Kapitel 4

Eingesetzte Roboterplattform

In diesem Kapitel gebe ich einen kurzer Überblick über das Small-Size-Team der FU-Fighters, denn dort habe ich die in den Kapiteln 5 bis 9 beschriebene Verfahren bisher eingesetzt und es existiert bisher keine aktuelle, ausführliche Beschreibung des Systems. Die FU-Fighters nehmen seit 1999 in der Small-Size-Liga [Behnke, 2000, Egorova, 2004] und seit 2002 auch in der Middle-Size-Liga [Behnke, 2002] am RoboCup teil. In [Ackers, 1999] ist eine ältere Beschreibung des Systems zu finden. Vieles hat sich jedoch seit dem so weiterentwickelt, dass eine aktuelle Beschreibung notwendig ist. Eine Beschreibung der Hardwareentwicklung seit den Anfängen der FU-Fighters ist in [Gloye, 2003] zu finden.

4.1 Systemüberblick

Die Regeln der Small-Size-Liga geben den groben Aufbau des Systems vor. Abbildung 2.2 aus Kapitel 2 zeigt den prinzipiellen Aufbau des Small-Size-Spielfelds. Eine oder mehrere Kameras hängen etwa 4m über einem 4m × 5m großen Spielfeld. Die Kameras sind mit einer Zentraleinheit des Teams verbunden, welche die Bildauswertung durchführt, indem die Positionen und Orientierungen der Roboter und die Position des Balls bestimmt werden. Bei den FU-Fighters ist der Zentralrechner auch für das Verhalten der Roboter zuständig. Er koordiniert das Team und sendet über ein Funkmodul Steuersignale an die Roboter.

Ein Mikrocontroller auf dem Roboter empfängt die Daten über ein entsprechendes Funkmodul, berechnet die Regelwerte für die Motoren und überwacht die Geschwindigkeiten der Räder. Die Systemschleife auf dem Roboter wird alle

8ms wiederholt.

Abbildung 4.1 zeigt die Ausführungsschleife des FU-Fighters Systems, die 30 bis 60 mal in der Sekunde — durch die Bildwiederholrate der Kamera vorgegeben — durchlaufen wird. Im Folgenden wird auf die einzelnen Komponenten der FU-Fighters genauer eingegangen.

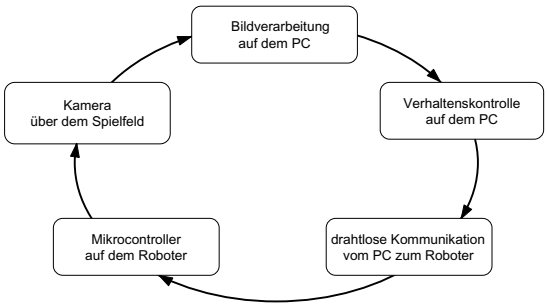

Abbildung 4.1: Die Systemschleife des FU-Fighters Small-Size-Teams. Sie besteht aus fünf Hauptelementen, die nacheinander ausgeführt werden und jeweils Daten für die nächste Einheit liefern. Die Schleife wird — vorgegeben durch die Bildwiederholrate der Kamera — zwischen 30 und 60 mal in der Sekunde durchlaufen.

4.2 Bildverarbeitung

Angefangen haben die FU-Fighters in der Small-Size-Liga mit einer NTSC Kamera, die über ein S-Video Kabel an eine preiswerte Frame-Grabber-Karte eines PCs angeschlossen wurde. Später kam eine zweite Kamera hinzu, da sich das Spielfeld vergrößerte und mit einer Kamera die Objekte zu klein und die Ballüberdeckung zu stark wurde (siehe Abbildung 4.2).

Um eine höhere Bildwiederholrate zu erreichen und die Nachteile der Halbbilder von NTSC- und PAL-Kameras zu unterdrücken wurden ab 2004 zwei progressive-scan IEEE 1394 (Fire-Wire) Kameras mit einer Bildwiederholrate von bis zu 60 Herz eingesetzt. Zur Zeit benutzen die FU-Fighters zwei Marlin F-046C progressive-scan Kameras der Firma Allied Vision Technology (AVT) mit einer Auflösung von 780x582 Punkten und einer maximalen Bildwiederholrate von 53Hz. Entscheidend für die Wahl der Kamera war die höhere Auflösung und ein geringfügig besseres Bild.[1]

[1] Ein Überblick über aktuelle Fire-Wire Kameras gibt die Seite www.tele.ucl.ac.be/PEOPLE/DOUXCHAMPS/ieee1394/cameras/.

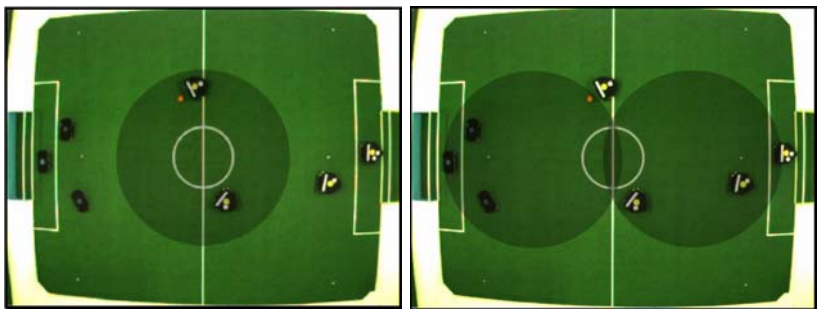

Abbildung 4.2: Ballüberdeckung mit einer (links) oder zwei (rechts) Kameras. In den dunkelgrünen Bereichen ist der Ball immer zu sehen, auch wenn ein Roboter mit voller Höhe von 15cm in Richtung der Kamera vor dem Ball steht. Der sichtbare Bereich hängt auch von der Kamerahöhe ab. Um so niedriger die Kamera hängt, um so weniger ist der Ball zu sehen. Bilder aus [Carr, 2003].

4.2.1 Bilderfassung

Digitale Farbkameras verwenden entweder für jede Grundfarbe einen eigenen CCD-Chip oder benutzen ein Farbfilter über einem CCD-Chip. Ein sehr verbreiteter Filter ist der Bayerfilter (siehe Abbildung 4.3 links). Die von den FU-Fighters verwendeten AVT Kameras haben nur einen CCD-Chip.

Die von der Kamera gelieferten Daten sind die Helligkeitsinformationen der CCD-Punkte. Um ein Farbbild zu erhalten müssen die fehlenden Farbinformationen geschätzt werden. In Abbildung 4.4 ist links ein Ausschnitt des von der Kamera gelieferten Bildes zu sehen. Zur Veranschaulichung sind im mittleren Bild die Pixel entsprechend ihrer gemessenen Farbintensität eingefärbt. Im rechten Bild sind die Farbwerte linear interpoliert.

Bessere Ergebnisse der Interpolation erhält man durch komplexere Algorithmen, wie sie zum Beispiel in [Longère, 2002, Kimmel, 1999, Muresan, 2002] beschrieben sind. Diese besseren Verfahren sind jedoch sehr rechenintensiv und für eine Echtzeitbildverarbeitung nicht sinnvoll. In der Bildverarbeitung der FU-Fighters wird sogar ein noch einfacherer Filter verwendet. Es werden dabei immer die Informationen von vier benachbarten Farbwerten für jeden Farbwert summiert. Die Auflösung der blauen und roten Farben ist dann zwar geringer, aber das Verfahren ist sehr schnell. Einen Nachteil gibt es allerdings. An Farbkanten — besonders an weißen Linien — kommt es zu falschen Farbrändern, welche die Bildauswertung fehleranfällig machen würde (siehe Abbildung 4.3). Um dies zu verhindern werden die weißen Linien auf dem Feld durch eine zusätzliche Maske ausgeblendet.

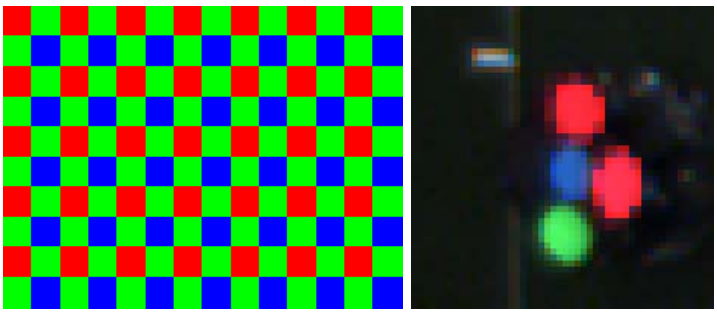

Abbildung 4.3: Beim Bayerfilter gibt es genauso viele grüne wie rote und blaue Farbwerte zusammen (rechts). Der Grund für diese Wahl liegt im menschlichen Auge, das viel sensitiver auf grün reagiert. Durch eine einfache Interpolation scheint die Auflösung für blaue und rote Objekte gegenüber den grünen Objekten pro Dimension nur halb so groß zu sein (rechts). In dieser Abbildung scheint der blaue Marker etwas besser aufgelöst zu sein. Dies liegt jedoch daran, dass der rote Marker gesättigter ist, also ein reineres rot ist. Deutlich sind auch die Farbränder an einer weißen Linie zu sehen.

4.2.2 Farbkodierung der Identität der Roboter

Wie schon in Kapitel 2 vorgestellt, haben die Roboter eine Farbkodierung auf der Oberseite, da sie immer von der Kamera von oben gesehen werden. Dabei gibt es nur einen obligatorischen Marker, den gelben oder blauen Teammarker in der Mitte, an denen die beiden Teams unterschieden werden. Die Teams dürfen außerdem alle anderen nur denkbaren Marker benutzen; sie müssen sich dabei nur gut genug von den Teammarkern und dem Ball unterscheiden. Die zusätzlichen Markierungen werden zur Bestimmung der Orientierung des Roboters und seiner Identität benutzt.

Die ursprüngliche Kodierung der Roboter der FU-Fighters bestand aus drei in einer Linie angeordneten Farbmarker unterschiedlicher Farben: Meistens links ein grüner Marker, in der Mitte der Teammarker und rechts ein roter Marker. Zur Orientierungsbestimmung würde eine Farbmarke zusätzlich zur Teammarkierung ausreichen. Es wurden jedoch zwei unterschiedliche Farbmarken benutzt, da dadurch das Finden der Roboter sicherer ist, falls eine der drei Farben wegen unterschiedlicher Beleuchtung oder ungünstigen Reflektionen nicht korrekt zu identifizieren ist.

Aus dieser Farbmarkeranordnung ist jedoch nur die Orientierung der Roboter herleitbar. Später wurden deshalb die IDs binär mit drei weißen bzw. schwarzen Quadraten vorne am Cover kodiert. Später dann mit zwei Flächen vorne und hinten am Roboter (siehe Abbildung 4.5). Dadurch stieg die Anzahl der möglichen Kodierungen von 3 Bit auf 4 Bit und es konnten dann die

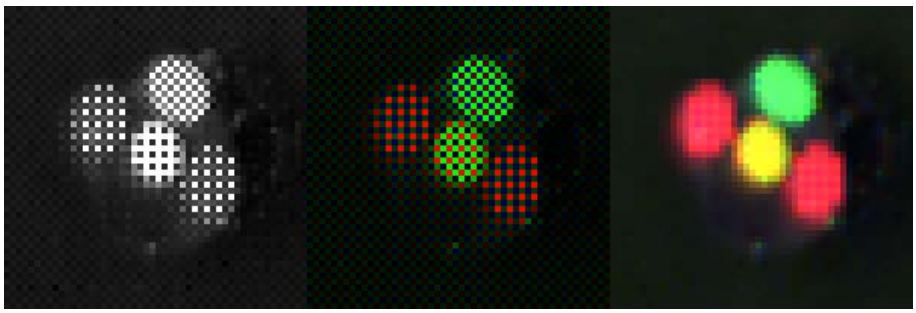

Abbildung 4.4: Rechts die Information, wie sie von der Kamera geliefert wird. In der Mitte mit den zugehörigen Intensitäten der zugeordneten Farben. Rechts eine lineare Interpolation für alle Bildpunkte.

14 IDs `0001`, `0010`, `0011`, `0100`, `0101`, `0110`, `0111`, `1000`, `1001`, `1010`, `1011`, `1100`, `1101`, `1110` statt der 6 IDs `001`, `010`, `011`, `100`, `101`, `110` verwendet werden. Kodierungen ausschließlich aus Nullen (schwarz) und Einsen (weiß) wurden ausgeschlossen, da die Grenze zwischen schwarz und weiß dynamisch anhand der lokalen Helligkeitsdifferenzen bestimmt wird. Dazu sind mindestens ein schwarzes Quadrat und ein weißer Quadrat notwendig.

In Abbildung 4.6 sind die Kodierungen für die Orientierung und Identifikationsnummer angegeben, wie sie von den FU-Fighters seit 2004 eingesetzt werden. Sie sind von den Codierungen der Cornell Big Red[2] abgeleitet, mit dem Unterschied, dass die Farbmarker oval statt rund sind und nur drei statt vier Marker für die Kodierung verwendet werden. Pro Team können bei dieser Kodierung mit zwei Farben zusätzlich zur Teamfarbe acht Roboter gleichzeitig verfolgt werden. Für mehr Roboter eines Teams ist eine weitere Farbe notwendig, da bei einem vierten Marker der vorgegebenen Farben der Informationsgewinn nicht steigt. Soll die Anzahl der Farben gleich bleiben, dann müssen die Markierungen so angeordnet werden, dass sie nicht rotationssymmetrisch sind, wie bei der Kodierung des Teams der Carnegie Mellon University [Bruce, 2003].

Einen guten Überblick über die Probleme, die bei der Small-Size-Liga auftreten, und über eine grundsätzlich andere Kodierung für die Orientierung und Nummerierung der Roboter stellt eine Arbeit des RoboRoos-Teams vor [Nolan, 2002]. Das Team benutzt keine Farbmarker, sondern weiße Striche und Punkte auf dem Cover. Dies hat zwar den Vorteil, dass für das eigene Team nur die Teamfarbe kalibriert werden muss, aber den Nachteil, dass die Roboter leichter von der Vision verloren werden können. Eine weitere Kodierung für die Orientierung erfordert eine radiale Abtastung in einem gewissen Abstand um den Teammarker (siehe Abbildung 4.7). Diese Art der Kodierung wird von vie-

[2] robocup.mae.cornell.edu

Abbildung 4.5: Die Farbkodierung, wie sie von den FU-Fighters bis einschließlich 2003 eingesetzt wurde. Links ohne Identifikationskodierung. Die Roboter mussten für die Bildverarbeitung anfänglich zugeordnet werden und durften nicht verloren werden. In der Mitte die 3 Bit Kodierung (hier ID 6) und rechts die 4 Bit Kodierung (hier ID 11), bei der nicht nur die Anzahl der möglichen IDs größer war, sondern auch die Größe der Kodierungsfläche. Dadurch konnte das Spielfeld vergrößert werden, ohne dass die ID-Marker relativ zum Spielfeld kleiner wurden. Das niederwertigste Bit ist vorne links, das höchstwertige bei 4 Bit Kodierung hinten rechts und bei 3 Bit Kodierung vorne rechts.

len Teams, zum Beispiel 5dpo[3], ITUFlash[4] und RoboDragons[5] verwendet, hat jedoch den weiteren Nachteil, dass der Orientierungsfehler mit 8^o recht groß ist [Hibino, 2002]. Die Kalibrierung der Farben kann aber automatisch geschehen (siehe Abschnitt 4.2.4), sodass der Vorteil der Kodierung ohne Farben gering ist [Egorova, [2]2004].

4.2.3 Auswerten der Bilder

Die Aufgabe der Bildverarbeitung ist es, die ankommenden Bilder aus den Kameras so schnell wie möglich zu analysieren und die Position des Balls und die Positionen und Orientierungen der Roboter zu finden. Vier Anforderungen spielen dabei eine besondere Rolle: Schnelligkeit, Genauigkeit, Zuverlässigkeit und Robustheit. Je schneller die Bildverarbeitung funktioniert, desto schneller hat die Verhaltenskontrolle die nötigen Informationen, um neue Befehle für die Roboter zu generieren. Je genauer die Daten aus der Bildverarbeitung sind, desto präziser kann die gegebene Situation abgeschätzt werden und eine effektive und genaue Strategie für das Verhalten gebildet werden. Im Folgenden werden die Hauptansätze für die Auswertung der Bilder bei den FU-Fighters präsentiert.

Eine ältere Darstellung der Bildverarbeitung ist in einer Publikation aus dem Jahr 2001 zu finden [Simon, 2001]. Sie bezieht sich auf die Bildverarbeitung mit nur einer NTSC-Farbkamera, die über eine preiswerte Frame-Grabber-Karte mit

[3] paginas.fe.up.pt/~robosoc
[4] www.iut.ac.ir/robocup
[5] www.aichi-pu.ac.jp/ist/lab/narulab/indexE.html

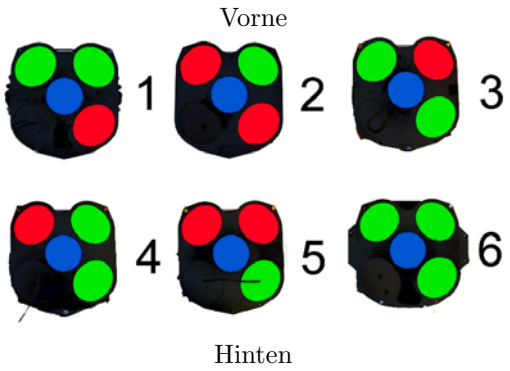

Abbildung 4.6: Die Farbkodierung, wie sie von den FU-Fighters seit 2004 eingesetzt wird. Im Zentrum der Teammarker. Außen die Marker für die Orientierung und Roboternummer in zwei gesättigten Farben (rot und grün). Die Cover mit den Kodierungen (grün, rot, rot) und (rot, rot, rot) werden zur Zeit nicht benutzt (im Uhrzeigersinn vorne links beginnend).

dem Computer verbunden ist. Dabei entstehen andere Probleme, insbesondere auf Grund des Zeilensprungverfahrens der Kamera. Außerdem wird in der Arbeit nicht auf die gleichzeitige Verwendung mehrerer Kameras eingegangen.

Zur Zeit wird an der Veröffentlichung des Bildverarbeitungs-Systems der FU-Fighters gearbeitet. Auf der Webseite[6] der FU-Fighters wird es zu gegebener Zeit genauere Informationen geben.

Wie weiter oben erwähnt wurde, basiert die Analyse der Bilder auf den Farbmodellen, die für die Roboter und den Ball existieren. Jeder Roboter, sei er vom eigenen oder vom gegnerischen Team, verfügt über ein eindeutiges Modell, das ihn von allen anderen unterscheidet. Dieses Modell besteht aus einer bestimmten Anordnung von farbigen Markern auf seiner Oberseite. Die Form der Marker spielt dabei keine Rolle, es wird aber eine kompakte Darstellung erwartet. Zum Beispiel werden lange, dünne Linien nicht als Marker identifiziert, weil sie nicht kompakt sind.

Einen Überblick der Bildauswertung gibt Abbildung 4.8. Ist das Bild einer Kamera erfasst, wird die zugehörige Bildauswertung gestartet. Um diese zu beschleunigen wird versucht, nur Teile des Bildes zu betrachten. Dazu greift die Bildauswertung auf ein globales Weltmodell der Positionen, Orientierungen und Geschwindigkeiten der zu verfolgenden Objekte zurück. Wir nehmen an, dass sich die Objekte zwischen zwei aufeinanderfolgenden Bildern nicht sehr weit bewegen. Deswegen kann die nächste Position der Objekte relativ gut anhand seinen letzten Position und Geschwindigkeit vorhergesagt und nur eine kleine Fläche im Bild durchsucht werden. Dies wird als lokale Suche bezeichnet.

[6] www.fu-fighters.de

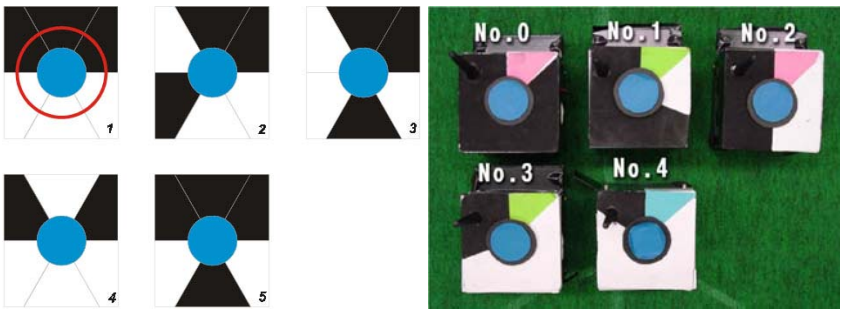

Abbildung 4.7: Links die Cover des Teams 5dpo, mit der Kodierung der IDs und Orientierung durch schwarze und weiße Flächen um den Teammarker (bearbeitete Abbildung aus [Costa, 2002]). Der rote Kreis zeigt die radiale Abtastung um den Teammarker zum Finden der Übergänge zwischen schwarz und weiß. Rechts die Kodierung durch das RoboDragons-Team nach dem gleichen Prinzip aber mit zusätzlichen Farben (Abbildung aus [Hibino, 2002]).

Schlägt die lokale Suche fehl, wird der Suchrahmen bis zu einer maximalen Größe expandiert. Ist dann das gesuchte Objekt noch nicht gefunden, wird die globale Suche aufgerufen. Nachdem beide Kamerabilder ausgewertet wurden, wird eine Fusion der Daten durchgeführt. Wir gehen jetzt auf die einzelnen Schritte genauer ein.

Lokale Suche

Als erstes werden in den kleinsten, lokalen Suchrahmen der vermuteten Positionen alle Objekte gesucht. Dafür wird für jedes Objekt das für den gesuchten Marker beste Pixel im Suchrahmen gesucht. Anschließend wird der zugehörige Segmentierer an der Stelle des besten Pixels aufgerufen. Wurden die Objekte gefunden, so werden sie markiert. Sollten sie nicht gefunden worden sein, so wird der Suchrahmen für diese Objekte vergrößert. Dies geschieht so lange, bis alle Objekte gefunden wurden oder eine maximale Suchrahmengröße erreicht wurde. Wurden nicht alle Objekte gefunden, startet die globale Suche.

Für die Suche des besten Pixels im Suchrahmen können unterschiedliche Verfahren angewendet werden. Zum Beispiel können nacheinander alle Pixel im Suchrahmen betrachtet werden oder es kann ein Subsampling auf den Suchrahmen angewendet werden. Die sind nur zwei Möglichkeiten, weitere Methoden stehen zur Verfügung.

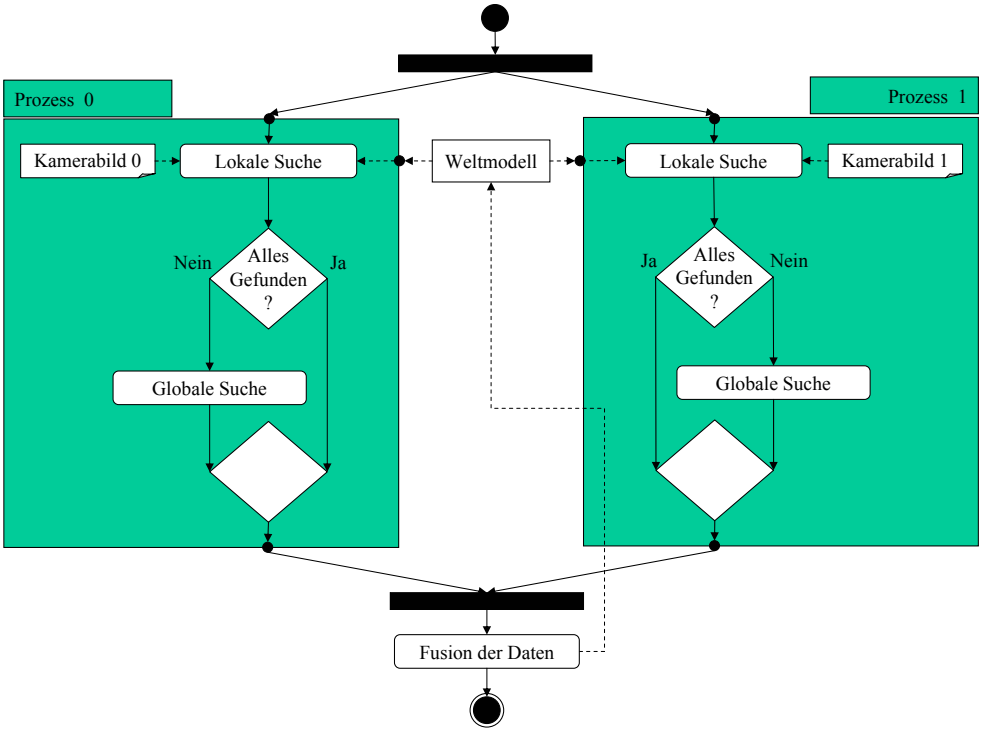

Abbildung 4.8: Die Bilder der Kameras werden gleichzeitig ausgewertet. Zur optimalen Suche der Objekte wird ein gemeinsames Weltmodell benutzt. Nachdem die Daten aller Kameras vorliegen, werden die Daten fusioniert und an das Verhaltenssystem weitergegeben. Anschließend wird auf neue Kamerabilder gewartet.

Globale Suche

Ein globales Suchfenster wird benutzt, wenn ein Objekt verloren wurde. Der Ball geht beispielsweise oft verloren, weil er von den Robotern verdeckt wird (siehe Abbildung 4.2). Die Roboter selbst können auch verloren gehen, wenn sie zum Beispiel von einem Menschen verdeckt werden, vom Spielfeld entfernt werden, umkippen und ihre Farbmarkierungen nicht mehr zu sehen sind oder sich die Lichtverhältnisse abrupt ändern. In diesen Fällen wird eine globale Suche nach dem verlorenen Objekt gestartet.

Eine globale Suche geht über alle Farben des nicht gefundenen Objekts. Dafür wird ein Distanzbild des aktuellen Kamerabildes mit der Farbkarte jeder gesuchten Farbe erstellt. Gleichzeitig wird ein zweifaches Subsampling des Bildes durchgeführt, um das Verfahren zu beschleunigen und das Rauschen im Bild zu

reduzieren. Da sehr viele gute Pixel für eine Farbe gefunden werden können, ist der Aufruf des Segmentierers für einige Modelle, anderes als in der lokalen Suche, an weitere Bedingungen geknüpft. Nur wenn für ein Objekt hinreichend viele beste Pixel für die Marker des Modells gefunden und stimmt die Anordnung der Pixel ungefähr mit dem Modell des Objekts überein, dann wird der zugehörige Segementierer an der Stelle des besten Pixels des mittleren Markers aufgerufen.

Segmentierer

Für jede Farbe und für einige Roboter gibt es eigene Segmentierer. Außerdem können für jede Farbe unterschiedliche Segmentierungsfunktionen benutzt werden. Der Segmentierer arbeitet in der Regel im HSI-Farbraum, kann aber auch wie die Pixelsuche im RGB-Farbraum arbeiten. Im HSI-Farbraum zu rechnen ist zwar etwas aufwändiger als im RGB-Farbraum, aber dafür variieren die Distanzen des Farb- und Sättigungs-Kanals nicht so stark. Die Ergebnisse sind sicherer, gerade bei gesättigten Farben, wie sie die FU-Fighters einsetzen.

Der einfachste Segmentierer sucht in einem vorgegebenen Rahmen alle Pixel, die innerhalb eines gewissen Toleranzbereichs der gesuchten Farbe entsprechen (siehe Abbildung 4.9). Für alle diese Pixel wird dann der gemeinsame Schwerpunkt, der durchschnittliche Farbwert, die Anzahl der gefundenen Pixel (also die Größe des Markers) und die Kompaktheit gemessen. Diese Werte bilden dann die Qualität des gefundenen Markers.

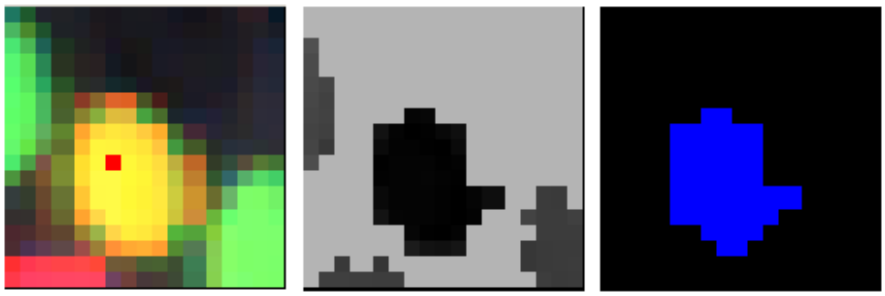

Abbildung 4.9: Bei der Farbmarkensuche wird in einem Bereich des Bildes das beste Pixel bezüglich der Farbkarte im RGB-Raum gesucht. Um dieses Pixel (das Zentrum im linken Bild) wird dann ein Rahmen gelegt, in dem alle Pixel im HSI-Raum nach einem Schwellwertverfahren klassifiziert werden. Hinreichend gute Pixel werden zur Bestimmung des Markers herangezogen. Im mittleren Bild sind die Pixel dargestellt, welche die Helligkeits- und Intensitäts-Bedingung erfüllen. Im rechten Bild sind die Pixel dargestellt, die zusätzlich dem Farbwert hinreichend genügen. Der rote Punkt im linken Bild gibt den Schwerpunkt der zugewiesenen Pixel an.

Ein anderes Segmentierungsverfahren sucht mit einem Füllalgorithmus um den besten Pixel herum passende Farben. Ein weiterer Segmentierer sucht die Pixel im RGB-Farbraum. Für bestimmte Roboter gibt es außerdem Segmentierer, die verschmolzene Marker einer Farbe wieder trennen, indem eine Trennungslinie vom Mittelpunkt des Roboters aus durch eine zu große Farbfläche gelegt wird.

Wenn alle Farben des Modells gefunden wurden, wird ihre Anordnung überprüft und es wird die Summe der Qualitäten der Farbmarker berechnet, um endgültig zu entscheiden, ob das Objekt zu diesem Modell gefunden wurde. Es wird auch längerfristig die Qualität des Modells beobachtet. Falls die Farbe und die Größe der Marker dem Modell ähnlich sind, ist die Qualität hoch. Falls die Qualität eines Objekts über mehrer Frames niedrig ist, wird das Objekt für verloren erklärt und eine globale Suche gestartet.

Ist ein Objekt sicher gefunden, dann werden die zugehörigen Farbkarten aktualisiert.

Datenfusion

Da zwei Kameras benutzt werden, müssen die Ergebnisse der Analyse der beiden Bilder konsolidiert werden, um das gemeinsame Weltmodell zu aktualisieren. Dabei werden auch die Positionen der Objekte interpoliert, die in beiden Kamerabildern zu sehen sind, um einen glatten Übergang zu gewährleisten, wenn ein Roboter oder der Ball sich von einer Seite des Spielfelds auf die andere Seite bewegt. Bei der Fusion wird außerdem auf die Plausibilität der Daten geachtet. Zum Beispiel darf keine Roboter-ID doppelt vorkommen.

4.2.4 Kalibrierung der Bildverarbeitung

Die meiste Zeit beim Setup der Small-Size-Liga wird durch die Kalibrierung der Bildverarbeitung aufgewendet. Die Farben müssen den Beleuchtungsverhältnissen entsprechend eingestellt werden, nicht nur die eigenen Farben, sondern auch die Farben des aktuellen Gegners. Die Verzerrung der Kamera und die Transformation der Kamerakoordinaten in die Spielfeldkoordinaten muss bestimmt werden.

Um diesen Prozess zu beschleunigen wurden im FU-Fighters-System einige Automatismen eingeführt, auf die jetzt genauer eingegangen wird.

Kalibrierung der Farben

Durch die Entwicklung natürlichere Beleuchtungsverhältnisse bei Wettbewerben zu verwenden, verwenden immer mehr Teams Farbkarten. Eine Farbkarte ist eine Darstellung des ganzen Spielfelds mit Angaben über die tatsächliche Farbe und Größe der benötigten Farbmarker der Modelle an mehreren Knotenpunkten

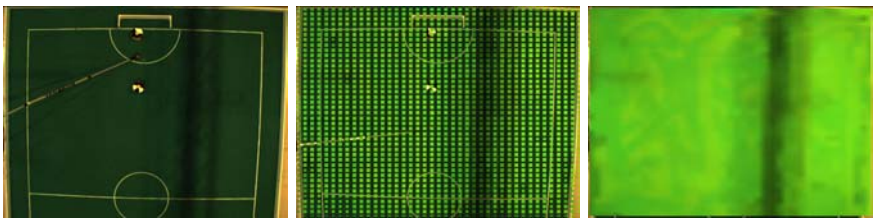

Abbildung 4.10: Eine Beispiel-Farbkarte der FU-Fighters für die Farbe Grün. Besonders bei einem unregelmäßig beleuchteten Feld (links) sind Farbkarten notwendig. Es müssen Karten für jede Farbe erstellt und bei jeder stärkeren Änderung der Lichtverhältnisse wieder angepasst werden. Kleine Farbänderungen werden während der Verfolgung von Objekten automatisch angepasst. Die Farbe wird nicht für jeden Punkt auf dem Spielfeld gespeichert, sondern nur für eine bestimmte Anzahl von Knotenpunkten (mittig). Die Farben zwischen ihnen werden interpoliert (rechts).

(siehe Abbildung 4.10). Die Farbkarten werden meistens per Hand erstellt. Eine Probe der Farbe wird an den Knotenpunkten gelegt und die tatsächliche Farbe und Größe in die Karte eingetragen. Dieser Prozess ist langwierig und muss jedes Mal wiederholt werden, wenn sich etwas an den Lichtverhältnissen ändert.

In einer früheren Arbeit der FU-Fighters [Egorova, [2]2004] wurde ein Verfahren entwickelt, das die Farbkalibrierung beschleunigt und vereinfacht. Es beruht auf der Idee, dass die Farbvariationen des grünen Teppichs auf dem Spielfeld mit den Farbvariationen der restlichen benutzten Farben im engen Zusammenhang steht. Dadurch können aus den HSI-Werten des grünen Teppichs die entsprechenden Werte für die anderen Farben geschätzt werden. Dafür sind lediglich zwei Proben auf der Gesamtfläche des Spielfelds erforderlich, um die genauen Parameter der Transformation herauszufinden (siehe Abbildung 4.11).

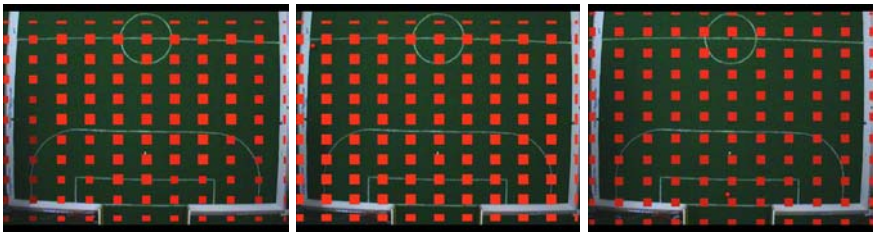

Abbildung 4.11: Links eine von Hand erstellte Farbkarte, in der Mitte eine uniforme Farbkarte und rechts eine automatisch erstellte Farbkarte

Für eine feinere Kalibrierung kann dann ein Roboter langsam über das Spielfeld fahren, um die Farben genauer anzupassen. Durch die grobe Justierung der

Farben geht der Roboter nicht mehr verloren.

Kameraentzerrung

Die Verzerrung des Kamerabildes hat zwei Ursachen: Als erstes ist die Kamera keine „Pin-Hole“ Kamera, welche die ankommenden Strahlen unverzerrt auf den CCD Chip abbildet, sondern hat eine Linse, die das Kamerabild radial verzerrt. Die zweite Ursache ist die Aufhängung der Kamera, die nicht ideal über dem Spielfeld hängt.

Eine weit verbreitete Entzerrungsfunktion ist die Methode von Tsai [Horn, 2000, Tsai, 1986, Tsai, 1987]. Ein Satz von *intrinsischen* und *extrinsischen* Parametern beschreibt dabei die Kamerafunktion. Der Nachteil des Verfahrens ist, dass die intrinsischen Parameter meistens nicht bekannt sind, sie müssen geschätzt werden. Für eine ausreichend gute Kamerakalibrierung werden etwa 50 Beispielpunkte benötigt, von denen die Koordinaten bekannt sein müssen [Wang, 2001]. Das RoboRoos-Team verwendet beispielsweise die Tsai-Methode [Nolan, 2002].

Einige Teams wenden für die Ermittlung der Parameter Teppiche, Papier o. ä. mit einem speziellen Muster (Abbildung 4.12 rechts). Das Team der Cornell University verwendete 2002 und 2003 ein Schachbrettmuster (Abbildung 4.12 links), das auf das Spielfeld gelegt wurde [Loomis, 2003]. In der ersten Version wurden die Schnittpunkte der Karos mit der Maus angewählt. Ein Setup hat so circa eine dreiviertel Stunde gedauert. Eine spätere geringe Verschiebung der Kamera oder des Spielfelds hätte somit gravierende Folgen gehabt. In der Version von 2003 wurde deshalb ein Algorithmus implementiert, der das Verfahren vereinfachte, indem die Ecken der Karos automatisch gefunden wurden.

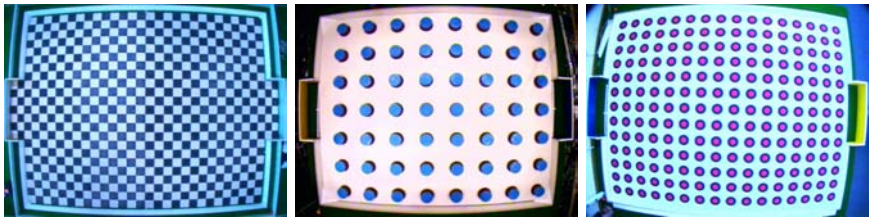

Abbildung 4.12: Links der vom Team BigRed verwendete Teppich, bei dem die Schnittpunkte der Karos als Koordinaten benutzt werden. Mittig eine Kombination aus Teppich und Blumentöpfen eines französischen Teams. Die wirkliche Position der Roboter, die sich auf Grund der Höhe der Roboter von der scheinbaren Position der Roboter unterscheidet kann direkt gemessen werden. Dadurch ist eine Berechnung oder Messung der Kamerahöhe nicht notwendig. Ob das den Aufwand rechtfertigt, soll hier nicht diskutiert werden. Rechts ein Test der FU-Fighters. Die Farbkreise haben gegenüber dem Quadratmuster den Vorteil, dass die Koordinaten leichter gefunden werden können.

Die ältere Methode der FU-Fighters benutzt eine biquadratische Abbildung, um beiden Ursachen der Kameraverzerrung entgegenzuwirken [Egorova, [2]2004]. Das Verfahren benutzt nur die weißen Linien auf dem Feld und einige leicht ermittelbare Parameter zur Bestimmung der Entzerrung.

Der Algorithmus benutzt die weißen Linien des Feldes für die Transformation, in dem er sie mit einem vordefinierten Modell des Spielfeldes vergleicht und anpasst. Das Verfahren besteht aus drei Schritten: Zuerst werden die weißen Linien auf dem Teppich erkannt und extrahiert, danach wird eine einfache Initialisierung der Anpassung auf das Modell vorgenommen und zum Schluss wird diese Anpassung mit konventionellen Optimierungsverfahren verfeinert. Zur Messung der erreichten Qualität wird die Korrelation zwischen dem theoretischen Modell und den transformierten weißen Linien benutzt (siehe Abbildung 4.13).

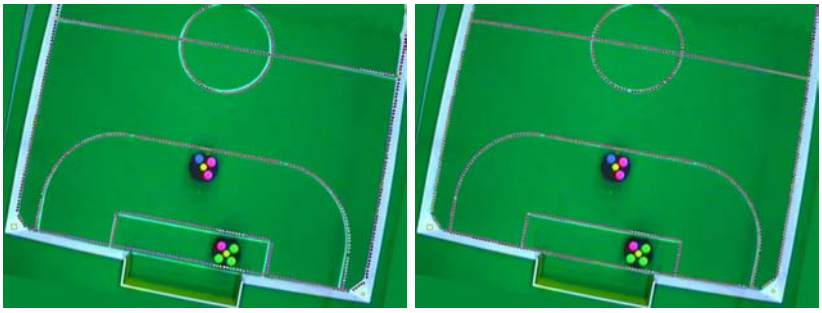

Abbildung 4.13: Links die Transformation nach dem ersten Schritt der Initialisierung. Rechts die Transformation nach der Optimierung der Parameter.

Wichtig ist bei diesem Verfahren, dass das theoretische Modell der Linien mit den Linien auf dem Feld übereinstimmt.

Das Verfahren arbeitet völlig automatisch und kann jederzeit neu durchgeführt werden, wenn sich die relative Position der Kamera gegenüber dem Spielfeld verschoben hat.

Später wurde eine neue Kameraentzerrung implementiert, die eine bessere Abbildungsfunktion erzeugt, weil sie die physikalischen Eigenschaften der Kamera berücksichtigt [Rojas, 2004]. Es wird zusätzlich eine radiale Entzerrung so vorgeschaltet, dass nur noch die projektive Transformation bestimmt werden muss.

In Abbildung 4.14 ist in der Mitte ein Bild zu sehen, welches aus gleichabständigen Kreisen besteht. Dieses Bild wird zur Berechnung radialen Entzerrung verwendet. Für die Approximation der Entzerrungsfunktion wird ein Polynom dritten Grades verwendet. Die Parameter des Polynoms werden so gewählt, dass die Kreise wieder als gleichabständig dargestellt werden können, wobei das Polynom den Abstand der Punkte vom Mittelpunkt aus verschiebt. Das linke Bild zeigt die Verzerrung durch die Kamera. Auf der rechten Seite ist das ent-

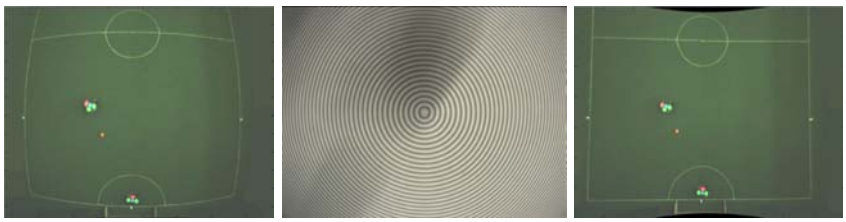

Abbildung 4.14: Die Radialentzerrung des Kamerabildes. Sie muss für jede Kamera-Objektiv-Kombination nur einmal durchgeführt werden. Links oben ist das Kamerabild zu sehen. Die Radialverzerrung wird mit Hilfe konzentrischer Kreise mit gleichabständigen Radien berechnet (Mitte). Rechts ist das Kamerabild entzerrt. Während eines Setups wird nur noch eine einfache Transformation anhand der Schnittpunkte der äußeren Begrenzungslinien berechnet um die extrinsischen Parameter zu berechnen. Eine biquadratische Interpolation ist jedoch auch möglich, um eine noch exaktere Abbildung zu erhalten.

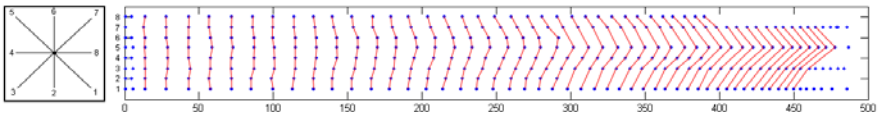

Abbildung 4.15: Die Abstände der konzentrischen Kreise aus Abbildung 4.14 wurden in acht verschiedenen Richtungen. Zur optimalen Berechnung der Entzerrungsfunktion müsste das Bild zentriert und orthogonal zur Kameraachse aufgenommen werden. Wie zu sehen ist, wurde das Bild nicht optimal ausgerichtet. Deshalb sollte zur Berechnung der Verzerrung in naher Zukunft ein anderes Muster genommen werden, zum Beispiel Linien oder Punktraster, damit die projektive Transformation gleichzeitig berücksichtigt werden kann.

zerrte Bild zu sehen. Bei der Kalibrierung der Entzerrungsfunktion ist auf ein genaues Ausrichten des Kreisbildes zu achten, da dies sonst zu einer falschen Abstandsfunktionen führen würde (siehe Abbildung 4.15).

Die nicht im Lot über dem Spielfeld hängende Kamera führt zu einer linearen Transformation. Die projektive Transformationsmatrix kann jedoch leicht anhand der vier Schnittpunkte der Begrenzungen Seitenaus-, Toraus-, und Mittellinie berechnet werden.

4.3 Verhaltenskontrolle

In den 1970er Jahren war es in der Robotik und der Künstlichen Intelligenz „state of the art“ hoch komplexe Softwaresysteme zu bauen, um Probleme „in-

telligent“ zu lösen. Man spricht heute von der deliberativen Verhaltenssteuerung oder vom Sense-Model-Plan-Act-Framework.

4.3.1 Subsumption Architektur

Mitte der 80er Jahre hat der damals noch unbekannte Wissenschaftler Rodney Brooks das Gegenteil proklamiert, was von etablierten Wissenschaftlern mit den Worten „Warum wirft der junge Mann seine Karriere weg?“ kommentiert wurde [Breuer, 2002]. Sehr einfache reaktive Verhalten steuern den Roboter auf einer niedrigen Ebene. Schon allein dies führt zu intelligent wirkenden Verhaltensweisen, wie Valentin Braitenberg mit seinen berühmten Braitenberg Vehikeln — die er nie selbst baute — zeigte [Braitenberg, 1984]. Brooks fügte den einfachen Verhalten sogenannte Kompetenzebenen hinzu, die, hierarchisch angeordnet, untergeordnete Verhalten beeinflussen [Brooks, 1985]. Die Verhalten auf höherer Ebene waren durchaus komplexerer Natur. Brooks selbst nannte die Struktur *Subsumption Architektur*. Er erhob das Paradigma dieser Architektur zu einer Art Philosophie des Handelns, um intelligente Systeme zu schaffen und nicht um über Intelligenz zu philosophieren [Brooks, 1991, Brooks, [2]1991].

4.3.2 Dual-Dynamics

Je komplexer das Gesamtsystem in einer Subsumption-Architektur jedoch ist, desto schwerer ist es auch zu verstehen und zu analysieren. Am Fraunhofer Institut für Autonome Intelligente Systeme wurde der *Dual-Dynamics*-Ansatz zur Steuerung von verhaltensbasierten Robotern entwickelt, bei dem sich die Verhalten durch Differenzialgleichungen beschreiben lassen [Jaeger, 1996]. Der Ansatz wurde in einer Programmier- und Simulations-Umgebung mit dem Namen DDSim eingesetzt [Jaeger, 1996].

Abbildung 4.16 zeigt den allgemeinen Aufbau einer Verhaltensebene der Dual-Dynamics-Architektur. Sie besteht aus einer Aktivierungsdynamik und einer Zieldynamik. Der Unterschied besteht darin, dass die Zieldynamik entscheidet, wie die Aktuatoren beeinflusst werden und die Aktivierungsdynamik entscheidet, ob dieser Einfluss ausgeführt wird oder nicht. Auf diese Weise werden die beiden Entscheidungen für jedes einzelne Verhalten voneinander getrennt: Einmal wird entschieden, ob das Verhalten die Umwelt durch die Aktuatoren beeinflussen darf und zweitens, wie es sie beeinflusst. Die Ebenen werden strikt getrennt. Die Aktivitäten höherer Ebenen werden zu Steuerungsparametern auf der niedrigeren Ebene. Auf der niedrigsten Ebene werden von den Verhalten die Aktuatoren in Abhängigkeit davon beeinflusst, wie stark deren Aktivierung ist [Jaeger, 1997].

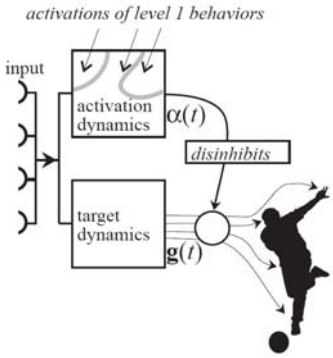

Abbildung 4.16: Der Dual-Dynamics-Ansatz von Jaeger. Jede Ebene des Verhaltens besteht aus einer Aktivierungsdynamik, die entscheidet, ob das Verhalten die Aktuatoren beeinflussen darf, und einer Zieldynamik, welche die Aktuatoren des Roboters tatsächlich beeinflusst. Abbildung aus [Bredenfeld, 1999]

4.3.3 Verwendete Verhaltensarchitektur

Das Verhaltenssystem der FU-Fighters baut auf dem Dual-Dynamics-Ansatz auf, hat jedoch weitere Strukturierungsmerkmale, die das System leichter analysierbar werden lässt [Behnke, [2]2000, Behnke, [3]2000]. Es steht auch eine Entwicklungsumgebung zur Verfügung, die es gestattet verhaltens-, sensor- und aktuator-basiert zu programmieren [Egorova, [3]2004]. Sie vereinfacht die Programmierung neuer Verhalten und veranschaulicht die Struktur der Verhaltenssteuerung.

Die besonderen Merkmale am Verhaltenssystem der FU-Fighters sind, dass strikt zwischen Sensoren, Aktuatoren und Verhalten unterschieden wird (siehe Abbildung 4.17). Die Sensoren, Aktuatoren und Verhalten sind in unterschiedlichen Ebenen organisiert. Das Ziel des Verhaltenssystems ist es, Agenten zu steuern. Jeder Roboter, aber auch das Team aller Roboter sind einzelne Agenten. Jeder Agent besitzt mehrere Ebenen, in denen Sensoren, Aktuatoren und Verhalten ihrer Komplexität nach geordnet sind. In der untersten Ebene befinden sich einfache Verhalten, die kleine Aufgaben bewältigen und sehr schnell zu berechnen sind, wie zum Beispiel das Verhalten „Fahren zum Ort“. Seine einzige Aufgabe ist es, den lokalen Bewegungsvektor des Roboters zu berechnen, um an einer vorgegebenen globalen Zielposition mit einer vorgegebenen Zielgeschwindigkeit anzukommen. In Kapitel 9 wird diese Verhalten durch ein Lernverfahren optimiert.

Höhere Ebenen beinhalten komplexere, langsamere und abstraktere Verhalten. Ein Beispiel wäre zum Beispiel die Pfadplanung, die auf einem hohen Abstraktionsniveau den Pfad für einen Roboter um Hindernisse berechnet und Zwi-

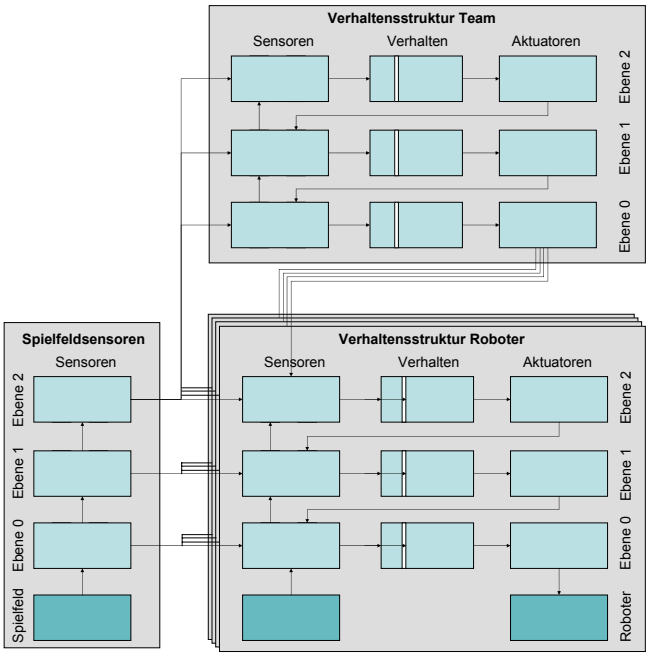

Abbildung 4.17: Der prinzipielle Aufbau der Verhaltenssteuerung der FU-Fighters. Die Kommunikation zwischen den Ebenen findet nur über Sensoren und Aktuatoren statt. Es existieren drei unterschiedliche Typen von Ebenen, um unterschiedliche Agenten zu steuern - Team-, Roboter- und Spiel-Ebenen.

schenpositionen als Aktuatorwunsch generiert. Die Ebenen mit der höchsten Komplexität sind die Ebenen des Teams aller Roboter. Dort werden Rollen zugewiesen, eine Taktik ausgewählt und auf Spielmodi (Elfmeter, Einwurf, usw.) reagiert.

Der dritte Typ Agent ist das so genannte Spiel. Dieser Agent hat keine Aktuatoren oder Verhalten, sondern nur Sensoren. Das Spiel liefert alle nötigen Informationen über den Verlauf des Spiels, die nicht roboterspezifisch sind, wie zum Beispiel die Ballposition.

Informationsfluss

Die Kommunikation zwischen den Ebenen findet nur über Sensoren und Aktuatoren statt. Die Ebenen sind in einer strikten Hierarchie angeordnet. Die Aktuatoren einer Ebene sind mit Sensoren der Ebene darunter verbunden. In der unteren Ebene werden die Sensoren der Ebene von den Verhalten benutzt, um die Aktuatorenwünsche für die Aktuatoren dieser Ebene zu berechnen. Da-

durch fließt die Information von den höchsten Ebenen zu den tiefsten Ebenen, die direkt mit dem Agenten und seinen physischen Aktuatoren verbunden sind.

Der Zusammenhang zu dem Dual-Dynamics-Ansatz besteht in der Aufteilung der Verhalten in eine Zielfunktion und eine Aktivierungsfunktion. Die Zielfunktion berechnet Aktuatorwünsche. Da mehrere Verhalten ein und denselben Aktuator ansprechen können, wird nach der Evaluierung aller Verhalten einer Ebene ein Aktuatorwert aus den Aktuatorwünschen berechnet. Die Berechnung erfolgt gewichtet mit den Aktivierungen der einzelnen Verhalten.

Information kann aber auch umgekehrt im System fließen: Direkt über so genannte aggregierte Sensoren. Sie sind nicht mit Aktuatoren verbunden, sondern aggregieren Sensorinformationen nidrigerer Ebenen. Auf diese Weise werden Informationen der physischen Sensoren nach oben gereicht, wobei die Informationen allerdings geglättet verarbeitet werden.

Neben den Sensoren, die mit den Aktuatoren der höheren Ebene verbunden sind, und den Sensoren, die Informationen von der niedrigeren Ebene aggregieren, gibt es noch Sensoren, deren Werte in Ebene selbst berechnet werden. Wenn, zum Beispiel, auf einer Roboter-Ebene die Information mehrfach benötigt wird, wie groß der kleinste Abstand zu einem gegnerischen Roboter ist, kann ein Sensor für diese Ebene geschrieben werden, der zu jedem Zeitschritt nur einmal berechnet werden muss.

Die Organisation der Verhalten in unterschiedlichen Ebenen hat neben der besseren Verständlichkeit des Codes noch einen weiteren Vorteil: Komplexere Verhalten sind zeitaufwändig zu berechnen, müssen aber nicht so oft berechnet werden, wie diejenigen, die sich in den untersten Ebenen befinden, da Entscheidungen höherer Ebenen länger bestand haben. So müssen die Aufgaben des Teams nicht 50 mal in der Sekunde neu verteilt werden. Deshalb ist es sehr sinnvoll, die unterschiedlichen Komplexitätsebenen in unterschiedlichen Zeitintervallen evaluieren zu lassen. So werden Verhalten der untersten Roboterebene in jedem Frame neu berechnet. Jede höhere Ebene wird halb so oft wie die darunter stehende Ebene evaluiert.

Hemmung

In jeder Ebene existieren viele Verhalten, die unter Umständen untereinander konkurrieren. Deshalb wurde das Prinzip der Hemmung eingeführt. Die Verhalten in jeder Ebene haben eine Prioritätshierarchie. Wenn viele Verhalten den Wunsch äußern, aktiv zu werden, um die Aktuatoren zu beeinflussen, einige Verhalten aber wichtiger sind als andere, dann wird auf diese Priorität zurückgegriffen. Die Aktivierung eines hemmenden Verhaltens hemmt entsprechend seiner eigenen Aktivierung die von diesem Verhalten gehemmten Verhalten, indem dessen Aktivierungswert heruntergesetzt wird. Zum Beispiel hemmt das Verhalten „hau rein“ alle anderen Verhalten auf der gleichen Ebene. Das Verhalten lässt einen Roboter nach vorne fahren, wenn er zum Tor ausgerichtet

ist und ein Ball von der Seite auf ihn zurollt. Dadurch kann reaktiv ein Tor aus einer zufälligen Situation geschossen werden, indem der Ball abgefangen und sofort weitergeschossen wird, ohne dass sich der Roboter ausrichtet. Andere Aufgaben, wie das Decken eines anderen Roboters, sind in dieser Situation unbedeutend.

4.3.4 Beispiel einer Verhaltenssequenz

In Abbildung 4.18 ist eine Spielsequenz zu sehen, in der verschiedene Verhalten so aktiviert werden, dass ein Zusammenspiel der Roboter entsteht. Dieses Zusammenspiel ist jedoch nicht explizit programmiert, sondern ergibt sich aus den Verhalten der einzelnen Roboter.

Die Sequenz zeigt einen Spielausschnitt, in dem zwei gelbe Roboter gegen drei blaue Roboter vor dem blauen Tor spielen. Die gelben Roboter versuchen, ins blaue Tor zu schießen, werden aber von einer Mauer zweier blauer Roboter daran gehindert. Das Verhalten des gelben Teams spielt sich in mehreren Ebenen ab: Die Teamebene des gelben Teams legt fest, welche Aufgaben die Roboter haben. Zum Beispiel wer von den Roboter zum Ball gehen soll, wer das Tor decken soll, wer Torwart ist, wer Angreifer oder Verteidiger sein soll. Im folgenden Beispiel ist der gelbe Roboter mit der Nummer 0 im Ballbesitz und der gelbe Roboter mit der Nummer 1 ist Angreifer und soll sich freistellen. Jeder Roboter betrachtet nun die Situation aus seiner eigenen Sicht, unter den einschränkenden Bedingungen der Teamvorgaben.

Betrachten wir die Roboterwelt zuerst aus der Sicht des Roboters im Ballbesitz. Auf der obersten Roboterebene trifft der Roboter die Entscheidung für die Auswahl des Verhaltens in einer Ebene weiter unten. Mögliche Verhalten sind „dribbeln“, „passen“, „schießen“. Da der Weg des Balls zum Tor nicht frei ist und kein Roboter als Anspielpartner zur Verfügung steht, wird das Verhalten „dribbeln“ aktiviert. Der Roboter versucht nun, die Mauer der beiden blauen Roboter zu umgehen und Richtung Tor zu dribbeln. Im dritten Bild (rechts oben) ändert sich nun die Situation: Der gelbe Roboter 1 hat seine Position durch das Verhalten „freistellen“ nach vorne verlagert. Im Teampartner Nummer 0 aktiviert sich das Verhalten „passen“, da er sieht, dass ein anderer Roboter gut positioniert ist. Er passt den Ball in Richtung des anderen Roboters. Da sich der Ball nun in Richtung des Passspielers bewegt, soll nun dieser „zum Ball“. Im gelben Roboter 0 aktiviert sich nun das Verhalten „freistellen“.

Sehen wir nun, wie die Welt aus der Sicht des gelben Roboters 1 aussieht. Er ist nicht in Ballbesitz und befindet sich auf einer ungünstigen Position. Das Feld ist in ein Raster von Positionen aufgeteilt, die jeweils anhand von Eigenschaften wie Anspielbarkeit, Eigentorgefahr, Torwinkel usw. bewertet werden. Das Verhalten „freistellen“ evaluiert alle möglichen Positionen auf dem Spielfeld und fährt maximal schnell zur am besten bewerteten Position. Im vierten Bild (links unten) sieht er nun, dass der Ball auf ihn zurollt. Dadurch wird das

Verhalten „abfangen“ aktiv und der Roboter positioniert sich möglichst genau zum Abfangen des Balls. Dazu gehört zum Beispiel eine leichte Bewegung nach hinten, um den Ball nicht abprallen zu lassen. Nachdem der Ball direkt vor dem Roboter ist und das Tor frei ist, aktiviert sich „schießen“. Dieses Verhalten hemmt alle anderen Verhalten und der Roboter schießt — erfolgreich — ins Tor.

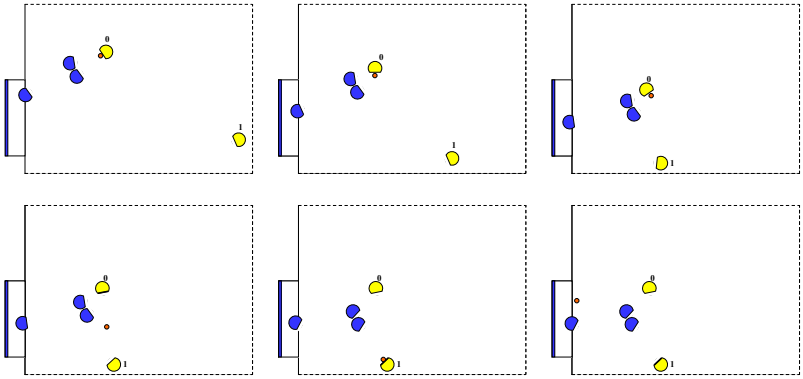

Abbildung 4.18: Eine Verhaltenssequenz von oben links zeilenweise nach unten rechts, die das Zusammenspiel der Roboter verdeutlicht.

4.4 Pfadplanung

Die aktuelle Pfadplanung arbeitet mit einer hoch optimierten Potentialfeldmethode. Durch die Vergrößerung des Spielfeld ist die Pfadplanung aber entweder zu ungenau, wenn das Raster zu grob gewählt wird, oder sehr zeitkritisch, wenn die Auflösung erhöht wird. Es ist daher erforderlich die Pfadplanung in nächster Zeit zu ersetzen.

Grob kann zwischen geometrischer Pfadplanung und Gitterpfadplanung unterscheiden werden [Lima, 2002]. Zur geometrischen Pfadplanung gehört die Pfadplanung mit Voronoy Diagrammen und Sichtbarkeitsgraphen. Zur Klasse der Gitterpfadplaner gehört die Potentailfeldmethode. Zelldekompositionsmethoden sind hybride Methoden.

4.4.1 Potenzialfeldmethode

Zur Pfadplanung wird das Spielfeld zur Zeit in 66×54 Koordinatenpunkte diskretisiert. Diese Koordinaten werden als Knoten aufgefasst, die mit ihren 8 Nachbarn verbunden sind. Für jeden Übergang von einem Knoten zu einem anderen Knoten entstehen Kosten. Die Kosten setzen sich aus Grundkosten,

Abständen zu anderen Objekten (Roboter, Ball, Wände, Strafräume usw.) und taktischen Kosten (zum Beispiel verursacht ein Pfad zwischen Ball und gegnerischem Tor höhere Kosten). Von der momentanen Position des Roboters zu der Zielposition wird dann mit dem A^*-Algorithmus der kostengünstigste Pfad gesucht. Auf dem berechneten Pfad werden dann Zwischenziele gewählt, durch die der Roboter mit einer bestimmten Geschwindigkeit fahren soll. Dabei können alle Parameter, zum Beispiel die Abstandsfunktion zu den Objekten, über das FU-Fighters-Framework eingestellt werden. In Abbildung 4.19 sind ein geplanter Pfad und einige einstellbare Parameter dargestellt.

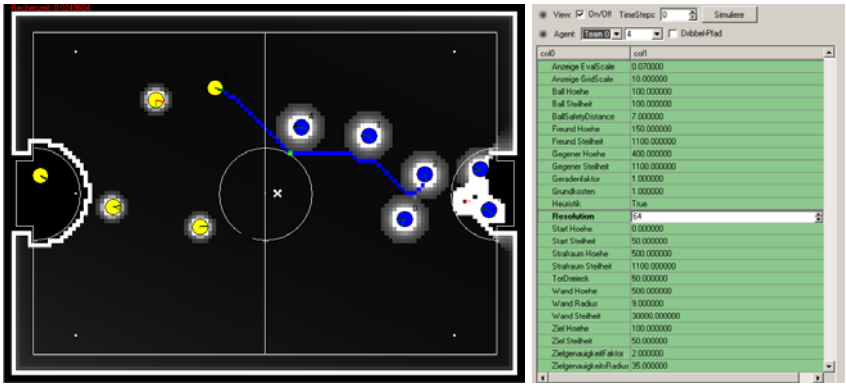

Abbildung 4.19: Pfadplanungsdarstellung im FU-Fighters-Framework. Auf der linken Seite ist das Gitter mit den Kosten der besuchten Knoten zu sehen. Der gelbe Roboter, bei dem der blaue Pfad beginnt, soll sich vor dem gegnerischen Tor postieren. Leider ist die Position besetzt, sodass der Pfad sehr teuer ist und fast das gesamte Spielfeld evaluiert werden muss, da die verwendete Schätzfunktion für den A^*-Algorithmus nicht perfekt ist. Der grüne Punkt auf dem Pfad ist die generierte Zwischenposition für den Roboter. Rechts befindet sich eine Tabelle mit den einstellbaren Parametern.

4.4.2 Geschwindigkeitssteigerung

Zwei Ansätze sind bei den FU-Fighters in der engeren Wahl, die aktuelle Pfadplanung zu ersetzen. Die eine Methode ist eine gitterbasierte Pfadplanung mit entfernungsabhängiger Maschengröße [Behnke, [2]2003]. Das Verfahren hat den Vorteil, dass der Algorithmus recht einfach ist. Der Nachteil ist allerdings, dass die Pfadplanung für jeden Roboter individuell durchgeführt werden muss. Die zweite Methode ist eine geometrische Pfadplanung, die schon als Prototyp in das FU-Fighters-Framework integriert wurde [Ourima, 2004]. Sie hat neben dem 10-fachen Geschwindigkeitsvorteil und der Skalierbarkeit den Vorteil, dass sie leicht zu einer Pfadplanung für mehrere Roboter erweitert werden kann und auch die eingeschränkte Bewegungsfreiheit eines fahrenden Roboters berücksichtigt.

4.5 Simulationsmodul

Das Simulationsmodul ersetzt das Bildverarbeitungssystem und das Kommunikationsmodul, es ist direkt mit dem Verhaltenssystem verbunden. Die Roboter-Positionen und -Orientierungen können interaktiv verschoben werden, der Ball lässt sich verschieben und mit unterschiedlicher Stärke schießen. Über Menüs können die Modi und Parameter für die Simulation eingestellt werden. Die Daten (Positionen, Orientierungen), die dem Verhaltenssystem übergeben werden, werden mit Rauschen versehen, um eine möglichst realistische Abbildung zu erreichen. Das Modul besteht aus einer Kollisions-Erkennung und -Behandlung für die Roboter ein einfaches, physikalisches Modell, welches die Trägheit der Roboter nachahmt. Für den Ball sind die Roboter Kreise, an denen er gebremst und reflektiert wird. Die physikalische Simulation kann außerdem durch gelernte Robotermodelle ersetzt werden. Dadurch ist eine flexible, schnell anpassbare und realistische Modellbildung der Roboter möglich. In Kapitel 6 wird näher auf dieses Thema eingegangen.

4.6 Verteilte Daten

Die Daten des Bildverarbeitungsmoduls können zur Visualisierung über das Verhalten auf entfernte Rechner über UDP/IP übertragen werden. Zur Zeit wird auch daran gearbeitet, das Modul so unabhängig zu machen, dass sie als Bildverarbeitungs-Server für zwei Teams und ggf. einen automatischen Schiedsrichter verwendet werden kann. In Abbildung 4.20 ist links das laufende FU-Fighters-Framework mit zwei gefundenen Robotern zu sehen und rechts die Positionen der beiden Roboter auf einem entfernten Rechner.

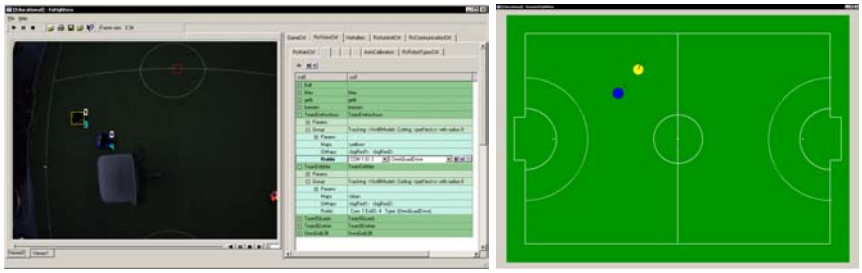

Abbildung 4.20: Links die Bildverarbeitung mit zwei beobachteten Robotern und rechts die Ansicht auf einem entfernten Rechner.

Es gibt auch ein Verhalten im FU-Fighters-Framework, welches eine TCP/IP-Verbindung mit einem entfernten Rechner aufbauen kann. Damit ist es möglich, die Roboter über das Netzwerk zu steuern. Eine Beispielanwendung hierfür ist

die Anbindung eines LOGO-Systems (UCBLogo[7] von der Berkeley University of California). Damit gibt es im modifizierten LOGO-System nicht nur eine virtuelle Schildkröte auf dem Bildschirm, sondern auch einen realen Roboter, der die Anweisungen — wie in der ursprünglichen Idee von LOGO[8] — auch ausführt.

4.7 Kommunikation

Die Elektronik ist so modular aufgebaut, dass beliebige Kommunikationsmodule (bis zu zwei Stück gleichzeitig) verwendet werden können. So ist es möglich, eine direkte (Draht-)Verbindung mit einem Roboter aufzubauen oder Funkmodule mit unterschiedlichen Übertragungsraten, Frequenzbereichen oder Kommunikationsschichten zu verwenden.

Die Daten können zu den Robotern entweder mit dem FU-Fighters-Framework — meist vom Verhalten gesteuert — oder mit einem speziellen Programm übertragen werden. Mit dem Programm ist es möglich, beliebig viele Roboter, zum Beispiel bei Vorführungen, ohne ein spezielles Setup fahren zu lassen. Es ist damit auch leicht möglich die Roboter schnell zu testen.

4.7.1 Vom PC zum Roboter

Für die Kommunikation werden zur Zeit die RX3/TX3 Module der Firma Radiometrix[9] eingesetzt [Radiometrix, 2001]. Sie arbeiten im Frequenzbereich von 869,85 und 914,5 MHz. Sie sind sehr zuverlässig und haben auch bei ungünstigen Verhältnissen mit vielen Sendern, wie zum Beispiel bei Wettbewerben, wenig Funkstörungen. Es können aber auch Bluetooth-Module oder SE200 Module von der Radio Communication Systems GmbH im 433MHz Band eingesetzt werden.

Die Bluetooth-Module werden zur Zeit nur zu Testzwecken eingesetzt, da sich eine Multicast-Kommunikation als sehr viel schwieriger erwiesen hat, als vorher angenommen. Außerdem werden die Bluetooth-Module bei kommenden Wettbewerben eventuell verboten, da sie andere Funk-Module, sowie WiFi-Netze mehr oder weniger stark stören.

Die SE200 Module sind auch sehr zuverlässig, sie haben aber eine geringere Übertragungsrate als die Radiometrix-Module. Allerdings bieten sie die Möglichkeit, als Sender oder als Empfänger zu arbeiten. Diese Eigenschaft wurde von den FU-Fighters jedoch nicht genutzt, da die Umschaltung etwas Zeit beansprucht und nur immer ein Sender aktiv sein darf, was eine ständige und

[7] www.cs.berkeley.edu/~bh/logo.html
[8] el.media.mit.edu/logo-foundation
[9] www.radiometrix.co.uk

schnelle Befehlsübermittlung vom Zentralrechner zu den Robotern unmöglich machen würde. Außerdem können die Module auf unterschiedlichen Frequenzen arbeiten.

4.7.2 Paketkodierung

Die Funkmodule sind sehr einfach aufgebaut. Sie übertragen die Bits der seriellen Schnittstelle auf einer Trägerfrequenz. Damit sich das Empfangsmodul mit dem Sender synchronisieren kann müssen genauso viele Nullen wie Einsen übertragen werden. Eine unkodierte Übertragung der Daten ist also nicht möglich, da die Daten in der Regel nicht in dieser Form ausgeglichen sind.

Mit den SE200 Funkmodulen wurden die Daten in einer Manchesterkodierung übertragen. Bei der Manchesterkodierung wird jeweils 1 Nibble (4 Bit) in 1 Byte (8 Bit) konvertiert, indem jedes Bit durch sein inverses wiederholt wird. Aus der Bitfolge 0010 wird also 01011001. Die Übertragungsrate halbiert sich demnach bei der Manchesterkodierung.

Seit dem Einsatz der RX3/TX3 Module wird eine spezielle 6b8b-Kodierung verwendet (siehe Abbildung 4.21), die von Radiometrix empfohlen wird [Radiometrix, [2]2001].

17	1B	1D	1E	27	2B	2D	2E	33	35	36	39	3A	3C	47	4B	4D
4E	53	55	56	59	5A	5C	63	65	66	69	6A	6C	71	72	74	78
87	8B	8D	8E	93	95	96	99	9A	9C	A3	A5	A6	A9	AA	AC	B1
B2	B4	B8	C3	C5	C6	C9	CA	CC	D1	D2	D4	D8	E1	E2	E4	E8

Abbildung 4.21: Die zulässigen Bytes in hexadezimaler Schreibweise, bei denen es genauso viele Einsen wie Nullen gibt. Die erlaubten Bytes 0F und F0 wurden ausgelassen, weil sie eine zu lange Folge aufeinander folgender Nullen und Einsen erzeugen würde. Die 64 Zahlen 17 bis D8 kodieren aufeinanderfolgend die 6 Bit Daten (grau eingerahmt). Die letzten 4 Zahlen E1, E2, E4, E8 können für unterschiedliche Paket-Header verwendet werden.

Die Befehle für die Roboter werden in Datenpaketen übertragen. Ein Datenpaket besteht dabei aus einer speziellen Bitfolge, bestehend aus einem Byte, das in der 6b8b-Kodierung sonst nicht vorkommt und den Beginn des Datenpakets markiert. Anschließend wird die Roboter ID übertragen, denn ein Paket ist immer nur für einen bestimmten Roboter vorgesehen. Da die ID aus 4 Bits besteht stehen noch 2 Bits zur Verfügung, die für die Auswahl von Schuss oder Dribbler zuständig ist. Zum Schluss werden 4 Bytes übertragen, welche die 3 Fahrwerte und die Dribbler- bzw. Schussgeschwindigkeit im Zahlbereich [-32,31] kodieren.

Neben den Datenpaketen gibt es noch Parameterpakete, über die 16Bit Werte und einem 128 Byte Speicherbereich auf der Roboterelektronik geschrieben werden können. Dadurch ist es zum Beispiel möglich, die PID-Parameter auf

einem Roboter während eines Spiels zu ändern. Dies wurde beim Lernen des PID-Reglers in Kapitel 9 benutzt.

Früher wurde dem Datenpaket auch noch eine Prüfsumme angehängt, um eine höhere Sicherheit der Richtigkeit der übertragenden Daten zu gewährleisten. Nach der Analyse der Datenpakete stellte sich aber heraus, dass alle Fehler schon durch die Dekodierung der 6b8b kodierten Daten herausgefiltert werden konnten. Es konnte also auf die Prüfsumme verzichtet werden. Die Erklärung ist, dass die Prüfsumme aus einer XOR-Verknüpfung der Daten bestand. Diese erkennt nur einzelne Bitfehler zuverlässig. Diese einfachen Fehler werden aber schon durch die 6b8b-Kodierung erkannt, denn die Anzahl der Nullen und Einsen sind bei Ein-Bit-Fehlern nicht mehr ausgeglichen. Andere Fehler treten sehr selten auf.

4.7.3 Vom Roboter zum PC

Zur Fernanalyse der Roboter kann zusätzlich ein DECT-Modul (Siemens MD32) auf die Elektronik gesetzt werden. Für die Zukunft ist geplant, Bluetooth-Module einzusetzen, damit auf einfachere Weise eine bidirektionale Kommunikation zwischen Roboter und Zentraleinheit möglich ist. Eine Kommunikation vom Roboter zum Zentralrechner ist sehr hilfreich, um die Regelkreise besser analysieren zu können und um innere Zustände — wie zum Beispiel die Batterieladung — auf dem Zentralrechner zur Verfügung stellen zu können.

Für die Kommunikation vom Roboter zum Zentralrechner wird kein spezielles Protokoll verwendet, da die Kommunikation über das DECT-Modul auf einer höheren Schicht erfolgt, in der die Daten transparent zur Verfügung stehen. Für den Zentralrechner wurde ein spezielles Programm geschrieben, mit dem die vom Roboter empfangenden Daten in Echtzeit visualisieren werden können und die Möglichkeit besteht, die Daten für eine spätere Analyse wahlweise in einem XML-Format oder im CSV-Format zu sichern und zu laden. Durch das CSV-Format ist es möglich, die Daten in anderen Programmen, wie zum Beispiel Matlab, weiterzuverarbeiten. In Abbildung 4.22 ist ein eine Bildschirmkopie des Programms zu sehen. Das Programm ist sehr flexibel gestaltet, sodass ohne Neuprogrammierung die unterschiedlichsten gesendeten Daten des Roboters analysiert werden können. Auch wenn sich durch Umprogrammierung des Roboters die Anzahl oder die Reihenfolge der gesendeten Daten ändert, muss das Programm nicht angepasst werden.

Die Flexibilität wird durch ein einfaches Protokoll erreicht. Jedes Datenpaket beginnt mit dem Wert -128. Anschließend können beliebig viele Bytes gesendet werden, die nur nicht dem Wert -128 entsprechen dürfen.

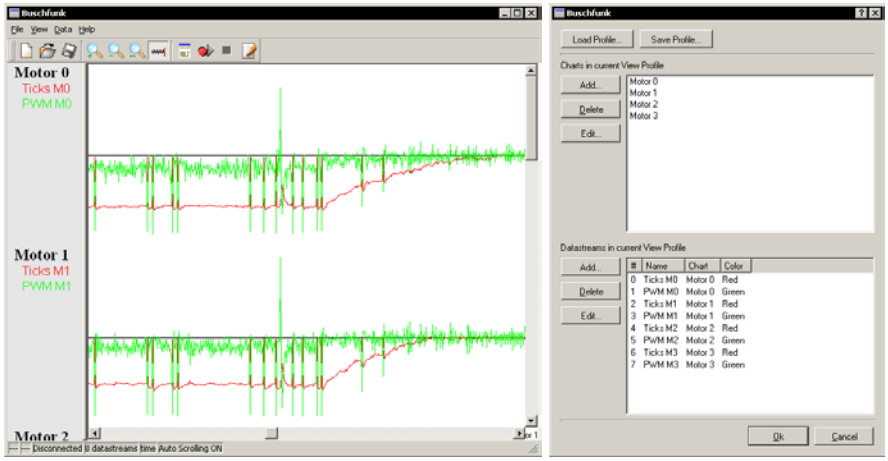

Abbildung 4.22: Das Programm zur Analyse der Roboter. Links die Darstellung der Kurven, rechts die Auswahl der Daten und eine Möglichkeit der Beschriftung.

4.8 Roboterelektronik

Die Elektronik auf dem Roboter besteht aus drei Komponenten: Der Hauptplatine mit Mikrocontroller und Motortreibern, einem Funkmodul und einer Schusselektronik.

Als Mikroconroller dient ein Motorola[10] HC12 (genauer MC68HC912DG128A). Es ist eine 16Bit MCU mit 128kByte Flash, 8kByte RAM und 2kByte EEPROM. Als Schnittstellen besitzt der Controller unter anderem zwei serielle Schittstellen, I^2C-Bus, CAN-Bus, 16 digitale I/0-Ports und zwei 10Bit Analog-Digital-Konverter [Freescale, 2003]. Die meisten Schnittstellen sind nach aussen geführt, wenn sie nicht direkt für die Motorsteuerung benötigt werden (siehe Abbildung 4.23).

Die Schusselektronik besteht aus einem DC/DC Step-Up-Regler, der die Spannung von 9,6 Volt auf 60 Volt erhöht und etwas Energie in einem Kondensator (27000μF) sammelt. Zusätzlich enthält die Schusselektronik eine Auslösungs- und Entladungs-Automatik. Der Magnetschuss wird über einen Digitalausgang des Mikrocontrollers ausgelöst. Über einen Digitaleingang der MCU ist außerdem eine Lichtschranke angeschlossen, um zu messen, ob sich der Ball genau vor dem Schussmechanismus befindet.

[10] www.freescale.com

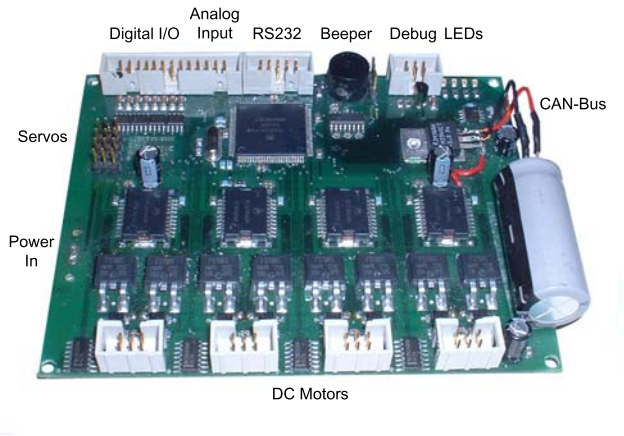

Abbildung 4.23: Die in der Small-Size 2004 verwendete Elektronik. Den größten Teil der Platine nehmen die Motortreiber im unteren Teil ein. Im oberen Bereich befindet sich der Mikrocontroller (HC12), dessen Spannungsversorgung, ein Pegelwandler für eine RS232 Schnittstelle, der Quarz und einige passive Bauelemente.

4.9 Roboterhardware

Die Roboter ändern sich jedes Jahr so gravierend, dass man von einer jährlichen Neuentwicklung sprechen kann. In Abbildung 4.24 ist ein FU-Fighters Small-Size-Roboter 2004 zu sehen. Gegenüber dem Vorjahresmodell verfügt er über einen Hochschuss (die Kupferspule an der Vorderseite des Roboters), über einen direkten linearen Schuss (zu erkennen an der roten Ummantelung an der Unterseite des Roboters), über vier statt drei omnidirektionale Räder, ein kürzer untersetztes Getriebe (Faulhaber 20/1 mit 14:1 statt 22 mit 19:1), einen stärkeren Kondensator für die beiden Schüsse (27000μF statt 10000μF) und eine neue Hauptplatine, die den Strom für die Motoren nicht mehr auf 1A beschränkt. Die Motoren (Faulhaber 2224, 6V mit IE2/64 Impulsgeber statt IE2/16) und Akkumulatoren (6 x Sanyo HR 4/5SC 2100mAh á 1.2V statt 2000mAh) sind nahezu gleich geblieben. Außerdem sind die Akkumulatoren jetzt schnell auswechselbar, so dass sie in der Halbzeitpause gewechselt werden können.

4.10 Mikrocontrollersoftware

Die Software auf dem Mikrocontroller ist recht übersichtlich. Sie setzt sich aus den Teilen (a) Kommunikation mit Paketdekodierung, (b) PID Controller und (c) Modellumrechnung zur Motorsteuerung mit Fehlerberechnung zusammen.

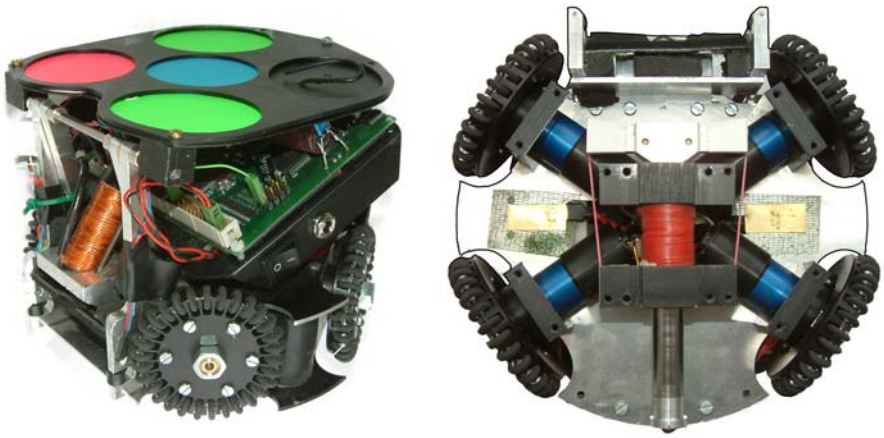

Abbildung 4.24: Der FU-Fighters Small-Size-Roboter von 2004. Links in der Seitenansicht. Es sind der Hochschuss, die Elektronik, ein Akku und die omnidirektionalen Räder deutlich zu erkennen. Rechts der Roboter von unten. Die Akkus sind herausgenommen, es sind jedoch die Kontakte für den Akku zu sehen. Außerdem ist der Flachschuss sowie die Motor-Getriebe-Kombinationen mit angeschraubten Rädern zu erkennen.

Die erste Komponente wird in einem eigenen Task unabhängig von den letzten beiden Komponenten ausgeführt. Der Roboter empfängt vom externen Rechner alle Pakete. Die Pakete werden dekodiert und es wird überprüft, ob das Paket zur Roboter-ID passt und das Paket vollständig und ohne Fehler angekommen ist (siehe Abschnitt 4.7). Wenn ja, dann werden die Steuerungsbefehle für die Zielgeschwindigkeiten (v_x, v_y, ω) für die Bewegung im lokalen Koordinatensystem der drei Freiheitsgrade vorwärts, seitwärts, und drehen und die Befehle für Hochschuss, Flachschuss, Dribbler und deren Schussstärke in den gemeinsamen Speicher für die weitere Verarbeitung geschrieben. Dies geschieht ohne Locking-Mechanismus, denn auch wenn nicht alle Daten aktuell sind, sind die Befehle zwar falsch aber sie liegen sozusagen auf dem Weg zwischen alten und neuen Daten und nach 8ms werden nur noch die neuen Daten berücksichtigt.

Die Komponenten b und c werden in einer Systemschleife alle 8ms ausgeführt (siehe Abbildung 9.1). Die Modellumrechnung führt in eine Richtung eine Konvertierung der Fahrgeschwindigkeiten in Radgeschwindigkeiten bzw. PWM-Werte für die Motoren durch. In die andere Richtung führt sie eine Konvertierung der Daten der Impulsgeber der Motoren in Fahrgeschwindigkeiten des Roboters durch. Der PID-Controller vergleicht die ausgeführten Geschwindigkeiten des Roboters mit den gewünschten Fahrgeschwindigkeiten und korrigiert die Befehle entsprechend.

4.10.1 Roboterkoordinaten und Radgeschwindigkeiten

Die Umrechnung von lokalen Robotergeschwindigkeiten (v_x, v_y, ω) in Radgeschwindigkeiten w_i für n Räder geschieht nach folgender Gleichung:

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \frac{1}{r} \begin{pmatrix} x_1 & y_1 & b \\ x_2 & y_2 & b \\ \vdots & \vdots & \vdots \\ x_n & y_n & b \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ \omega \end{pmatrix}. \tag{4.1}$$

Die Variable r gibt den Raddurchmesser an, b ist der Abstand vom Rotationszentrum des Roboters zu den Rädern und $F_i = (x_i, y_i)$ ist der Kraftvektor des i-ten Rads. Zur Vereinfachung wird das Rotationszentrum in der Mitte des Roboters angenommen, in der sich die Motorachsen des Roboters schneiden und die Motoren alle den gleichen Abstand zum Rotationszentrum haben. Der Spezialfall, in dem alle Räder wie im Palm Pilot Robot Kit[11] einen Winkel von 120^o haben, kann einfach ausgerechnet werden [Reshko, 2002]. Der erste omnidirektionale Roboter der FU-Fighters hatte auch diese Radanordnung [Rojas, 2002, Gloye, 2003].

Die Rückrechnung von den Radgeschwindigkeiten in Robotergeschwindigkeiten kann durch Umformung von Gleichung 4.1 und der Berechnung der Pseudoinverse der Transformationsmatrix erfolgen:

$$\begin{pmatrix} v_x \\ v_y \\ \omega \end{pmatrix} = r \begin{pmatrix} x_1 & y_1 & b \\ x_2 & y_2 & b \\ \vdots & \vdots & \vdots \\ x_n & y_n & b \end{pmatrix}^{+} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}. \tag{4.2}$$

Dadurch werden n Motorwerte in 3 Bewegungsvektoren linear umgerechnet. Wenn es mehr als 3 Räder gibt, dann ist die Gleichung überbestimmt und dies führt gleichzeitig zur Fehlerkorrektur der Impulswerte der Motoren, wenn der Fehler nicht allzu groß ist. Ist der Fehler jedoch groß, kann der gegenteilige Effekt eintreten und die umgerechneten Robotergeschwindigkeiten wertlos sein. Wir werden auf dieses Problem in Kapitel 8 zurückkommen.

Die Kraft des Motors hängt von der Spannung und der Geschwindigkeit ab [Kalmár-Nagy, 2004]. Es gilt $f = \alpha U - \beta v$, wobei f die Kraft, U die Spannung und v die Geschwindigkeit ist. Die Konstanten α und β geben die charakteristischen Eigenschaften des Motors wieder. Wenn die Spannung konstant ist und die Motoren bei einem kleinen Winkel zwischen den Rädern viel Kraft aufwenden müssen, um gegeneinander zu arbeiten, dann sinkt automatisch die Geschwindigkeit. Im Folgenden wollen wir diesen Sachverhalt allerdings ignorieren.

[11] www-2.cs.cmu.edu/~reshko/PILOT

Das Modell zur Robotergeschwindigkeit bezüglich der Radgeschwindigkeiten ist sehr einfach. Es berücksichtigt nur die Geschwindigkeiten der Motoren. Wenn wir annehmen, dass die maximalen Geschwindigkeiten der Motoren konstant sind und nicht von der Kraft bzw. dem Drehmoment abhängen, dann können wir mit Gleichung 4.2 die maximale Geschwindigkeit des Roboters berechnen. Dabei ist die maximale Geschwindigkeit abhängig von der Richtung, in der der Roboter fahren soll. Wie in Abbildung 4.25 zu sehen ist, ist der Roboter in Vorwärts- und Rückwärtsrichtung (in den Graphen rechts und links) schneller, als in der Seitwärtsrichtung (in den Graphen oben und unten). Bei den spezifischen Robotern liegt dies daran, dass die Geschwindigkeit in der Seitwärtsrichtung von dem hinteren Rad beschränkt wird.

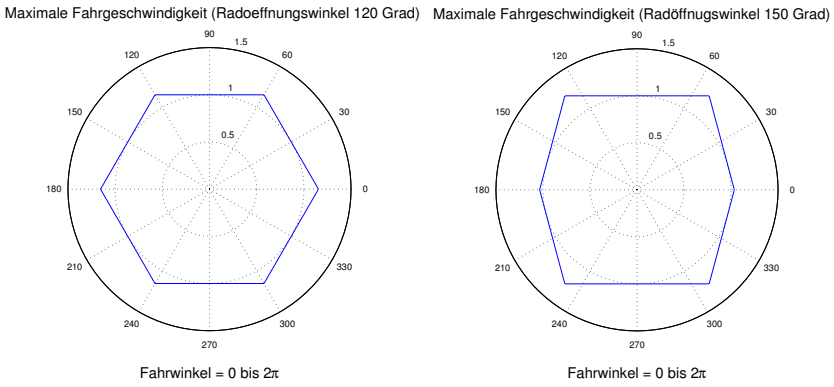

Abbildung 4.25: Die theoretische Maximalgeschwindigkeit der Roboter in Abhängigkeit der Fahrtrichtung unter der Annahme, dass die Maximalgeschwindigkeit der Motoren unabhängig von der aufzuwendenden Kraft ist. Links ist die maximale Geschwindigkeit eines dreirädrigen Roboters mit gleichem Winkel zwischen den Rädern gezeigt. Rechts für einen Roboter mit einem vorderen Öffnungswinkel von 150^o. Bei beiden Robotern ist ein Rad hinten quer zur Fahrtrichtung angebracht.

Für einen vierrädrigen Roboter, bei dem die Räder vorne und hinten einen größeren Öffnungswinkel als seitlich haben, würde der Roboter seitlich schneller fahren als vorwärts. In der Realität ist es jedoch genau umgekehrt! Bei einer seitlichen Fahrt müssen die Motoren so viel Kraft aufwenden, da sie Räder gegeneinander arbeiten, dass eine viel geringere Endgeschwindigkeit erreicht werden kann.

4.10.2 PID-Controller

Im Regelkreis gibt es drei PID-Controller. Jeweils einen für die Vorwärtsbewegung v_y und Seitwärtsbewegung v_x des Roboters und eine für die Drehung ω des

Roboters um die eigene Achse. Wenn e_y die Differenz zwischen dem Sollwert und dem Istwert der Vorwärtsgeschwindigkeit zum Zeitpunkt t ist, dann wird der Steuerwert s_y für die Vorwärtsgeschwindigkeit durch

$$s_y(t) = P_y e_y(t) + I_y(\sum_{k=0}^{\ell} e_y(t-k)) + D_y(e_y(t) - e_y(t-1)) \tag{4.3}$$

berechnet, wobei P_y,I_y und D_y die Konstanten für den proportionalen, integralen und differenziellen Anteil eines PID Reglers sind. Bevor die Steuerwerte an die PWM-Regler weitergegeben werden, wird noch eine Modifikation durchgeführt: Je nach Vorzeichen des Steuerwerts wird ein Offset C_y addiert bzw. subtrahiert, die Ausgabe mit einem Skalierungsfaktor S_y multipliziert und die Änderung des Steuerwerts gegenüber dem vorherigen Steuerwert durch eine Beschleunigungsbegrenzung A_y abgeschnitten.

Der Proportionalterm sorgt dafür, dass der Fehler schnell korrigiert wird. Der Integralterm akkumuliert einen Fehler über längere Zeit und sorgt damit für den Ausgleich kleiner aber dauerhafter Abweichungen. Der Differenzialterm sorgt für eine Dämpfung der Korrektur, um ein starkes oszillieren zu verhindern. Der Offset dient dazu die Startschwelle, an der die Motoren beginnen sich zu drehen, herabzusetzen. Der Skalierungsfaktor dient dazu den Bereich der PWM-Steuerwerte auf den Bereich der Motorticks auf eine einfache Art abzubilden. Die Beschleunigungsbegrenzung dient dazu das Durchdrehen der Räder zu verringern.

Die Berechnung der Steuerwerte für die Rotation und Seitwärtsgeschwindigkeit ist äquivalent.

In Kapitel 9 komme ich noch einmal auf den PID-Controller zurück. Dort wird eine Methode vorgestellt, wie die Parameter des Controllers optimal eingestellt werden können.

Im nächsten Kapitel wird gezeigt, wie das System erweitert wurde, um die Dynamik des Roboters zu lernen. Dadurch kann das Verhalten des Roboters vorhergesagt werden. Aus dem Wissen über zukünftige Positionen und Orientierungen der Roboter und des Balls kann das Delay des Systems zu kompensieren.