieses auch als eigenständige Applikation zu verwenden. Dabei werden die von der Steuerungssoftware benötigten Daten mittels UDP über Netzwerk an die gewünschten Rechner gesendet.

7 Messen und Eliminieren der Latenzzeit des Systems

Wie jedes rückgekoppelte System hat auch dieses eine bestimmte Latenzzeit. Dabei tragen nicht nur das Vision-System, sondern sämtliche Komponenten des Gesamtsystems, wie in Abbildung 1.3 dargestellt, zur Verzögerungszeit bei: Während der Bilderfassung muss der CCD Chip für eine bestimmte Zeit das einfallende Licht integrieren. Das Bild muss an den Rechner übertragen und in den Hauptspeicher gelegt werden. In diesem werden nun vom Vision-System alle Objekte gefunden und an die Steuerungssoftware übertragen. Die Steuerungssoftware berechnet anhand dieser Daten Kommandos, die mittels drahtloser Übertragung an die Roboter übermittelt werden und dort alle 4ms ausgewertet werden. Schließlich wirkt noch die Trägheit der Mechanik selbst der Umsetzung des gesendeten Befehls entgegen.

Insgesamt summieren sich diese einzelnen Latenzen in unserem System auf 120ms - 150ms. Dies stellt für die Steuerungssoftware ein signifikantes Problem dar: Zwar bleiben langsam fahrende Roboter kontrollierbar, allerdings erreichen unsere Roboter derzeit Maximalgeschwindigkeiten um die $3m/s$, so dass während einer Totzeit des System von 130ms mehr als 30cm zurückgelegt werden können. Dies hat zur Folge, dass ein Roboter mehr als 30cm von seinem Weg abkommen kann, bevor es vom System korrigiert wird, oder er um seinen gewünschten Haltepunkt oszilliert, anstelle zum Stillstand zu kommen. Ich löse dieses Problem mit folgendem Ansatz: Anstelle die gefunden Positionen und Orientierungen von Robotern und Ball direkt in der Steuerungssoftware zu verwenden, werden diese zunächst um die Latenzzeit des Gesamtsystems in die Zukunft vorhergesagt. Auf diese Weise berechnet die Steuerungssoftware die Roboterkommandos anhand der Positionen und Orientierungen, die sie einnehmen, wenn sie auf die aktuell berechneten Kommandos reagieren können.

In den nächsten Abschnitten beschreibe ich kurz, wie man die Latenzzeit eines solchen Systems messen kann und wie ich mithilfe eines neuronalen Netzes die Vorhersage der Roboter automatisch erlerne. Die Arbeit hierzu wurde in einer Veröffentlichung ausführlich beschrieben [1], daher werde ich in dieser Arbeit nicht auf sämtliche Details eingehen.

7.1 Messen der Latenzzeit des Systems

Die Latenzzeiten jeder einzelnen Komponente direkt zu bestimmen ist kompliziert und mit hohem Aufwand verbunden, allerdings lässt sich die Latenzzeit des Gesamtsystems auf einfache Weise experimentell messen: Dazu lasse ich den Roboter in der Steuerungssoftware auf einer geraden Linie abwechselnd vor- und zurückfahren. Die gesendete Geschwindigkeit entspricht dabei einer Sinuskurve, wie in Abbildung 7.1 zu sehen, so

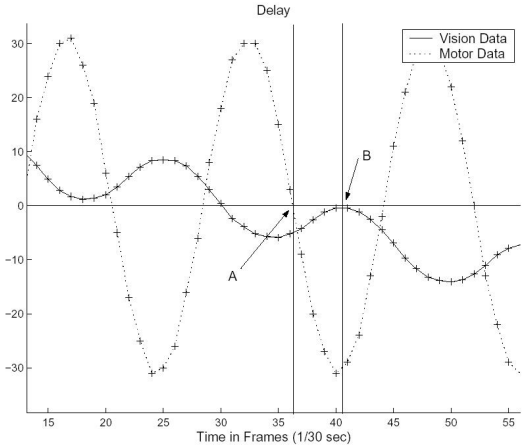


Abbildung 7.1: Zu sehen sind zwei Kurven. Die gestrichelte Linie entspricht der gesendeten Sinuskurve, die durchgezogene Linie der Position entlang der Bewegungsrichtung des Roboters, wie sie von der Vision beobachtet wurde. Die Änderung der gesendeten Fahrrichtung (A) entspricht der Richtungsänderung (B). Der zeitliche Unterschied beträgt um die 130ms. Bild entnommen aus [1].

dass der Roboter bis zur Mitte des Weges beschleunigt, den Rest der Strecke abbremst und an den Endpunkten die Richtung wechselt. Eine Richtungsänderung entspricht also einer Nullstelle der Sinuskurve. Betrachtet man nun gleichzeitig die von der Vision gesehene Position des Roboters, so entspricht eine Richtungsänderung einem lokalen Maximum bzw. Minimum. Folglich lässt sich die Latenzzeit als Verschiebung zwischen einer Nullstelle der Sinuskurve und dem darauf folgenden Minimum bzw. Maximum in der Position ablesen. In Abbildung 7.1 entspricht dies gerade 120ms.

7.2 Eliminieren der Latenzzeit - Vorhersage aller Objekte

Um die Latenzzeit zu eliminieren, berechne ich die Position und Orientierung aller Objekte um genau diese Zeit in die Zukunft. Auf die Vorhersage von Ball und gegnerischen Robotern möchte ich in dieser Arbeit nicht weiter eingehen, sie lassen sich z.B. anhand der zuletzt gesehenen Daten der Vision linear vorhersagen.

Für die eigenen Roboter stehen zusätzlich noch die zuletzt gesendeten Steuerungssignale zur Verfügung, die der Vorhersage als Eingabe dienen können. Auf diese Weise kann z.B. für einen stehenden Roboter, dem ein Rotationskommando geschickt wurde, seine Drehung bereits vorausberechnet werden, bevor die Vision eine Bewegung des Roboters wahrgenommen hat.

Der grobe Aufbau des Verfahrens unterteilt sich in folgende Schritte:

- Zunächst werden Daten gesammelt, während der Roboter mehrere Minuten lang über das Feld fährt. Dabei muss darauf geachtet werden, dass der Raum der Steuerungssignale möglichst gut abgedeckt ist und keine unnatürlichen Daten enthalten sind, wie z.B. Kollision mit Wänden oder anderen Robotern.

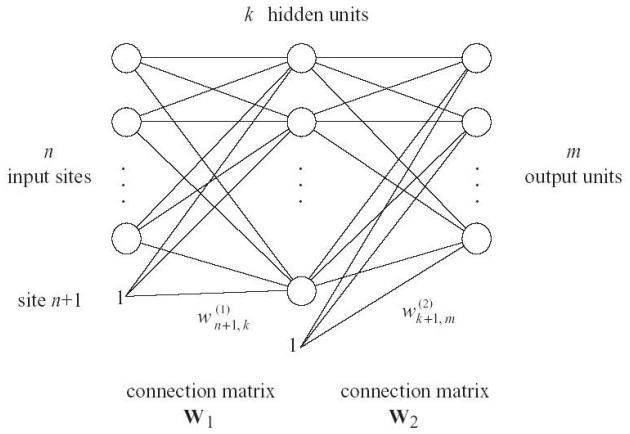


Abbildung 7.2: Darstellung eines dreischichtigen neuronalen Netzes. Bild entnommen aus [1].

- Diese Daten werden für den Lernprozess geeignet aufbereitet und in eine Trainings- und Testmenge zerlegt.
- Schließlich wird ein neuronales Netz mit diesen Daten trainiert.

Aufbereitung der Daten Bei einer Framerate von 30Hz verwende ich als Eingabe des Neuronalen Netzes Daten der letzten 6 Frames und als Ziel das vierte Frame in der Zukunft. Dies entspricht einer Latenzzeit von 133ms. Ein Eingabevektor enthält nun die Translationen und Rotationen bezüglich der letzen 6 Frames, sowie die letzten 6 Steuerungssignale.

Da für die Vorhersage des Roboters nicht die absolute Position auf dem Feld von Bedeutung ist, sondern nur sein relativer Bewegungszustand, kodiere ich seinen Zustand relativ zu seiner Position und Orientierung im aktuellen Frame. Auf diese Weise wird die Komplexität der Eingabedaten für das Netz verringert. Dazu transformiere ich die Translation in das lokale Koordinatensystem des Roboters. Steuersignale und Rotation befinden sich bereits in lokalen Koordinaten, allerdings gilt es bei der Kodierung der Rotation, Unstetigkeiten zu vermeiden, da diese von dem Neuronalen Netz schwierig zu approximieren sind. Daher wird der Winkel nicht mit einem Wert kodiert, da $\pi/2$ und $-\pi/2$ denselben Winkel beschreiben, sondern als sein Sinus und Kosinus. Damit ergeben sich $7 \times 6 = 42$ Eingabewerte.

Der zu lernende Zielvektor besteht analog aus Translation und Rotation vom aktuellen Frame zum vierten Frame in der Zukunft. Die gesammelte Menge solcher Datensätze wird zu gleichen Teilen in eine Trainings- und eine Testmenge unterteilt.

Trainieren des neuronalen Netzes Als Approximator verwende ich ein drei Schichten Feed-Forward Netz mit 42 Eingangsneuronen, 10 versteckten Neuronen in der mittleren Schicht und 4 Ausgabeneuronen. Als Transferfunktion dient die Sigmoide, siehe Abbildung 6.10. Während des Lernprozesses werden dem Netz Daten der Trainingsmenge prä-

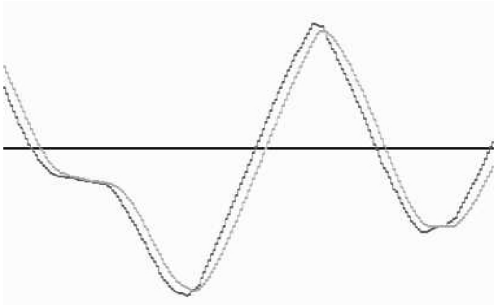


Abbildung 7.3: Die x-Koordinate der Position des Roboters aus der Vision (grau) und die Vorhersage für vier Frames des neuronalen Netzes (schwarz). Bild entnommen aus [1].

sentiert und mithilfe des standardisierten batch-Backpropagation-Algorithmus gelernt. Die Lernrate wird dabei abhängig von der Anzahl an Datensätzen bestimmt.

Eine umfangreiche Beschreibung neuronaler Netze ist unter [16] zu finden.

Ergebnisse Der Vorteil eines lernenden Systems ist seine Flexibilität. So muss bei einem neuen Roboter oder wenn sich der PID-Regler auf einem bekannten Roboter ändert, kein neues physikalisches Modell aufwendig bestimmt werden, sondern kann in relativ kurzer Zeit erlernt werden.

Die Vorhersage mittels neuronalem Netz wurde ausgiebig auf mehreren Wettbewerben und für verschiedenste Roboter getestet und eliminiert die negativen Effekte der Latenzzeit fast gänzlich. Abbildung 7.3 zeigt die Kurve der Position in der x-Koordinate, wie sie die Vision beobachtet hat (grau), und die Vorhersage für vier Frames des neuronalen Netzes (schwarz). Man sieht, dass sich beide Kurven beinahe gänzlich gleichen und um vier Frames verschoben sind.

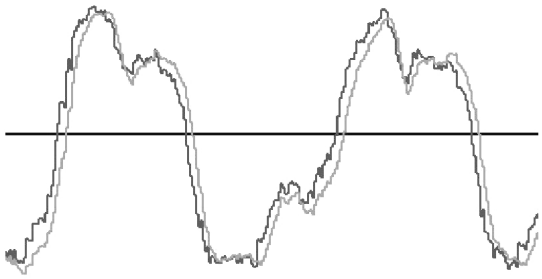


Abbildung 7.4: Die Orientierung des Roboters aus der Vision (grau) und die Vorhersage für vier Frames des neuronalen Netzes (schwarz). Bild entnommen aus [1].

Gleiches ist in Abbildung 7.4 für die Orientierung zu sehen. Allerdings ist die vorhergesagte Kurve weniger glatt. Dies liegt einerseits daran, dass das Rauschen der Vision in der Orientierung höher ist als in der Position, andererseits sehr schnelles Drehen in der

Trainingsmenge enthalten waren, das von dem neuronalen Netz schwerer erlernt werden kann.

Ausblick Inzwischen wird im System der FU-Fighters anstelle des neuronalen Netzes auch eine lineare Regression zur Approximation verwendet, die noch bessere Ergebnisse liefert, wie unter [1] beschrieben.

Zukünftig sollen die Trainingsdaten unabhängig von der Framerate bestimmt werden und auch während des Spiels gelernt werden können, um sich an Veränderungen der Motoren oder des Batterieladestatus anpassen zu können. Dazu müssen schlechte Daten, wie z.B. Kollisionen, automatisch erkannt und vom Training ausgeschlossen werden.

8 Ergebnisse und Ausblick

8.1 Messen des Rauschens des Vision-Systems

Der absolute Fehler ist der Differenzbetrag einer Messung in der Vision zu der wirklichen Position auf dem Feld. Diese lässt sich daher nur durch Messungen auf dem Feld bestimmen und ist abhängig von der Genauigkeit der Abbildungsfunktion von Pixelkoordinaten in das Feldkoordinatensystem.

Noch wichtiger allerdings für eine gute Steuerung ist der relative Fehler, da das Verhalten eines Roboters stark von der relativen Position des Balls und der anderen Roboter abhängt. Auch für die Vorhersage der einzelnen Objekte (siehe [1]) ist es wichtig zu wissen, in welcher Größenordnung die erkannten Positionen und Orientierung der Vision von Rauschen behaftet sind. So kann die Kenntnis bei einer Kalmanfilterung des Balls verwendet werden.

Um nun das Rauschen der Lokalisierung eines Roboters zu messen, gehen wir von der Annahme aus, dass die Bewegung sowohl in der Position als auch in der Orientierung in der wirklichen Welt glatt ist. Dazu verwendet man an jeder Stelle der Bewegungskurve eines Roboters jeweils die drei vorherigen und die drei nachfolgenden Punkte. Anhand dieser 6 Punkte kann der mittlere per linearer Regression bestimmt werden. Unter der Annahme, dass die Welt glatt ist, ist dieser vorausgesagte Wert besser als der in der Vision gemessene. Die Differenz entspricht dem Rauschen der Vision.

Mit diesem Ansatz ergibt sich für mein System ein Rauschen der Position von $2.1mm$ in x und y-Richtung, d.h. insgesamt $3mm$ - dies entspricht einer Genauigkeit unter einem Pixel. In Anbetracht der Geschwindigkeit der Roboter von bis zu $3m/s$ ist dies ein erstaunlich gutes Ergebnis.

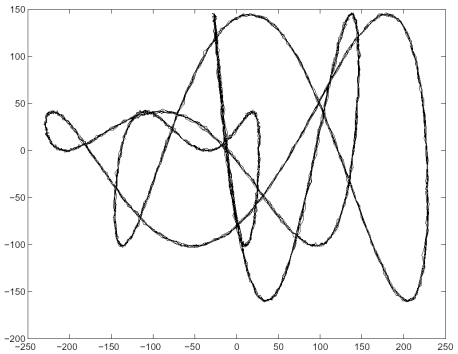


Abbildung 8.1: Künstlich erzeugte Daten mit gaußschem Rauschen.

Mit derselben Methode beträgt das Rauschen der Vision bei der Bestimmung des Orientierung des Roboters 5.58 Grad. Das entspricht einer Abweichung von $5.58/360 = 1.55$ Prozent.

Die Korrektheit dieses Ansatzes wurde getestet, indem künstlich erzeugte Daten mit gauss'schem Rauschen versehen wurden. Die Methode überschätzte dabei das Rauschen um 20 Prozent. Folglich stellen die gemessenen $3mm$ bzw. 5.58 Grad eine obere Grenze dar.

8.2 Zusammenfassung und Ausblick

Ich habe gezeigt, wie man mit den in dieser Arbeit entwickelten Methoden ein robustes und adaptives Vision-System entwerfen kann. Es ermöglicht die Verarbeitung von Bildströmen zweier Kameras bei einer Bildwiederholungsrate von 53Hz und einer Bildauflösung von 780×582 Pixeln an einem einzigen Rechner bei durchschnittlicher Rechenlast in beinahe allen Situationen.

Nach dem Prinzip von variablen Suchrahmen werden Objekte verfolgt, ihre zukünftige Position vorhergesagt, so dass nur kleine Bereiche des Bildes untersucht werden müssen.

Mithilfe von qualitätsabhängiger Adaption und adaptiven Farbkarten ist es dem System möglich, sowohl mit starken räumlichen Unterschieden, als auch langsamen zeitlichen Veränderungen der Beleuchtung umzugehen.

Es wurde eine Vielzahl verschiedener Robotermodelle implementiert, so dass sich unterschiedlichste Roboter anderer Teams mit höherer Präzision als nur anhand des Teammarkers verfolgen lassen.

Das System kann in wenigen Minuten kalibriert werden. Aufgrund des Prinzips des Trackings und der adaptiven Farbkarten können die Farben sogar während des Spiels automatisch kalibriert werden. Es genügt jede Farbe an einer Stelle einmal per Mausklick zu initialisieren, die restliche Karte wird automatisch erlernt, während sich der Roboter über das Feld bewegt. In Zukunft könnte diese einmalige Initialisierung auch automatisiert werden. Dazu müsste eine Methode farbige Marker auf dem Feld finden, diese anhand der groben Farbrichtung der entsprechenden Farbkarte zuordnen und diese initialisieren. Ebenso könnten so gefundene Marker anhand des Abstands zu Gruppen zusammengefasst werden, und anhand ihrer geometrischen Anordnung ein geeignetes Robotermodell zur Verfolgung automatisch ausgewählt werden.

Die Kamerakalibrierung erfolgt noch unkomplizierter. Aufgrund der getrennten Behandlung von radialer Verzerrung und projektiver Transformation, genügen wenige Kalibrierungspunkte, um eine akkurate Transformationsfunktion zu bestimmen. Mithilfe analytischer Methoden, die die dreidimensionale Kameraposition anhand der projektiven Transformation ermittelt, kann sowohl der Abstand der Kamera zum Spielfeld, als auch ihren Lotpunkt auf dem Feld automatisch bestimmt werden und muss nicht mehr manuell aufwendig gemessen werden. Schließlich wurde eine automatische Kalibrierungsmethode entwickelt, die die Parameter der Transformationsfunktion anhand der Linien auf dem Feld selbstständig findet und optimiert. Auf diese Weise ist es dem Benutzer möglich die gesamte Kamerakalibrierung per Knopfdruck durchzuführen.

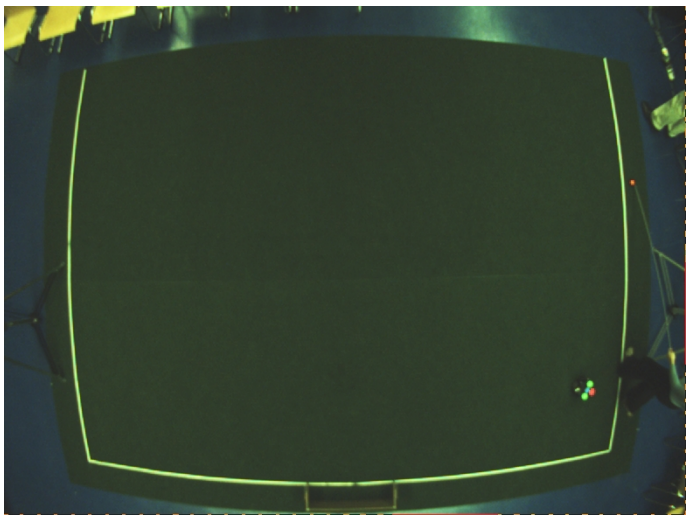


Abbildung 8.2: Test auf einem $8m \times 10m$ großen Spielfeld. Das entspricht der doppelten Fläche des offiziellen Feldes, das momentan in der Small-Size Liga verwendet wird.

FU-Vision, das in dieser Arbeit beschriebene Computer-Vision-System, wurde auf etlichen Turnieren und Präsentationen erfolgreich eingesetzt und hat zu den Erfolgen der FU-Fighters entscheidend beigetragen. Diese sind vierfacher Gewinner der *German Open*, *European Champion 2000*, sowie zweifacher und amtierender Weltmeister. 2004 erhielt das Vision-System zudem den *Engineering Award* in der Small-Size Liga.

Schließlich wurde FU-Vision 2005 veröffentlicht und anderen Teams zur Verfügung gestellt. So stand auf der Weltmeisterschaft 2005 in Osaka neben den *FU-Fighters* mit *BigRed*, das sehr erfolgreiche Team der Universität Cornell, noch ein zweites Team im Finale, das das *FU-Vision* System verwendete.

Als letztes wurde das System erfolgreich auf einem doppelt so großen Feld getestet. Es wurden keine Veränderungen oder Parameteranpassungen vorgenommen, lediglich die neue Feldgröße eingegeben. Abbildung 8.2 zeigt ein Bild dieses Feldversuchs. Das System verhielt sich dabei ohne erkennbaren Unterschied zum jetzigen Feld. Somit wurde gezeigt, dass *FU-Vision* jetzt schon für künftige Feldvergrößerungen vorbereitet ist.

Literaturverzeichnis

[1] Sven Behnke, Anna Egorova, Alexander Gloye, Raúl Rojas, and Mark Simon. Predicting away robot control latency. In *RoboCup*, pages 712–719, 2003. 25, 69, 70, 71, 72, 73, 75

[2] James Bruce and Manuela Veloso. Fast and accurate vision-based pattern detection and identification. In *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan, May 2003. 53

[3] G. Buchsbaum. A spatial processor model for object color perception. *Franklin Inst.*, 310:1–26, 1980. 23

[4] Leighton Carr. Vision software for the 2003 roboroos. bachelor degree thesis, School of Information Technology and Electrical Engineering, University of Queensland, 2003. 64

[5] Anna Egorova, Mark Simon, Fabian Wiesel, Alexander Gloye, and Raúl Rojas. Plug and play: Fast automatic geometry and color calibration for cameras tracking robots. pages 394–401. 37, 64

[6] Wolf Lindstrot Mark Simon Lars Wolter Felix v. Hundelshausen, Kirill Koulechov and Raúl Rojas. Körper und seele einer neuen maschine, förderpreis. *Deutscher Studienpreis, BODYCHECK Wieviel Körper braucht der Mensch ?* 7

[7] Ketill Gunnarsson, Fabian Wiesel, and Raúl Rojas. The color and the shape: Automatic on-line color calibration for autonomous robots. In Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld, and Yasutake Takahashi, editors, *RoboCup-2005: Robot Soccer World Cup IX*, Osaka, Japan, 2006. LNAI, Springer. 30

[8] Peter Haberaecker. *Praxis der Digitalen Bildverarbeitung und Mustererkennung.* Carl Hanser Verlag, Muenchen, 1995. 12

[9] Bastian Hecht, Alexander Gloye, Achim Liers, Marian Luft, Artem Petakov, Raúl Rojas, Mark Simon, Oliver Tenchio, and Fabian Wiesel. Fu-fighters small size 2005. In Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld, and Yasutake Takahashi, editors, *Proceedings of The 9th RoboCup International Symposium*, Osaka, Japan, 2005. 4

[10] H. Jaeger and T. Christaller. Dual dynamics: Designing behavior systems for autonomous robots. In S. Fujimura and M. Sugisaka, editors, *Proceedings International Symposium on Artificial Life and Robotics (AROB '97), Beppu, Japan*, pages 76–79, 1997. 7

[11] R. Kimmel. Demosaicing: Image reconstruction from color CCD samples. *IEEE Trans. Image Processing*, 8(9):1221–1228, September 1999. 23

[12] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife*, 1995. 1

[13] P. B. Delahunt P. Long'ere, X. Zhang and D. H. Brainard. Perceptual assessment of demosaicing algorithm performance. *Proc. IEEE*, vol.90:123–132, Jan. 2002. 23

[14] W. E. Snyder R. Ramanath and G. L. Bilbro. Demosaicking methods for bayer color arrays. *J. Electronic Imaging*, vol.11:306–315, July 2002. 23

[15] Mark Simon Raul Rojas and Oliver Tenchio. Parabolic flight recontruction from multiple images from a single camera. In *Proceedings of The 10th RoboCup International Symposium*, Bremen, Germany, 2006. Submitted for review. 42, 43

[16] R. Rojas. *Theorie der neuronalen Netze*. Springer, 1993. 72

[17] Raúl Rojas. Calibrating an overhead video camera. Arbeitsgruppe Künstliche Intelligenz, Institut für Informatik, Freie Universität Berlin, January 2004. 60, 62, 63

[18] Mark Simon, Sven Behnke, and Raúl Rojas. Robust real time color tracking. *Lecture Notes in Computer Science*, 2019:239–??, 2001. 7

[19] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Trans. Robotics and Automation*, 3(4):323–344, 1987. 58

[20] Fabian Wiesel. Kontrolle und steuerung von omnidirektionalen robotern. Master's thesis, Freie Universität Berlin, February 2006. 7