UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

Diplomarbeit

Autonome Wegplanung mittels lokaler und globaler Sensorik für mobile Roboter

Sven Bursch
16. Oktober 2006

INTERNE BERICHTE
INTERNAL REPORTS

Diplomarbeit am
Fachbereich Informatik
der Universität Dortmund
Lehrstuhl Informatik I

Betreuer:
Dr. Lars Hildebrand
Prof. Dr. Bernd Reusch

Inhaltsverzeichnis

Abbildungsverzeichnis

Erklärungen

Hiermit erkläre ich, dass ich diese Diplomarbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Ich erkläre mich einverstanden, dass meine Diplomarbeit nach §6 (1) des URG der Öffentlichkeit durch die Übernahme in die Bereichsbibliothek zugänglich gemacht wird. Damit können Leser der Bibliothek die Arbeit einsehen und zu persönlichen wissenschaftlichen Zwecken Kopien aus dieser Arbeit anfertigen. Weitere Urheberrechte werden nicht berührt.

Dortmund, den 16. Oktober 2006 _____

Danksagung

An dieser Stelle möchte ich mich bei allen Leuten bedanken, die mir in der Zeit der Diplomarbeit geholfen haben. Insbesondere bei:

Simon Schulz

Ohne seine Hilfe als Roboter-Doktor hätte ich diese Arbeit aufgrund von Hardwareproblemen sicherlich nicht in dieser Form vollenden können. Seine Hilfe rechne ich ihm besonders an, weil er zeitgleich mit dem Schreiben seiner Diplomarbeit ([Sch06]) beschäftigt war.

Simone Osewold

Sie hat mit mir die Höhen und Tiefen dieser Arbeit durchlebt. Obwohl es sicherlich nicht immer leicht gewesen ist, stand sie mir die ganze Zeit über bei.

Tobias Jäger und Christian Kintzel

Diese wackeren Recken haben es auf sich genommen diese Arbeit Korrektur zu lesen. Sollte jetzt noch ein Fehler in dieser Arbeit sein, so hat er sich diese Position mühsam erkämpft.

Dr. rer.-nat. Lars Hildebrand

Betreuer meiner Diplomarbeit, der immer ein offenes Ohr bei Problemen und Fragen hatte.

Teil I

Grundlagen

1 Einleitung

Die drei Robotergesetze aus [Asi86]:

1. Ein Roboter darf kein menschliches Wesen verletzen oder durch Untätigkeit gestatten, dass einem menschlichen Wesen Schaden zugefügt wird.
2. Ein Roboter muss den ihm von einem Menschen gegebenen Befehlen gehorchen - es sei denn, ein solcher Befehl würde mit Regel Eins kollidieren.
3. Ein Roboter muss seine Existenz beschützen, solange dieser Schutz nicht mit Regel Eins oder Zwei kollidiert.

1.1 Überblick und Motivation

Bei dem Wort Roboter denken viele Menschen entweder an Industrieroboter, z.B. aus der Automobilherstellung, oder an Robotern aus Science-Fiction Filmen. Den wenigsten Menschen ist bewusst, dass Roboter aus unserem Leben nicht mehr wegzudenken sind. Roboter waschen – in Form einer Waschstraße – unsere Autos, Roboter untersuchen die Kanalisation nach Defekten, erforschen fremde Planeten oder kämpfen in Kriegen. Zunehmend steigt die Zahl an Servicerobotern, die in unseren Wohnungen agieren. Roboter, die staubsaugen und den Rasen mähen sind nur zwei Vertreter dieser Klasse.

Die ersten Roboter wiederholten monotone Bewegungsabläufe, die einmal einprogrammiert wurden. An autonome Roboter werden andere Anforderungen gestellt. Autonome Roboter müssen ihre Umgebung mit Sensoren wahrnehmen, die zum Teil ungenaue Daten liefern. Mit Hilfe der Messungen dieser Sensoren muss sich der Roboter ein Bild von seiner Umgebung machen und dann entsprechend seiner Aufgabe agieren. Roboter, die über das hinaus auch noch mobil sein sollen, müssen außerdem in der Lage sein, von einem Ort zu einem anderen zu fahren, ohne mit Hindernissen zu kollidieren.

Während es für einen Mensch in der Regel kein Problem darstellt, seine Umgebung wahrzunehmen und ohne Kollisionen von einem Punkt zu einem anderen zu gelangen, ist diese Anforderung für Roboter noch immer eine Herausforderung. Diese Herausforderung wird um so größer, je weniger Sensorinformationen, Rechenleistung und Speicher zur Verfügung stehen. An diesem Punkt setzt diese Arbeit an.

1.2 Aufgabenstellung

Schon seit mehreren Jahren beschäftigt sich der Lehrstuhl Informatik I der Universität Dortmund mit teilautonomen Robotern im Rahmen des Roboterfußball-Projektes. In der MiroSot-Liga der FIRA-Organisation (s. [Fir06b] und [Fir06a]) kommen fahrende

Roboter zum Einsatz, die über zwei angetriebe Räder verfügen. Sie haben eine kubische Form mit einer Kantenlänge von maximal 7,5 cm. Abbildung 1.1 zeigt einen der am

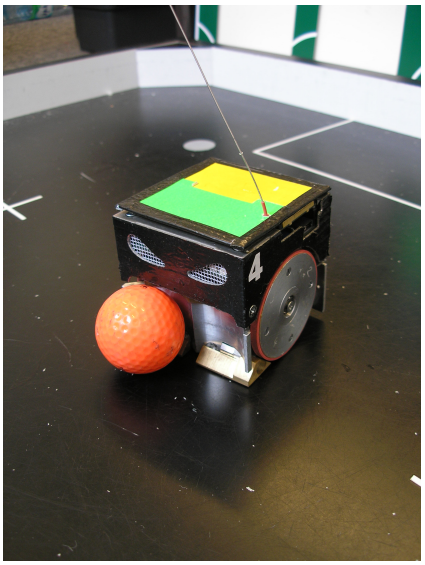

Abbildung 1.1: Ein Roboter der FIRA MiroSot-Liga

Lehrstuhl Informatik I zur Verfügung stehenden Robotern aus dieser Liga.

Bislang werden die Roboter im Rahmen des Roboterfußballs zentral von einem Hostsystem gesteuert, das die aktuelle Umgebung über eine unter die Decke montierten Kamera wahrnimmt. Nach der Bearbeitung der aufgenommenen Bilder durch die Bildverarbeitung der Strategieberechnung werden an den Roboter per Funk Befehle der Art „Drehe dich um x Grad" oder „Fahre mit einer Geschwindigkeit von y" gesendet. Der Grund für diese einfachen Befehle liegt in dem verwendeten DSP. Bislang kam von Texas Instruments der TMS320F240 zum Einsatz, der über eine maximale Leistung von 20 MIPS (Million instructions per second) verfügt und keine ausreichenden Leistungsreserven hat, um komplexere Befehle auszuführen (s. [Klu01], [Moh01]).

Seit dem Ende der Projektgruppe 449 (s. [Pro05]) und den darauf folgenden Arbeiten der studentischen Hilfskräfte, steht jedoch ein Prototyp eines wesentlich leistungsfähigeren Roboters zur Verfügung. Bei diesem Roboter kommt ein TMS320F2812 zum Einsatz, der eine maximale Leistung von 150 MIPS hat, was die Möglichkeit eröffnet, dass der Roboter wesentlich mehr Aufgaben übernehmen kann.

Im Rahmen dieser Diplomarbeit soll die Autonomie des Roboters gesteigert werden. Hierzu soll untersucht werden, inwieweit der Roboter selbstständig die Umgebung durch Sensoren wahrnehmen und diese Informationen zu einem Weltmodell verarbeiten kann. Außerdem soll untersucht werden, ob der Roboter selbstständig auf Basis des erstellten Weltmodell von einer zu einer anderen Position fahren kann.

1.3 Aufbau der Arbeit

Diese Arbeit gliedert sich in die Teile Grundlagen und Realisierung.

In den Grundlagen werden Mobile Roboter (Kapitel 2) und die Sensoren, die im Bereich der Mobilen Robotern genutzt werden (Kapitel 3), vorgestellt. Anschließend werden Verfahren, die Informationen von Sensoren zu einem Weltmodell zusammenführen (Kapitel 4), und Algorithmen zur Wegplanung (Kapitel 5) vorgestellt. Im letzten Kapitel des

ersten Teils wird die am Lehrstuhl Informatik I vorhandene Testumgebung für Mobile Roboter beschrieben (Kapitel 6).

In zweiten Teil wird die Realisierung der Aufgabenstellung beschrieben. Kapitel 7 und Kapitel 8 geben einen Überblick für die Implementierung auf dem Roboter und auf dem Hostsystem. Die weiteren Kapitel gehen auf Teilaspekte der Implementierung näher ein. In Kapitel 9 werden die genutzten Sensoren evaluiert und in Kapitel 10 wird der zur Sensorfusion genutzte Algorithmus erläutert. Kapitel 11 zeigt, wie der Roboter die Wegplanung realisiert.

Kapitel 12 beschließt diese Arbeit mit einem Fazit und einem Ausblick.

1.4 Einordnung der Arbeit

Im Bereich des Roboterfußballes, speziell in den Ligen der RoboCup-Organisation (s. [Rob04], [Rob05] und [Rob06b]), sind bereits autonome, mobile Roboter verbreitet. Die meisten autonome Roboter sind fahrende Roboter, die eine maximale Größe von 50 cm pro Seite haben dürfen, oder in Form der von der Firma Sony entwickelten Aibo[1]-Hunde. Während in den Robotern oftmals normale Laptops zum Einsatz kommen, verfügen die Aibos in der neusten Generation über Prozessoren mit 576 MHz und 64 MiB[2] Speicher.

In beiden Fällen ist mehr Rechenleistung als auch mehr Speicher vorhanden, als bei den Robotern, die für diese Arbeit zur Verfügung stehen. Während bei vielen Arbeiten zum Thema „autonome, mobile Roboter" die zur Verfügung stehenden Ressourcen nur eine zweitrangige Rolle spielen, sind diese in dieser Arbeit von zentraler Bedeutung.

[1] *A*rtificial *I*ntelligence ro*BO*t

[2] In dieser Arbeit werden die Präfixe gemäß Norm 60027-2 der *International Electrotechnical Commission* genutzt. Während die Standard-SI-Präfixe (k, M, G) laut Norm für 10^x stehen, werden sie in der Informatik für 2^{x*10} genutzt. Um zukünftig Missverständnise zu vermeiden, wurden die Präfixe „Ki", „Mi", „Gi" eingeführt, die für 2^{x*10} stehen.

2 Mobile Roboter

> „Roboter (zu tschechisch robota Fronarbeit) der, von K. Capek 1920 geprägte Bezeichnung für einen künstlichen Menschen, eine Puppe, die Bewegungen scheinbar selbstständig ausführt, z. B. aufgrund drahtlos übermittelter Befehle. [...]
> Im allgemeinen Sprachgebrauch wird heute als Roboter jedes automatische System mit bestimmten sensorischen und adaptiven Eigenschaften zur Ausführung manipulatorischer und ortsveränderlicher Vorgänge verstanden." [Mey06]
>
> „mobil (lateinisch-französisch beweglich), beweglich, nicht an einen festen Standort gebunden; [...]. „ [Mey06]

Roboter sind aus unserem Leben nicht mehr wegzudenken und die Gründe für ihren Einsatz sind vielfältig. Erkundungsroboter werden überall dort eingesetzt, wo es für einen Menschen zu gefährlich bzw. unmöglich wäre zu agieren. Beispiele sind Roboter, die in kontaminierten Gebieten eingesetzt werden (z.B. Kernkraftwerk Tschernobyl) oder die Planeten erforschen (z.B. Luod 1, Sojourner, Spirit/Opportunity). Serviceroboter helfen Menschen im täglichen Leben (Rasen mähen, Staub saugen oder Servier-Aufgaben). Vom Militär werden Roboter zur Aufklärung, Verteidigung und zum Angriff eingesetzt (z.B. Global Hawk, Tomahawk). In der Industrie werden Schweißroboter oder Transportroboter eingesetzt.

Je nach Einsatzgebiet bewegen sich die Roboter mit unterschiedlichen Methoden fort (vgl. [DM00]). Roboter, die sich auf dem Boden fortbewegen, sind oftmals mit Rädern, Kettenantrieben oder mit beinartigen Elementen ausgestattet. Roboter, die auf oder unter Wasser agieren, sind, wie Schiffe und U-Boote auch, oftmals mit Schiffsschrauben ausgestattet. Fliegende Roboter bewegen sich wie Flugzeuge oder Hubstauber mit Propellern oder Düsenantrieben fort.

Unabhängig von dem Aufgabenbereich und Fortbewegungsart können die für einen mobilen (beweglichen) Roboter notwendigen Komponenten nach [DM00, S. 15] in vier Bereiche unterteilt werden.

- Bewegung
  Der Roboter muss sich innerhalb seines Einsatzbereiches fortbewegen können.
- Wahrnehmung
  Der Roboter muss die Umgebung wahrnehmen können, um ein Modell seiner Umgebung zu generieren.
- Handeln
  Der Roboter muss Aktionen ausführen können, um die festgelegten Aufgaben zu erfüllen.

- Kommunikation
  Der Roboter muss über eine Kommunikationmöglichkeit mit der Umwelt verfügen, sei es, um über seinen Zustand Auskunft zu geben oder um neue Informationen zu erhalten.

Diese Unterteilung wird genutzt, um diese Arbeit zu gliedern.

In Hinblick auf die Aufgabenstellung und die Testumgebung, die in Kapitel 6 vorgestellt wird, beschränkt sich diese Arbeit auf differentielle zweiräderige Roboter, die sich auf dem Boden fortbewegen. Diese Art von Robotern ist mit zwei angetriebenen Rä-

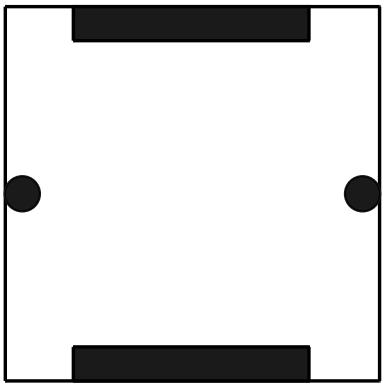

Abbildung 2.1: Schematische Darstellung eines differentiellen zweiräderigen Roboters

dern ausgestattet und, um die Balance zu halten, mit zwei Schleifern. Abbildung 2.1 zeigt schematisch, wie seitlich die Räder sowie vorne und hinten die Schleifer angebracht sind. Drehungen des Roboters sind durch unterschiedliche Radgeschwindigkeiten möglich. Wenn sich die Räder gleichmäßig entgegengesetzt drehen, ermöglichst dieses eine Drehung auf der Stelle.

2.1 Modell

Um die Bewegung eines Roboters planen zu können, ist es notwendig zu wissen, wie sich der Roboter verhält. Hierzu wird das Verhalten des Roboter mit Hilfe von Modellen beschrieben. Um die Komplexität der Modelle zu reduzieren, wird die Realität vereinfacht abgebildet. Für alle hier vorgestellten Modelle gilt, dass jedes Rad in genau einem Punkt den Boden berührt und immer Kontakt zum Boden hat. Effekte wie Traktionsverlust, das Rutschen des Roboters oder äußere Einflüsse, wie Kollisionen, werden nicht berücksichtigt. Die vorgestellten Modelle basieren auf [DM00], [SN04] und [Pro03].

Bevor verschiedene Modelle hergeleitet werden können, muss die Lage bzw. die Konfiguration des Roboters beschreibbar sein. Im einfachsten Fall ist die Angabe der Position und der Orientierung des Roboters im Raum hierfür ausreichend. Die Konfiguration k eines Roboter, der auf einer Ebene agiert, ist durch den folgenden Tupel beschreibbar:

$$k_1 \quad = \quad (x, y, \theta) \tag{2.1}$$

Hierbei ist x die X-Position, y die Y-Position und θ die Ausrichtung des Roboters bezüglich der X-Achse. Abbildung 2.2 zeigt das genutzte Koordinatensystem. Die Menge

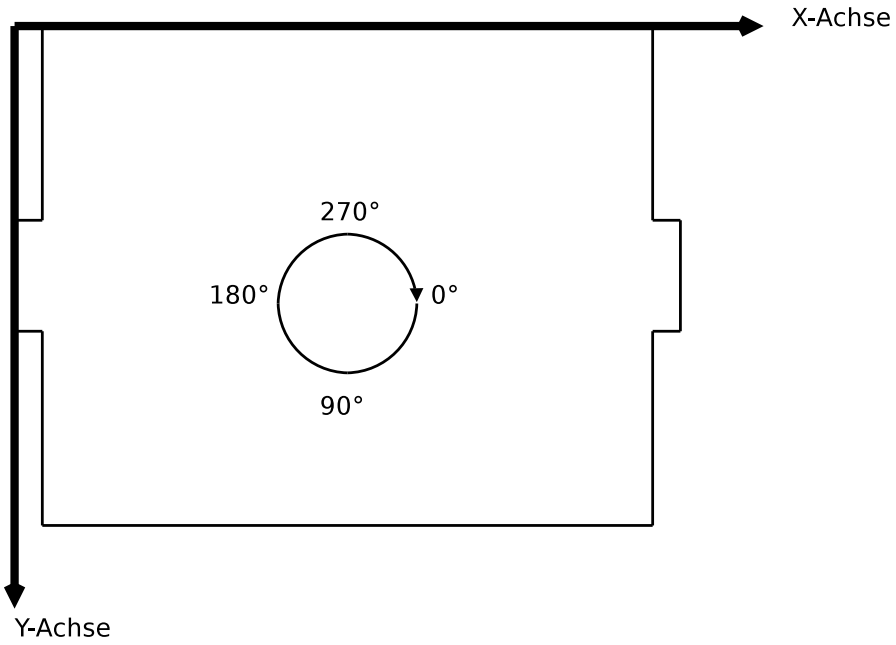

Abbildung 2.2: Genutztes Koordinatensystem

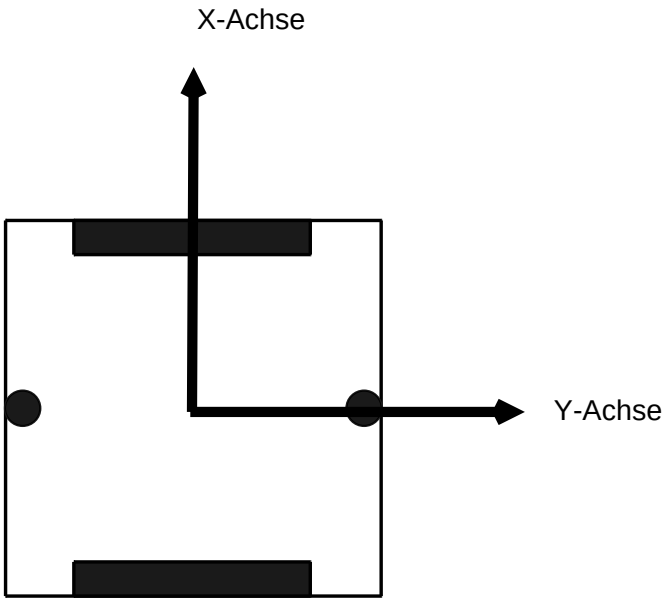

Abbildung 2.3: Bezugssystem des Roboters

aller möglichen Konfigurationen ergibt den Konfigurationsraum des Roboters. Ein komplexeres Modell wird zusätzlich die Geschwindigkeit der beiden Räder (v_l, v_r) in die Konfiguration einbeziehen:

$$k_2 = (x, y, \theta, v_l, v_r) \tag{2.2}$$

Die Lineargeschwindigkeit des Roboters, v, ergibt sich aus der linken (v_l) sowie der rechten (v_r) Radgeschwindigkeit:

$$v = \frac{v_r + v_l}{2} \tag{2.3}$$

Zur Berechnung der Winkelgeschwindigkeit ω, wird der Radabstand a, benötigt:

$$\omega = \frac{v_r - v_l}{a} \tag{2.4}$$

2.1.1 Kinematisches Modell

Kinematik beschreibt durch mathematische Gleichungen eine Bewegung, ohne die Ursache für diese Bewegungen zu berücksichtigen. Hierbei wird zwischen der direkten Kinematik (engl. forward kinematics) und inversen Kinematik (engl. inverse kinematics) unterschieden.

Die Bestimmung, welche Konfiguration bei gegebenen Radgeschwindigkeiten erreicht wird, wird als direkte Kinematik oder Vorwärtstransformation bezeichnet. Wenn der Roboter zu jeweiligen Zeitpunkt t die Ausrichtung $\theta(t)$, die Lineargeschwindigkeit $v(t)$ und die Winkelgeschwindigkeit $\omega(t)$ hat, errechnet sich die Konfiguration zum Zeitpunkt t durch Integration der Linear- und Winkelgeschwindigkeiten unter Beachtung der jeweiligen Ausrichtung:

$$x(t) = \int_0^t v(t) \cos(\theta(t))\, dt \tag{2.5}$$

$$y(t) = \int_0^t v(t) \sin(\theta(t))\, dt \tag{2.6}$$

$$\theta(t) = \int_0^t \omega(t)\, dt \tag{2.7}$$

Da im Bereich Robotik oftmals diese Werte nur zu diskreten Zeitpunkten bekannt sind, ergibt sich aus den Gleichungen das folgende diskretisierte Differentialgleichungssystem.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{2.8}$$

Die inverse Kinematik beschäftigt sind hingegen mit der Frage, wie die Radgeschwindigkeiten zu wählen sind, um eine bestimmte Konfiguration zu erreichen.

Aufgrund seiner Konstruktion unterliegt ein differentieller zweirädriger Roboter sogenannten nicht-holonomen Zwangsbedingungen. In der Mechanik beschreiben Zwangsbedingungen Einschränkungen der Bewegungsmöglichkeiten eines Systems. Holonom bzw. nicht-holonom charakterisieren diese Zwangsbedingungen weiter.

Seien $k_1, \ldots, k_n$ unabhängige Elemente der Konfiguration. Ein System ist holonom, wenn es keine Zwangsbedingungen aufweist oder die Zwangsbedingungen holonom sind. Holonome Zwangesbedingungen sind in folgender Form beschreibbar:

$$f(k_1, \ldots, k_n) \quad = \quad 0 \tag{2.9}$$

Sollten die Zwangsbedingungen nicht durch eine solche Gleichung beschreibbar sein, handelt es sich um nicht-holonome Zwangsbedingungen und das System ist nicht-holonom. Nicht-holonome Zwangsbedingungen können z.B. durch Ungleichungen beschreibbar sein oder sie erfordern die zeitliche Ableitung von Konfigurations-Elementen $(\bar{k_1}, \ldots, \bar{k_n})$, um die Geschwindigkeit oder die Beschleunigung in die Bedingung einbeziehen zu können. Konkret unterliegt ein differentieller zweirädriger Roboter folgender nicht holonomen Zwangsbedingung:

$$\dot{x} \sin(\theta) - \dot{y} \cos(\theta) \quad = \quad 0 \tag{2.10}$$

Anschaulich bedeutet diese Einschränkung, dass der Roboter sich nur entlang seiner Y-Achse bewegen kann und nicht entlang seiner X-Achse. Bedingt durch diese Zwangbedingung sind die für die inverse Kinematik benötigen Werte aus der Gleichung 2.8 mit Standardmethoden nicht mehr bestimmbar, was die Wegplanung für diese Klasse von Robotern schwieriger als z.B. für holonome Roboter macht.

Für weitergehende Information bzgl. Zwangsbedingungen sei auf das Kapitel Lagrange-Mechanik in [Nol90] verwiesen.

2.1.2 Dynamisches Modell

Bei diesem Modell werden dynamischen Effekte, wie die Radgeschwindigkeiten, berücksichtigt.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v_l} \\ \dot{v_r} \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(v_l + v_r)\cos(\theta) \\ \frac{1}{2}(v_l + v_r)\sin(\theta) \\ \frac{1}{w}(v_l - v_r) \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} v_l + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} v_r \tag{2.11}$$

Da die Radgeschwindgkeiten v_l und v_r kleiner als die Höchstgeschwindigkeit des Roboters v_{max}, sein müssen, gilt:

$$\text{abs}(v_l) \quad \leq \quad v_{max} \tag{2.12}$$

$$\text{abs}(v_r) \quad \leq \quad v_{max} \tag{2.13}$$

Die Geschwindigkeitsänderung ist limitiert durch die maximale Beschleunigung a_{max}:

$$\left|\frac{dv}{dt}\right| \leq a_{max} \tag{2.14}$$

Der Wert für a_{max} wird in diesem Modell als konstant betrachtet.

2.1.3 Modell für die Kreisfahrt

Mit Hilfe dieses Modells wird bestimmt, welche Kräfte auf dem Roboter bei einer nicht beschleunigten Kurvenfahrt wirken. Basierend auf diesen Kräften kann der minimale Kurvenradius bei einer bestimmten Geschwindigkeit bestimmt werden.

Da sich der Autor dieser Arbeit in Rahmen der Projektgruppe 416 (s. [Pro03]) bereits mit diesem Thema beschäftigt hat, wird hier nur eine kurze Zusammenfassung präsentiert.

Der Roboter verliert bei einer nicht beschleunigten Bewegung die Bodenhaftung, wenn die Zentripetalkraft $\left(F_{ZP} = \frac{m*v^2}{r}\right)$ größer als die Reibungskraft $(F_R = \mu * F_n = \mu * m * g)$ wird. Es gilt also:

$$F_R > F_{ZP} \tag{2.15}$$

$$\mu * m * g > \frac{m * v^2}{r} \tag{2.16}$$

$$r > \frac{v^2}{\mu * g} \tag{2.17}$$

Mit Hilfe der Gleichung 2.17 kann der minimale Kurvenradius bei einer gegebenen Geschwindigkeit v berechnet werden, falls μ bekannt ist. Da in diesem Modell die Haftreibungszahl μ als konstant angenommen wird, kann diese Zahl empirisch bestimmt werden. Hierzu fährt der Roboter einen Kreis ab. Während der Fahrt wird langsam die Geschwindigkeit erhöht, so lange bis der Roboter die Bodenhaftung verliert. Mit Hilfe der Gleichung 2.17 kann danach μ berechnet werden.

3 Sensoren

> „Sensor [lateinisch] der, allgemein ein Funktions- oder Bauelement, das zur Erfassung physikalischer, chemischer oder elektrochemischer Größen und deren Umwandlung in elektrische Signale dient; in diesem Sinn ist der Sensor ein Messgrößenaufnehmer (Messfühler)." [Mey06]
>
> „Stimulus [lateinisch] der,Anreiz, Antrieb; Reiz." [Mey06]

In diesem Kapitel werden Sensoren vorgestellt, die im Bereich mobiler Roboter zum Einsatz kommen. Gegliedert sind die Sensoren nach Anwendungsgebieten. Beginnend mit Sensoren, die eine Bestimmung der Positionsänderung ermöglichen, werden Sensoren vorgestellt, die eine absolute Positionsbestimmung des Roboters ermöglichen. Kontaktsensoren erlauben eine Detektion von Hindernissen in der unmittelbaren Nähe des Roboters. Kamerabasierte Sensoren werden aufgrund ihres weiten Anwendungsbereiches separat behandelt. Zu Beginn des Kapitels werden Kriterien zur Bewertung von Sensoren vorgestellt.

Dieses Kapitel basiert auf [Eve95, BRF96, SN04, Fra04].

3.1 Bewertungskriterien

Basierend auf [SN04, S. 92ff] und [Fra04, S. 13ff] können Sensoren unter anderem bezüglichen den folgenden Kriterien bewertet werden:

- Messbereichsumfang (Range)
  Gibt den kleinsten und größten Stimulus an, der gemessen werden kann.
- Auflösung (Resolution)
  Kleinster Änderungschritt des Stimulus, der vom Sensor erkannt werden kann.
- Messabweichung (Systematic errors, Random errors)
  Bei Abweichungen zwischen den gemessenen Werten und den tatsächlichen Werten wird zwischen systematischen und zufälligen Abweichungen unterschieden.
  Die mehrmalige Messung eines unveränderten Stimulus führt bei zufälligen Abweichungen zu einer Streuung der Messwerte. Bei systematischen Abweichungen bleibt der gemessene Wert und somit auch die Messabweichung konstant. In der Praxis überlagern sich beide Arten der Messabweichung.
- Messrate (Bandwidth, Frequency)
  Anzahl der möglichen Messung pro Sekunden.
- Latenzzeit
  Die Zeit, bis sich in den gemessenen Werten, eine Änderung des Stimulus niederschlägt.

Diese Kriterien bilden die Basis für die Evaluation der Sensoren in Kapitel 9.

3.2 Sensoren für Koppelnavigation

In diesem Unterkapitel werden Sensoren vorgestellt, die sich für die Koppelnavigation eignen. Bei der Koppelnavigation (engl. Dead Reckoning) wird auf Basis einer bekannten Position und der seitdem zurückgelegten Strecke auf die aktuelle Position geschlossen. Die zurückgelegte Strecke kann auf verschiedenen Arten approximiert werden. In der Schifffahrt dienen der anliegende Kurs und die Geschwindigkeit als Grundlage. Bei fahrenden Robotern wird die zurückgelegte Strecke mit Hilfe von Radencodern bestimmt, die die Drehung der Räder messen. In diesem Fall wird die Koppelnavigation auch als Odometrie bezeichnet.

Ein weiterer Spezialfall ist die Inertialnavigation, auch Trägheitsnavigation genannt, die mit Hilfe eines Gyroskops auf die zurückgelegte Strecke schließt.

Unterstützt werden beide Verfahren durch Kompasssensoren und Beschleunigungssensoren. Kompasssensoren verbessern die Schätzung der zurückgelegten Strecke, da eine absolute Bestimmung der Ausrichtung des Roboters möglich ist. Mit Hilfe von Beschleunigungssensoren kann durch Integration der Messwerte auf die aktuelle Geschwindigkeit geschlossen werden. Außerdem kann erkannt werden, ob die Räder die Bodenhaftung verloren haben (s. [BP04]).

Da die zurückgelegte Strecke in allen Fällen nur geschätzt werden kann, summieren sich die Fehler im Laufe der Zeit auf und der Positionierungsfehler wird kontinuierlich größer. Aus diesem Grund muss in regelmäßigen Abständen die Position mit Sensoren bestimmt werden, die eine absolute Position liefern können.

3.2.1 Radencoder

Radencoder dienen zur Bestimmung der Bewegung eines Rades. Hierzu wird an der Antriebswelle eine Scheibe montiert, die mit einem Muster versehen ist und die von einem oder mehreren Sensoren abgetastet wird. Durch das Muster auf der Scheibe kann eine Bewegung detektiert werden. Bei den inkrementellen Radencodern kommt ein Sensor zum Einsatz, der ein schwarz-weiß Muster (s. Abbildung 3.1) abtastet. Der Sensor lie-

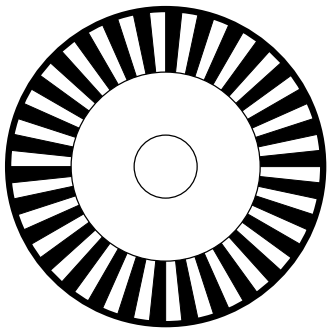

Abbildung 3.1: Radencoderscheibe für normale und Quadratur-Radencoder

fert, je nach detektierter Farbe, zwei unterschiedliche Messwerte zurück. Eine Änderung des Messwertes bedeutet eine Bewegung des Rades. Eine Erkennung der Drehrichtung ist mit dieser Art von Radencodern nicht möglich; hierfür sind Quadratur-Radencoder notwendig. Um die Drehrichtung zu bestimmen, wird ein zweiter Sensor um 90° phasenverschoben zum ersten angebracht. Je nach Drehrichtung des Rades ändern sich die

Messwerte der Sensoren in unterschiedlicher Reihenfolge, was eine Bestimmung der Drehrichtung ermöglicht.

Ein dritter Typ, der jedoch im Bereich der mobilen Roboter selten zum Einsatz kommt, sind die Absoluten-Radencoder. Hierbei tasten mehrere Sensoren die Scheibe ab. Durch das Muster auf der Scheibe (s. Abbildung 3.2) kann im Gegensatz zu den Inkrementellen-

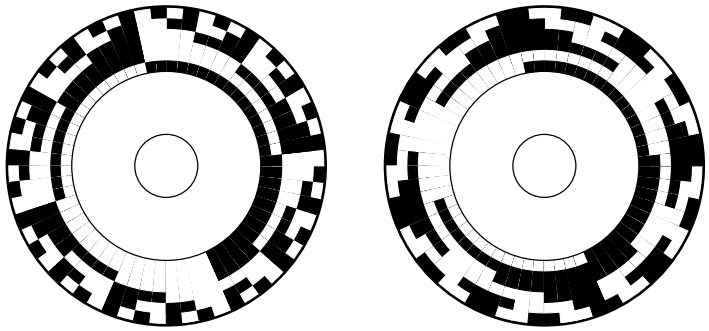

Abbildung 3.2: Radencoderscheiben für Absolute-Radencoder

oder den Quadratur-Radencoder die Ausrichtung der Scheibe bestimmt werden.

Bei einem differentiellen zweiräderigen Roboter kann aus der Bewegung der beiden Rädern die relative Bewegung des Roboters mit Hilfe der Messwerte von Quadratur-Radencoder approximiert werden. In der Literatur finden sich hierfür zwei unterschiedliche Ansätze. Bei beiden Ansätzen wird in einem ersten Schritt für jedes Rad aus der Anzahl der Sensorimpulse auf die zurückgelegte Strecke geschlossen, indem die Anzahl an Sensorimpulse mit einer Konstante multipliziert wird, die angibt, welche Strecke das Rad pro Impuls zurücklegt. Das Ergebnis ist die von den beiden Räder zurückgelegte Strecke d_l und d_r. Der erste Algorithmus (s. [Pro01, S. 33f]) berechnet die relative Bewegung unter der Annahme, dass der Roboter einen Kreisabschnitt gefahren ist (s. Abbildung 3.3). O.B.d.A. wird der Fall beschrieben, dass der Roboter eine Rechtskurve

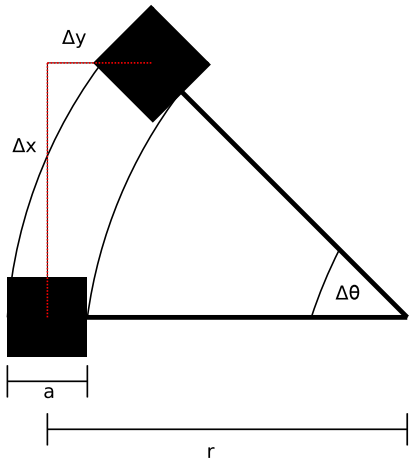

Abbildung 3.3: Approximation der Bewegung durch einen Kreisabschnitt (nach [Pro01, S. 33])

gefahren ist. Gleichung 3.1 beschreibt, wie die Richtungänderung ($\Delta\theta$), die Bewegung entlang der X-Achse (Δx) und entlang der Y-Achse (Δy) berechnet werden kann.

$$\begin{pmatrix} \Delta\theta \\ \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} \frac{d_l - d_r}{a} \\ r * \sin(\theta) \\ r * (1 - \cos(\theta)) \end{pmatrix} \tag{3.1}$$

Hierbei ist r der Radius des Kreisabschnitts für den gilt: $r = \frac{d_l}{\theta} - \frac{a}{2}$.
Der zweite Algorithmus (s. [BRF96, S. 19]) approximiert die zurückgelegte Strecke unter der Annahme, dass der Roboter, nachdem er eine Richtungsänderung ausgeführt hat, eine Gerade befährt. Die Richtungsänderung wird wie im ersten Algorithmus berechnet. Die zurückgelegte Strecke wird durch $d = \frac{d_l + d_r}{2}$ approximiert.

$$\begin{pmatrix} \Delta\theta \\ \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} \frac{d_l - d_r}{a} \\ d * \cos(\theta) \\ d * \sin(\theta) \end{pmatrix} \tag{3.2}$$

Unabhängig davon, wie die zurückgelegte Strecke approximiert wird, unterliegt die Odometrie nach [BF94] verschiedenen Fehlerquellen, die in systematische und nicht-systematische Fehler unterteilt werden.
Systematische Fehler sind:

- unbekannter Raddurchmesser
- ungleichmäßige Radoberfläche
- unzureichende Auflösung der Radencoder
- zu große oder zu kleine Berechnungsintervalle
- unbekannter Radabstand
- Fehlstellung der Räder
- unzureichendes Odometriemodell

Nicht-Systematische Fehler sind:

- unebener Boden
- verlorene Bodenhaftung
- Kollisionen mit anderen Objekten

Systematische Fehler sind besonders kritisch, da diese Fehler in jedem Schritt die Berechnung verfälschen. Nicht-systematische Fehler wirken sich hingegen nur innerhalb eines kleinen Zeitraumes aus. In Kapitel 9.1 wird ein Verfahren vorgestellt, dass die systematischen Fehler verringert.

3.2.2 Kompasssensor

Ein Kompasssensor dient zur Bestimmung der Ausrichtung des Roboters bezüglich des Magnetfelds der Erde. Die am häufigsten zum Einsatz kommenden Kompasssensoren sind Hall-Effekt, Fluxgate und Magnetoresistive Kompasse.
Beim Hall-Effekt baut sich ein elektrisches Feld auf, wenn sich ein stromführender Leiter in einem Magnetfeld befindet. Dieses elektrische Feld ist orthogonal zum magnetischen Feld und zum Strom ausgerichtet und wird zur Bestimmung von Magnetfeldern genutzt. Hall-Effekt Kompasse sind preiswert, jedoch ist ihre Auflösung gering und die Latenzzeit sehr hoch.

Ein Fluxgate Kompass besteht aus einem hochpermeablen Spulenkern, der von zwei getrennten Wicklungen umgeben ist. Durch die innere Wicklung fließt ein sinusförmiger Wechselstrom, der den Kern periodisch bis in die Sättigung magnetisiert. Durch das Magnetfeld kommt es bei den Ausgangssignalen der inneren und äußeren Wicklung zu einer Phasenverschiebung, die zur Bestimmung des Magnetfeldes genutzt wird. Im Vergleich zu einem Hall-Effekt Kompass ist ein Fluxgate Kompass genauer, jedoch auch größer und teurer.

Magnetoresistive Kompasse beruhen auf dem Anisotrope-Magneto-Resistive Effekt. Fließt Strom durch ein ferromagnetisches leitfähiges Material, ändert sich der elektrische Widerstand in Abhängigkeit vom Winkel zwischen Stromrichtung und Magnetfeld. Aus der Messung des Widerstandes wird auf die Richtung des Magnetfeldes geschlossen.

Unabhängig vom eingesetzten Typ können die Messergebnisse von Kompassen durch künstliche Magnetfelder oder durch Verzerrungen des Erdmagnetfeldes verfälscht werden. Künstliche Magnetfelder können zum Beispiel durch elektrische Motoren oder durch Stromleitungen entstehen, während durch Stahlträger oder andere Metallansammlungen das bestehende Magnetfeld verzerrt wird.

3.2.3 Gyroskop

Ein Gyroskop dient zur Bestimmung von Ausrichtungsänderungen. Da ein Gyroskop seine Ausrichtung auch bei Ausrichtungsänderung des Roboters beibehält, kann somit eine absolute Ausrichtung bestimmt werden. Unterschieden werden mechanische und optische Gyroskope.

Mechanische Gyroskope haben als zentrales Element eine mit hoher Geschwindigkeit rotierende Masse, die beweglich gelagert ist. Durch die Drehimpulserhaltung bleibt die Ausrichtung der Masse auch dann unverändert, wenn der Roboter seine Ausrichtung ändert.

Optische Gyroskope messen die Winkelgeschwindigkeit und berechnen aus diesen Informationen die Ausrichtungsänderung. Zwei Laser senden Lichtimpulse durch ein gebogenes Glasfaserkabel. Die Lichtimpulse des einen Lasers durchlaufen das Glassfaserkabel im Uhrzeigersinn, während die anderen Lichtimpulse es gegen dem Uhrzeigersinn durchlaufen. Im Ruhezustand kommen beide Lichtimpulse zeitgleich an. Bei einer Drehung hat einer der beiden Lichtimpulse einen minimal kürzeren Weg und kommt vor dem zweiten Lichtimpulse an. In der Praxis werden zwei Lichtwellen genutzt, die sich im Stillstand gegenseitig auslöschen. Bei einer Drehung löschen sich die beiden Lichtwellen nicht mehr vollständig aus und anhand der Interferenzmuster lässt sich die Beschleunigung bestimmen.

3.2.4 Beschleunigungssensor

Beschleunigungssensoren messen, wie der Name schon sagt, die Beschleunigung, die auf den Roboter wirkt. Unterschiedenen werden kapazitive, piezoresistive, piezoelektrische und thermische Beschleunigungssensoren. Bei allen Sensoren wird die Kraft gemessen, die auf eine Test-Masse wirkt. Aus diesen Kräften wird auf die Beschleunigung geschlossen. Die Sensoren unterscheiden sich in der Art, wie die auf die Test-Masse wirkenden Kräfte gemessen werden.

Bei kapazitiven Beschleunigungssensoren wird die Test-Masse elastisch gelagert. Oberhalb und unterhalb der Test-Masse wird jeweils ein Plattenkondensator angebracht. Von jedem Plattenkondensator befindet sich eine Platte am Gehäuse und eine Platte an der Test-Masse. Durch die Beschleunigung verschiebt sich die Test-Masse, der Abstand zwischen den Platten der beiden Kondensatoren ändert sich und somit auch die Kapazität der beiden Kondensatoren. Aus der Änderung der Kapazität wird die Beschleunigung berechnet.

Die Test-Masse ist bei piezoresistiven Sensoren über Piezowiderstände befestigt. Bei einer Beschleunigung entsteht eine mechanische Spannung und in Folge dessen ändern die Piezowiderstände ihren Widerstand. Über eine angelegte Spannung kann der Widerstand bestimmt und auf die Beschleunigung geschlossen werden.

Bei den piezoelektrische Sensoren ist die Test-Masse mit dem Gehäuse durch eine piezoelektrische Schicht verbunden. Durch die Beschleunigung bewegt sich die Test-Masse und die piezoelektrische Schicht wird deformiert. Die bei dieser Deformierung entstehenden Ladungsteilchen werden zur Detektierung der Beschleunigung genutzt. Da die Ladungsteilchen nur im Moment der Deformierung entstehen, können konstante Beschleunigungen mit diesem Verfahren nicht gemessen werden.

Thermische Sensoren bestehen aus der Test-Masse und zwei Temperatur-Sensoren, die oberhalb und unterhalb der Test-Masse angebracht sind. Die Test-Masse wird aufgeheizt und die beiden Sensoren messen die Temperaturen. Bei einer Beschleunigung verschiebt sich die Test-Masse und die Temperatur-Sensoren messen unterschiedliche Temperaturen. Aus dieser Differenz kann auf die Beschleunigung geschlossen werden.

3.3 Sensoren zur Positionsbestimmung

Sensoren zur Positionsbestimmung lassen sind in drei Gruppen unterteilen.

Die erste Gruppe (z.B. GPS-Empfänger) bestimmt die Position mit Hilfe von Signalfeuern. Signalfeuer (engl. beacon) sind Sender, die elektromagnetische Wellen (z.B. Licht oder Funk) aussenden, die von Sensoren direkt oder indirekt zur Positionsbestimmung benutzt werden können. Ist die Bestimmung der Winkel zu zwei Signalfeuern möglich, kann die Position durch Triangulation bestimmt werden. Durch die beiden Winkel und das Wissen, welche Position die beiden Signalfeuer haben, kann die aktuelle Position bestimmt werden, sofern die Position nicht genau auf der Verbindungslinie zwischen den beiden Signalfeuern liegt. Zwischen der aktuellen Position und der Positionen der Signalfeuer muss sich ein Dreieck aufspannen lassen.

Ist jedoch nur eine Abstandsmessung möglich, so ist der Abstand zu drei Signalfeuern notwendig, um eine Positionbestimmung durch Trilateration (s. Abbildung 3.4) zu ermöglichen.

Die zweite Gruppe (z.B. Kamerabasierte Sensoren) bestimmt die Position anhand von künstlichen oder natürlichen Landmarken, die durch ihre charakteristischen Merkmale

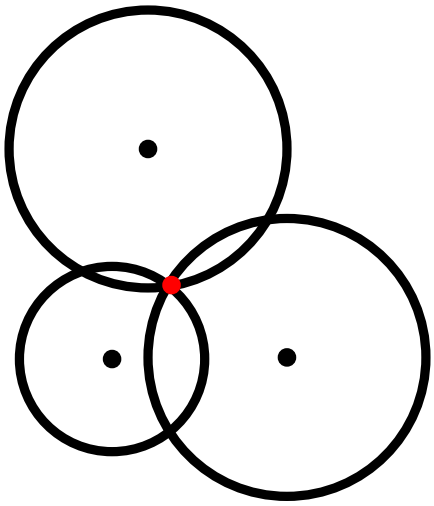

Abbildung 3.4: Trilateration am Beispiel von drei Signalfeuern

erkennbar sind. Natürliche Landmarken sind alle Landmarken, die nicht speziell für die Navigation aufgestellt werden, also z.B. Berge, Bäume, Flüsse, aber auch Häuser und innerhalb eines Hauses Tische, Stühle, Türen oder Fenster.

Die letzte Gruppe (z.B. Ultraschallsensoren und Laserscanner) sind die Kartenbasierten Sensoren. Diese Sensoren messen den Abstand zu Objekten, indem sie elektromagische Impulse aussenden. Diese Impulse werden von den Objekten reflektiert und von dem Sensor wieder empfangen. Aus der Zeit, die zwischen dem Senden und dem Empfangen der Impulse vergeht, kann auf die Entfernung zu dem Objekt geschlossen werden. Mit Hilfe dieser Informationen kann kontinuierlich eine Karte der Umgebung aufgebaut werden, auf Basis derer eine Positionsbestimmung möglich ist. Ist bereits eine Karte der Umgebung vorhanden, kann mit Hilfe der Sensorinformationen auf die aktuelle Position geschlossen werden.

3.3.1 GPS

GPS, das offiziell „Navigational Satellite Timing and Ranging - Global Positioning System" heißt, ist ein satellitengestütztes System, das zur weltweiten Positionsbestimmung genutzt wird. Das System basiert auf mindestens 24 Satelliten, die die Erde in einer Entfernung von ungefähr 20190 km umkreisen und permanent die Uhrzeit und ihre eigene Position übermitteln. Aus den Informationen von drei Satelliten und aus der Messung der Signallaufzeiten könnte die Position des Empfängers berechnet werden, falls der Empfänger über eine sehr genaue Uhr verfügt. Da die meisten Empfänger jedoch über keine ausweichend genaue Uhr verfügen, wird das Signal eines vierten Satelliten zur Positionsbestimmung benötigt. Eine Positionsbestimmung innerhalb eines Gebäudes war bis vor kurzem nicht möglich, da die Sendestärke der Satelliten nicht ausreichte. Verbesserte Empfänger ermöglichen mittlerweile jedoch auch in Gebäuden, die die Signale der Satelliten nicht übermäßig dämpfen, eine Positionsbestimmung. Bedingt durch die Datenübertragungsrate der Satellitenübertragung, kann in einer Sekunde maximal 5-mal eine Positionsbestimmung stattfinden, die mit einer Latenz von 200 bis 300 ms behaftet ist. Die Genauigkeit der Positionbestimmung beträgt ungefähr 15 m, sofern keine weite-

ren Hilfsmittel wie DGPS zum Einsatz kommen. Bei DGPS werden von Bodenstationen Korrekturwerte übermittelt, mit deren Hilfe die Genauigkeit erhöht werden kann.

3.3.2 Ultraschall-Sensoren

Ultraschall-Sensoren senden Schallwellen in einem für Menschen nicht hörbaren Bereich (über 20 kHz) aus. Treffen die Schallwellen auf ein Objekt, werden sie zum Teil zum Sender zurückgeworfen. Ein Empfänger detektiert diese reflektierten Schallwellen und berechnet aus der Laufzeit der Schallwellen die Entfernung zum Objekt.

Erschwert wird die Entfernungsmessung dadurch, dass die Ausbreitungsgeschwindigkeit der Schallwellen abhängig von der Umgebungstemperatur ist und dass die Schallwellen von mehreren Objekten reflektiert werden können, bevor sie den Empfänger erreichen. Beides verfälscht die Messergebnisse. Ein weiterer Nachteil ist, dass sich Schallwellen kegelförmig ausbreiten. Wird ein Objekt detektiert ist nur die Aussage möglich, dass sich Objekte in einem gewissen Bereich befindet. Es ist keine Aussage möglich, wieviele Objekte sich in diesem Bereich befinden.

3.3.3 Laserscanner

Laserscanner senden kurze Lichtimpulse aus, die von einem Objekt reflektiert werden und anschließend von einem Empfänger detektiert werden. Aus der Laufzeit der Lichtimpulse kann auf die Entfernung geschlossen werden.

Lichtdurchlässige Objekte können durch Laserscanner nicht erkannt werden. Ebenso bereiten Objekte, die Licht stark ablenken (z.B. Spiegel), Probleme. Hierbei kann der abgelenkte Lichtstahl von einem anderen Objekt zum Empfänger zurückgeworfen werden und so die Messergebnisse verfälschen.

3.4 Kamerabasierte Sensoren

In den Kamerabasierten Sensoren werden lichtempfindlichen Zellen eingesetzt, die matrixförmig angeordnet werden und auf elektromagetische Wellen aus dem ganzen sichtbaren und aus dem infraroten Spektrum reagieren. Um Farbbilder zu erzeugen, wird das sichtbare Licht durch Filter oder Prismen in seine Rot-, Grün- und Blauanteile zerlegt. Aus den Messungen der verschiedenen Farbanteile wird die Farbe des jeweiligen Bildpunkts berechnet.

Im Bereich der Robotik werden Kamerabasierte Sensoren hauptsächlich zur Positionsbestimmung mit Hilfe von Landmarken eingesetzt. Abbildung 3.5 zeigt künstliche Landmarken, die im Bereich Roboterfußball zur Markierung des Tores genutzt werden. Falls der Winkel zu den Landmarken aus den Kamerabildern berechnet werden kann, kommt das Verfahren Triangulation zum Einsatz. Ist nur die Entfernung zu den Landmarken bekannt, wird Trilateration genutzt. Abbildung 1.1 zeigt einen Roboter, der mit künstlichen Landmarken in Form von Farbflächen versehen wurde, um eine Erkennung durch eine an der Decke montierten Kamera zu ermöglichen.

Weitere Ansätze zur Erkennung von Objekten sind in der Diplomarbeit von Volkmar Frinken ([Fri06]) zu finden.

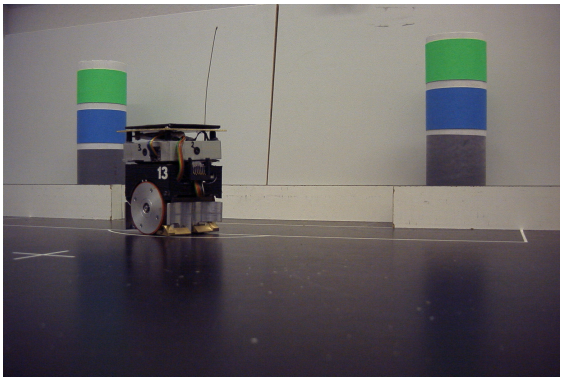

Abbildung 3.5: Künstliche Landmarken zur Markierung des Tores

4 Fusion von Sensordaten

> „Dynamisches System, abstrakte Beschreibungsweise für einen (physikalischen, chemischen, ökonomischen, ökologischen,...) Vorgang. Der Zustand eines dynamischen Systems wird durch eine Reihe von Variablen dargestellt, die die physikalische Situation beschreiben und einer Zeitentwicklung unterworfen sind." [Stö04]

In diesem Kapitel wird eine Klasse von Filtern vorgestellt, die eingesetzt werden, um aus fehlerbehafteten Sensorinformation eine möglichst gute Schätzung des beobachteten, dynamischen Systems zu erzeugen.

Der Zustand eines dynamischen Systems kann meistens nicht direkt bestimmt werden, sondern nur mittels Sensoren. Die von den Sensoren durchgeführten Messungen sind jedoch fehlerbehaftet (s. Kapitel 3). Ziel der Sensorfusion ist es, aus den Messungen aller Sensoren, zu einer möglichst guten Schätzung des Zustandes des dynamischen Systems zu gelangen. Hierbei fließen neben den reinen Messdaten auch Informationen über die Genauigkeit der Sensoren ein.

Dieses Kapitel basiert auf [FHL+03],[Neg03, S. 19ff] und [May79].

4.1 Methoden

Das Konzept des Bayes Filter[1] dient als Grundlage für eine ganze Klasse von Filtern, die den Zustand eines dynamischen Systems mit Hilfe von Beobachtungen bestimmen.

Der Zustand eines dynamischen Systems wird zum Zeitpunkt t durch die Zufallsvariable x_t repräsentiert. $Bel\,(x_t)$ beschreibt die Vermutung (engl. belief) bezüglich des Zustands des dynamischen Systems und nutzt eine Wahrscheinlichkeitsverteilung über x_t. $Bel\,(x_t)$ ist definiert als:

$$Bel\,(x_t) = p\,(x_t|\, z_1, z_2, \ldots, z_t) \tag{4.1}$$

Die Vermutung $Bel\,(x_t)$ gibt die Wahrscheinlichkeit an, mit der sich das dynamische System unter Berücksichtigung der Beobachtungen $z_1, \ldots, z_t$ zum Zeitpunkt t im Zustand x_t befindet. Der Rechnenaufwand zur Berechnung der Vermutung $Bel\,(x_t)$ steigt exponentiell mit der Anzahl der Beobachtungen. Unter der Annahme, dass das dynamische System die Markov-Annahme erfüllt, kann die exponentielle Komplexität vermieden werden, da in diesem Fall alle relevanten Informationen in der Zustandsvariable x_t enthalten sind. Für die Berechnung von x_t werden unter dieser Annahme nur x_{t-1} und z_t benötigt.

[1] Der Bayes Filter verdankt seinem Namen Thomas Bayes (1702-1761), der im seinem Aufsatz „Essay Towards Solving a Problem in the Doctrine of Chances" [Bay63] eine Regel für das Aktualisieren von Wahrscheinlichkeiten unter Berücksichtigung von neuen Beobachtungen aufgestellt hat.

Der Übergang vom Zeitpunkt t nach $t+1$ erfolgt in zwei Schritten:

Im ersten Schritt, der sogenannten Vorhersage, werden die Vermutungen gemäß des Zustandsänderungsmodell aktualisiert.

$$Bel^{-}(x_t) = \int p(x_t \,|x_{t-1}) \, Bel(x_{t-1}) \; dx_{t-1} \tag{4.2}$$

Das Zustandsänderungsmodell fließt an der Stelle $p(x_t \,|x_{t-1})$ in die Gleichung ein und gibt an, mit welcher Wahrscheinlichkeit das System vom Zustand x_{t-1} in den Zustand x_t übergeht.

Im zweiten Schritt, der sogenannten Korrektur, werden die Vermutungen gemäß des Beobachtungsmodell und der aktuellen Beobachtung aktualisiert.

$$Bel(x_t) = \alpha_t p(z_t \,|x_t) Bel^{-}(x_t) \tag{4.3}$$

$p(z_t \,|x_t)$ nutzt das Beobachtungsmodell, um zu bestimmen, mit welcher Wahrscheinlichkeit die Beobachtung z_t im Zustand x_t erfolgen kann. α_t dient zur Normalisierung, damit gewährleistet ist, dass sich die Wahrscheinlichkeiten zu 1 aufaddieren.

Alle im folgenden beschriebenen Filter beruhen auf diesem Konzept, unterscheiden sich jedoch zum einen in der Repräsentation der Vermutung und zum anderen darin, ob es nur eine Vermutung (unimodal) oder mehrere Vermutung (multimodal) über dem Zustand des Systems gibt.

4.1.1 Kalmanfilter

Der Kalmanfilter geht auf Rudolf Kalman zurück und wurde in [Kal60] erstmalig präsentiert. Er gehört zu der am häufigsten eingesetzten Variante des Bayes Filter (vgl. [FHL+03]). Der Kalmanfilter macht einige Annahmen, die den Rechenaufwand niedrig halten. Während der Lineare Kalmanfilter ein lineares dynamisches System voraussetzt, kann der erweiterte Kalmanfilter auch in nicht-linearen dynamischen System eingesetzt werden.

4.1.1.1 Linearer Kalmanfilter

Da bei dieser Variante ein lineares dynamisches System vorausgesetzt wird, können die Zustandsänderungen als lineare Gleichung formuliert werden:

$$x_k = Ax_{k-1} + w_{k-1} \tag{4.4}$$

$x_k \in R^n$ ist der wahre Zustand des Systems zum Zeitpunkt k, der bestimmt werden soll. x_{k-1} beschreibt den Zustand zum vorherigen Zeitpunkt. $w_{k-1} \in R^n$ repräsentiert die Ungewissheit des Modells durch nicht modellierbare Effekte. Die Matrix A (n × n) überführt das Modell aus den Zeitpunkt $k-1$ in den Zeitpunkt k, ohne Berücksichtigung der Ungewissheit bezüglich der Modellgüte.

Das Beobachtungsmodell ist ebenfalls als lineare Gleichung gegeben:

$$z_k = Hx_k + v_k \tag{4.5}$$

$z_k \in R^m$ ist der wahre Wert einer Beobachtung. Da die Beobachtungen jedoch durch diverse Fehler verfälscht werden können, modelliert $v_k \in R^m$ den Beobachtungsfehler.

Beobachtet wird mit Sensoren der Wert von $z_k - v_k$. Die Matrix H (m × n) überführt den aktuellen Zustand x_k in die Form einer Beobachtung ohne Berücksichtigung möglicher Beobachtungsfehler.

Repräsentiert werden die Wahrscheinlichkeiten im Kalmanfilter durch eine Gaußverteilung, die mit Hilfe des Erwartungswerts (μ_t) und der Varianz (Σ_t) eindeutig beschreibbar ist. Somit gilt laut [FHL+03]:

$$Bel\left(x_t\right) \approx N\left(x_t; \mu_t; \Sigma_t\right) \tag{4.6}$$

Der Vorhersageschritt besteht aus den folgenden Berechnung:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{4.7}$$
$$\hat{P}_k^- = AP_{k-1}A^T + Q \tag{4.8}$$

Während Gleichung 4.7 den Erwartungswert der Gaußverteilung aktualisiert, aktualisiert Gleichung 4.8 die Varianz, wodurch der Vorhersageschritt realisiert ist. Hierbei ist u_k der Regelungseingriff, der mit Hilfe der Matrix B an die Form des Systemzustandes angepasst wird. Q modelliert die Ungewissheit bezüglich der Modellgüte.

Der nächste Schritt, die Korrektur, wird durch drei Gleichungen berechnet:

$$K_k = P_K^- H^T \left(HP_k^- H^T + R\right)^{-1} \tag{4.9}$$
$$\hat{x_k} = \hat{x}_k^- + K_k\left(z_k - H\hat{x}_k^-\right) \tag{4.10}$$
$$P_k = \left(I - K_k H\right) P_k^- \tag{4.11}$$

In Gleichung 4.9 wird die Stärke der Korrektur unter Berücksichtigung der aktuellen Varianz und der Varianz der Messung (R) berechnet. Gleichung 4.10 aktualisiert den geschätzten aktuellen Zustand entsprechend der in Gleichung 4.9 berechneten Stärke. Gleichung 4.11 passt die Varianz an.

4.1.1.2 Erweiterter Kalmanfilter

Diese Erweiterung erlaubt es, den Kalmanfilter auch bei nicht-linearen, dynamischen Systemen zu benutzen. Hierzu wird das nicht-lineare System im Bereich des Erwartungswertes linearisiert, um ein lineares System zu approximieren.

In der Gleichung für die Zustandsänderung (Gleichung 4.4) wird Ax_{k-1} durch eine nicht lineare Funktion ersetzt:

$$x_k = f\left(x_{k-1}\right) + w_{k-1} \tag{4.12}$$

Der Vorhersageschritt:

$$\hat{x}_k^- = f\left(\hat{x}_{k-1}, u_k, 0\right) \tag{4.13}$$
$$\hat{P}_k^- = A_k P_{k-1} A_k^T + Q_{k-1} \tag{4.14}$$

A_k ist eine Jacobische Matrix, die das nicht-lineare System im Bereich des Erwartungswertes linearisiert.

$$A_k = \left.\frac{\partial f(x)}{\partial x}\right|_{x=\hat{x}_k^-} \tag{4.15}$$

Im Beobachtungsmodell wird die Matrix ebenfalls durch eine nicht-lineare Funktion ersetzt, die den Zustand in die Form der Beobachtung überführt:

$$z_k = h(x_k) + v_k \tag{4.16}$$

Der Korrekturschritt:

$$K_k = P_K^- H^T \left(H_k P_k^- H_k^T + V_k R V_k^T\right)^{-1} \tag{4.17}$$
$$\hat{x_k} = \hat{x}_k^- + K_k\left(z_k - h\left(\hat{x}_k^-, 0\right)\right) \tag{4.18}$$
$$P_k = (I - K_k H) P_k^- \tag{4.19}$$

Wobei H_k die folgende Jacobische Matrix ist:

$$H_k = \left.\frac{\partial h(x)}{\partial x}\right|_{x=\hat{x}_k^-} \tag{4.20}$$

Der erweiterte Kalmanfilter ist eine Verallgemeinerung des linearen Kalmanfilters. Wenn im erweiterten Kalmanfilter die Funktionen f und h linear sind, ergibt sich der lineare Kalmanfilter, da die Jacobischen Matrizen dann der Einheitsmatrix entsprechen.

4.1.2 Multihypothesen Tracking

Multihypothesen Tracking (MHT) hebt die Einschränkung des Kalmanfilters bezüglich der unimodalen Verteilung auf. Statt nur eine Vermutung zu verwenden, werden mehrere Vermutungen zur Bestimmung des Zustands verwendet und jede Vermutung wird für sich durch einen eigenen Kalmanfilter berechnet.
Laut [FHL+03] gilt:

$$Bel(x_t) \approx \sum_i w_t^{(i)} N\left(x_t; \mu_t^{(i)}, \Sigma_T^{(i)}\right) \tag{4.21}$$

$w_t^{(i)}$ ist hierbei die Gewichtung der jeweiligen Vermutung und ist proportional zur Wahrscheinlichkeit, dass diese Vermutung die Beobachtung erklärt. Die Implementierung des Multihypothesen Tracking ist fehleranfällig, da in Abhängigkeit von der Beobachtung sowohl neue Vermutungen erzeugt, als auch bereits bestehende zusammengeführt werden müssen. Neue Vermutungen werden erzeugt, wenn keine existierende Vermutung die Beobachtung hinreichend erklärt, während Vermutungen zusammengeführt werden, wenn sich mehrere Vermutung ähneln.

4.1.3 Gitterbasierte Verfahren

Im Gegensatz zu den bislang vorgestellten Filtern, wird die Vermutung $Bel(x_t)$ bei den gitterbasierten Verfahren diskret dargestellt. Die Umgebung wird hierzu in Zellen eingeteilt. Jede Zelle hat, unabhängig von allen anderen Zellen, eine Vermutung, mit welcher Wahrscheinlichkeit sich das Objekt in ihr befindet. Da bei einem gitterbasierten Verfahren die Rechenzeit und der Platzbedarf exponentiell mit der Dimension des Zustandsvektors wächst, ist das Verfahren nur bei Problemen mit einer niedrigen Dimension des Zustandsvektors effizient nutzbar. Vorteil dieses Verfahren ist, dass eine multimodale Verteilung möglich ist und die Vermutung nicht einer Gaußverteilung folgen muss.

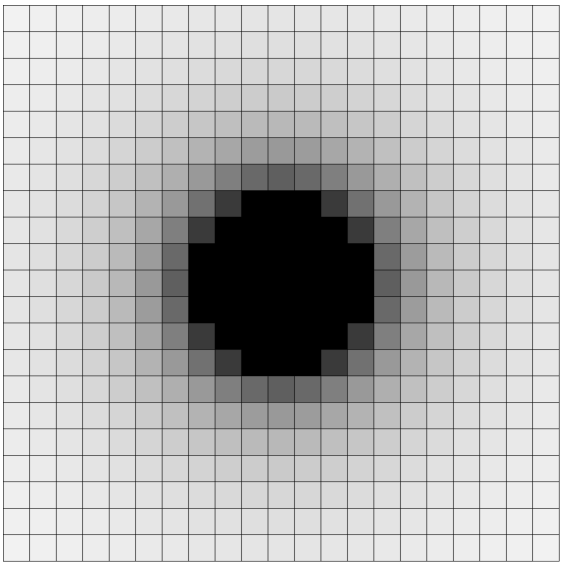

Abbildung 4.1: Beispiel für ein gitterbasiertes Verfahren

Abbildung 4.1 zeigt ein Beispiel, in welchem die Wahrscheinlichkeiten durch Graustufen dargestellt werden. Je dunkler eine Zelle ist, desto höher ist die Wahrscheinlichkeit, dass sich das Objekt in dieser Zelle befindet.

4.1.4 Partikelfilter

Laut [Sch05] entsprechen die Modelle für die Zustandsänderung und für die Beobachtung im wesentlichen denen aus dem erweiterten Kalmanfilter.

$$x_k = f(x_{k-1}, u_k, w_k) \tag{4.22}$$
$$z_k = h(x_k, v_k) \tag{4.23}$$

Statt durch eine Gaußverteilung, wird die Vermutung $Bel(x_t)$ beim Partikelfilter durch n Partikel dargestellt, die anfangs zufällig verteilt sind.

$$Bel(x_k) = \left\{ \left\langle x_k^{(i)}, w_k^{(i)} \right\rangle \middle| \, i = 1, \ldots, n \right\} \tag{4.24}$$

Jeder Partikel repräsentiert eine Vermutung (x), sowie die dazugehörige Wahrscheinlichkeit (w).

Im Vorhersageschritt wird die Vermutung entsprechend der Zustandsänderungsfunktion aus Gleichung 4.22 angepasst.

$$\hat{x}_k^{(i)-} = f(x_{k-1}, u_k, w_k) \tag{4.25}$$

Im Korrekturschritt wird für jeden Partikel die Wahrscheinlichkeit entsprechend der Beobachtung mit Hilfe der bedingten Wahrscheinlichkeit korrigiert.

$$w_k^{(i)} = p(z_k | x_k) \tag{4.26}$$

Basierend auf diesen Wahrscheinlichkeiten werden die Partikel in einem dritten Schritt neu verteilt. Die Neuverteilung der Partikel ist abhängig von den Wahrscheinlichkeiten. Je höher die Wahrscheinlichkeit einer Vermutung ist, desto höher ist die Wahrscheinlichkeit, dass ein Partikel in dieses Gebiet verschoben wird.

Weitergehende Informationen über Partikelfilter finden sich in dem Buch „Beyond the Kalman Filter - Particle Filters for Tracking Applications" ([RAG04]). Ein Beispiel für einen Partikelfilter findet sich in [FHL+03].

4.2 Latenzzeit

Die in Kapitel 4.1 vorgestellten Methoden gehen von der Annahme aus, dass die Beobachtungen direkt und ohne Zeitverzögerung zur Verfügung stehen, die Latenzzeit also 0 ist. Wie in Kapitel 3 gezeigt, ist diese Annahme jedoch für die meisten Sensoren nicht erfüllt. In der Literatur finden sich verschiedene Ansätze um dieses Problem zu lösen. Die gebräuchlichsten werden in den nächsten Unterkapiteln vorgestellt.

4.2.1 Verzögerte Berechnung

Die Berechnung des Zustands zum Zeitpunkt t wird solange verzögert, bis alle Beobachtungen für diesen Zeitpunkt eingetroffen sind. Erst dann wird die Berechnung durchgeführt. Nachteilig an dieser Lösung ist, dass nicht der Zustand zum aktuellen Zeitpunkt berechnet wird, sondern der Zustand für einen Zeitpunkt in der Vergangenheit. Die Verzögerung wird durch den Sensor mit der größten Latenzzeit bestimmt.

4.2.2 Extrapolation der Beobachtungen

Um die Beobachtungen direkt verarbeiten zu können, wird die Latenzzeit kompensiert, indem die Beobachtung durch das Zustandsänderungsmodell extrapoliert wird. Vorteil dieses Ansatzes ist, dass immer ein Zustand für den aktuellen Zeitpunkt berechnet wird. Daraus resultiert jedoch auch der Nachteil, dass der Zustand Ungenauigkeiten durch die Extrapolation unterliegt.

4.2.3 Neuberechnung

Wenn eine neue Beobachtung eintrifft, werden ab dem Zeitpunkt, zu dem die Beobachtung gehört, alle Berechnungen unter Berücksichtigung der neuen Beobachtung erneut ausgeführt. Somit sind im aktuellen Zustand immer alle verfügbaren Beobachtungen optimal einberechnet. Nachteil dieser Methode ist, dass der Berechnungsaufwand und der Speicherbedarf steigt.

5 Wegplanung

> „Path planning refers to the problem of geometrically specifying a sequence of robot positions and orientations that move the robot between the start and goal whilst avoiding collisions [...]" [Doy95]

Basierend auf einer erstellten Karte und der aktuellen Konfiguration des Roboters, soll die Wegplanung einen möglichst optimalen Weg zu einer gegebenen Zielkonfiguration berechnen. Was unter optimal zu verstehen ist, hängt von dem Einsatzzweck des Roboters ab. Je nach Einsatzgebiet werden an den Weg verschiedene Anforderungen gestellt, wie z.B. dass der Weg möglich kurz sein soll. Zur Bewertung eines Wegplanungs-Algorithmus können folgende, zum Teil konkurrierende, Kriterien zum Einsatz kommen:

- Vollständigkeit
  Gibt an, ob der Algorithmus immer einen Weg vom Startpunkt zum Zielpunkt findet, sofern mindestens einer existiert.
- Weglänge
  Der berechnete Weg vom Startpunkt zum Zielpunkt sollte möglichst kurz sein.
- Fahrzeit
  Der Weg sollte so beschaffen sein, dass der Roboter möglichst schnell ans Ziel kommt. Nicht immer ist der kürzeste Weg auch der schnellste.
- Kollisionsvermeidung
  Der Weg sollte so beschaffen sein, dass der Roboter nicht mit anderen Objekten kollidiert. Hierbei wird zwischen zeitlicher und räumlicher Kollisionsvermeidung unterschieden. Die meisten Ansätze versuchen, dass Hindernis zu umfahren. Falls das Hindernis sich bewegt, ist auch eine zeitliche Kollisionsvermeidung möglich. Hierbei wird versucht eine Kollision durch Änderung der Geschwindigkeit zu verhindern. Beispiel hierfür wäre eine Kreuzung, an der zwei Fahrzeuge gleichzeitig ankommen würden. Bei der zeitlichen Kollisionsvermeidung ändern die beiden Fahrzeuge bei der Annäherung an die Kreuzung ihre Geschwindigkeit so, dass sie nicht mehr gleichzeitig, sondern hintereinander die Kreuzung überqueren.
- Berücksichtigung der Robotereigenschaften
  Der Weg sollte so beschaffen sein, dass der Roboter ihn abfahren kann. Wie in Kapitel 2.1 beschrieben, unterliegen differentielle zweiräderige Roboter gewissen Einschränkungen, die von dem Algorithmus beachtet werden müssen.
- Laufzeit
  Wie lange dauert die Planung eines Weges?
- Speicherplatz
  Wieviel Speicherplatz wird für die Planung benötigt?

- proaktiv / reaktiv
  Proaktive Algorithmen planen den Weg vorausschauend. Im Extremfall wird der gesamte Weg vom Start bis zum Ziel geplant. Reaktive Algorithmen planen weniger vorausschauend, sind dadurch oftmals schneller als proaktive Algorithmen, können jedoch oft keine Vollständigkeit garantieren.

- Ausrichtung und Geschwindigkeit am Ziel
  Für gewisse Anwendungsfälle ist es notwendig, dass der Roboter am Ziel mit einer bestimmen Ausrichtung ankommt. Im Roboterfußball ist es sogar notwendig, dass der Roboter am Ziel mit einer definierten Geschwindigkeit und einer bestimmten Ausrichtung ankommt, z.B. um den Ball in eine gewünschte Richtung spielen zu können.

Im folgenden werden verschiedene Algorithmen zur Wegplanung vorgestellt. Es lassen sich die allgemeinen und die spezialisierten Algorithmen unterscheiden. Die allgemeinen Algorithmen (Straßenkarten-Methode, Segmentierung des Raumes und Potenzialfeld-Methode) sind allgemeine Lösungsansätze für die Wegplanung. Sie berechnen jedoch Wege, ohne die Einschränkungen von differentiellen zweiräderigen Robotern zu berücksichtigen und kommen deshalb bei diesen Roboter nicht zum Einsatz. Die spezialisierten Algorithmen (Dreh-Fahr-Dreh, CMU, VFH+, S-Kurve) kommen bei diesen Robotern zum Einsatz und berücksichtigen die entsprechenden Einschränkungen.

Die Potenzialfeld-Methode und der CMU-Algorithmus sind jedoch zwei Sonderfälle. Die Potenzialfeld-Methode kommt bei diesem Robotertyp zum Einsatz, obwohl die Einschränkungen dieser Roboter nicht berücksichtigt werden. Der CMU-Algorithmus ist für differentielle zweiräderige Roboter entwickelt worden, aber auch er berücksichtigt nicht die Einschränkungen dieser Roboter. In beiden Fällen muss entweder der Weg an die Einschränkungen des Roboter angepasst werden oder der Roboter muss den Weg oder Teile des Weges mit einer sehr niedrigen Geschwindigkeit abfahren. Beide Möglichkeiten führen jedoch nicht unbedingt zum Erfolg.

Weitere Informationen finden sich in [SN04, S.261].

5.1 Straßenkarten-Methoden

Bei dieser Klasse von Algorithmen wird aus dem Weltmodell ein Graph erzeugt. Hierbei wird das zweidimensionale Weltmodell auf einen Graph aus Linien reduziert. Wegplanung beschränkt sich in diesem Fall auf die Suche einer Folge von Kanten, die die Start- mit der Zielposition verbindet. Die verschiedenen Algorithmen unterscheiden sich in der Art, wie der Graph erzeugt wird. Bei der Auswahl der Kanten, aus denen der Graph besteht, fließen die Kriterien für einen optimalen Weg ein.

5.1.1 Voronoi Diagramm

Bei Voronoi Diagrammen hält der Roboter den maximal möglichen Abstand zu den Hindernissen ein. Der gesamte Raum wird in disjunkte Polygone zerlegt, wobei jedes Polygon genau ein Hindernis enthält. Ein Punkt des Raumes gehört zu einem Polygon genau dann, wenn der Abstand zwischen dem Punkt und dem dazugehörigen Hindernis minimal ist. Abbildung 5.1 zeigt ein Voronoi Diagramm. Hindernisse sind durch Punkte symbolisiert. In der Menge der Polygon-Kanten wird anschließend ein Weg vom Start zum Endpunkt gesucht.

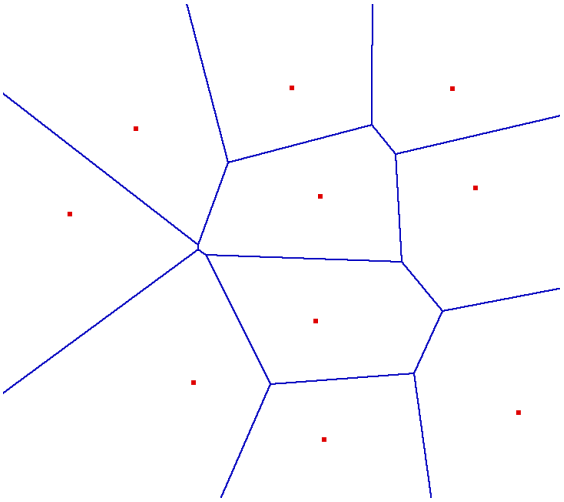

Abbildung 5.1: Voronoi Diagramm

5.1.2 Sichtbarkeitsgraph

Alle Ecken der Hindernisse sowie Start- und Zielposition werden durch Kanten miteinander verbunden. Kanten, die Hindernisse schneiden, werden anschließend entfernt (s. Abbildung 5.2). Anschließend wird in der verbleibenden Kanten-Menge eine Folge von Kanten gesucht, die die Start- mit der Zielposition verbindet. Nachteil dieses Ansatzes ist, dass der Robotern den Hindernissen so nah wie möglich kommt, was zu Problemen bei ungenauen Sensoren führen kann.

5.2 Segmentierung des Raumes

Die Grundidee der Segmentierung des Raumes (engl. Cell decomposition) ist, dass das Weltmodell in Zellen eingeteilt wird, die als frei oder belegt markiert werden. In den freien Zellen wird danach nach einem Pfad gesucht.

Nach [SN04, S. 264] lässt sich die Funktionweise folgendermaßen beschreiben:

- Einteilung das gesamten Weltmodells in disjunkte Zellen.
- Konstruktion eines Graphen.
  Die Knoten repräsentieren freie Zellen. Zwei Knoten sind genau dann adjazent, wenn die beiden dazugehörigen Zellen nebeneinander liegen.
- Bestimmung einer Menge von Knoten und Kanten, die einen Graph von der Startzelle bis zur Zielzelle bilden.
  Startzelle und Zielzelle sind die Zellen, in denen der Start- bzw. der Endpunkt liegt.
- Konstruktion eines Weges, der durch die Zellen geht, die im vorherigen Schritt ausgewählt wurden.

Abbildung 5.3 zeigt ein Beispiel für eine Wegplanung mit diesem Algorithmus. Die Hindernisse sind schwarz eingezeichnet und die nicht freien Zellen grau.

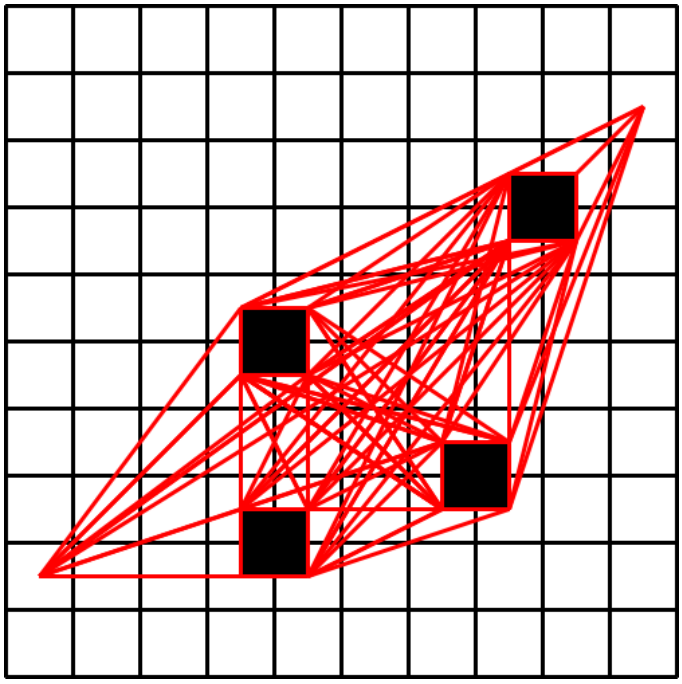

Abbildung 5.2: Sichtbarkeitsgraph

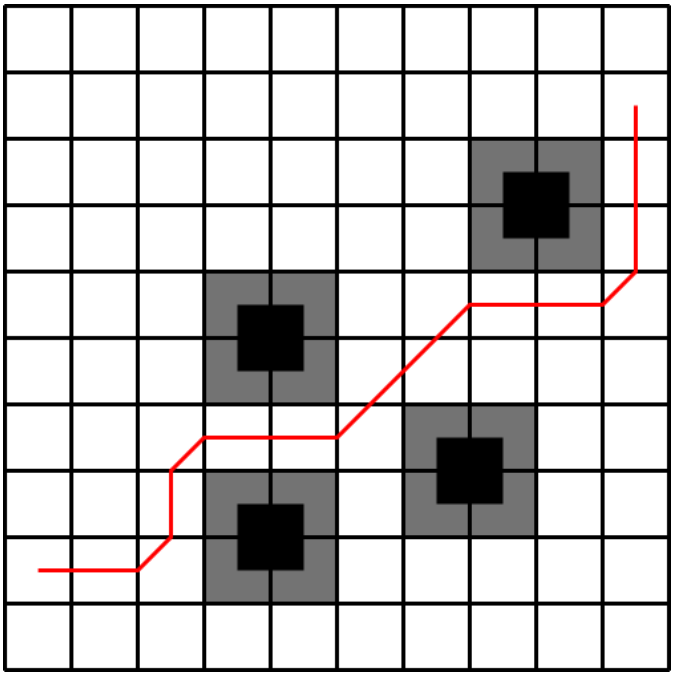

Abbildung 5.3: Segmentierung des Raumes

5.3 Potentialfeld-Methode

Potentialfelder sind Spezialfälle eines Vektorfeldes. Ein Vektorfeld ist eine Funktion, die für jeden Punkt eines Raumes einen Vektor beschreibt. Genutzt werden Potentialfelder z.B. in der Physik zur Beschreibung von elektrischen Feldern. Erstmalig wurde die Adaptierung dieses Ansatzes für mobile Roboter in [Kha86] beschrieben. Jedem Hindernis wird hierbei ein repulsives Potentialfeld zu geordnet, also ein Potentialfeld, welches vom Objekt weg zeigt. Das Ziel erhält ein attraktives Potentialfeld, also ein zum Ziel hingerichtetes. Abbildung 5.4 zeigt die Feldstärken innerhalb eines Potentialfelds. Der

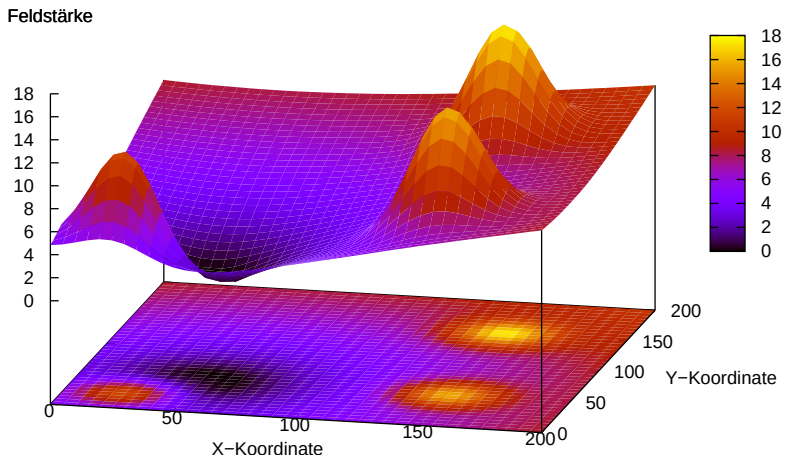

Abbildung 5.4: Feldstärke innerhalb eines Potentialfeldes

Zielpunkt hat ein Potenzial von 0, während die Feldstärken in Bereich der drei Hindernisse lokale Maxima bilden.

Koren und Borenstein beschreiben in [Kor91] die Probleme dieses Algorithmus. Der Roboter kann in kontinuierliche Schwingungen geraten, wenn ein Hindernis in der Nähe ist oder wenn er einen schmalen Gang befährt. Die Vollständigkeit des Algorithmuses ist nicht gegeben. Zum einen, da sich in dem Potentialfeld lokale Minima bilden können, aus denen der Roboter keinen Ausweg findet, zum anderen weil der Algorithmus unter ungünstigen Umständen keinen Weg zwischen zwei eng beieinander liegenden Hindernissen findet.

5.4 Dreh-Fahr-Dreh-Methode

Dieser Algorithmus ist der einfachste Ansatz, um einen differentiellen zweirädrigen Roboter von einer Startposition zu einer Zielposition fahren zu lassen. An der Startposition dreht sich der Roboter im Stand in Richtung des Zieles, fährt anschließend eine Gerade und hält an der Zielposition. Sollte eine bestimmte Ausrichtung am Ziel gewünscht sein, dreht der Roboter sich anschließend auf die gewünschte Ausrichtung.

Dieser Algorithmus ermöglichst es nicht, dass der Roboter an der Zielposition eine bestimmte Geschwindigkeit in Verbindung mit einer bestimmten Ausrichtung erreicht. Außerdem bietet er ohne Erweiterungen keine Kollisionsvermeidung.

5.5 CMU-Methode

Der in [VSHA98] und in [VBA+99, S. 88ff] vorgestellte Algorithmus ist speziell für einen differentiellen zweirädrigen Roboter konzipiert. Dieser reaktive Algorithmus berechnet in jedem Steuerungsschritt aus der relativen Position zum Zielpunkt und der gewünschten Ausrichtung am Zielpunkt einen Winkel, den der Roboter einschlagen soll. Abbildung 5.5 verdeutlicht die Idee. Sei (x, y, ϕ) die aktuelle Konfiguration des Roboters und (x^*, y^*, θ^*)

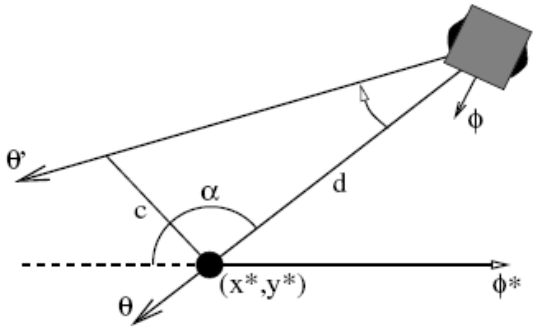

Abbildung 5.5: Der CMU-Algorithmus (Quelle: [VSHA98])

die Zielkonfiguration, dann errechnet sich die neue Ausrichtung ϕ', die der Roboter einschlagen soll, mit Hilfe der folgenden Gleichung.

$$\phi' = \phi + \min\left(\alpha, \tan^{-1}\left(\frac{c}{d}\right)\right) \tag{5.1}$$

Hierbei ist d der Abstand zum Zielpunkt und α der Winkel zwischen der Richtung, in welcher der Zielpunkt liegt und θ^*. Der Parameter c wird als „clearance parameter" bezeichnet und gibt an, welchen Abstand der Roboter zum Zielpunkt haben soll, bevor er in Richtung Zielposition einschwenkt.

Die Kollisionsvermeidung passt ϕ' bei Annäherung an ein Hindernis so an, dass die neue Ausrichtung an dem Hindernis vorbeiführt.

Obwohl der Algorithmus bei differentiellen zweirädrigen Robotern zu Einsatz kommt, berücksichtigt dieser Algorithmus nicht die Einschränkungen dieser Roboter und kann deshalb Wege zurückliefern, die nicht fahrbar sind.

5.6 S-Kurven-Methode

Die Grundidee dieses Algorithmus geht auf [Dub57] und [RS90] zurück. Während die erste Arbeit differentielle zweirädrige Roboter zum Gegenstand hat, die sich nur in eine Richtung bewegen können, ist die zweite Arbeit eine Erweiterung auf Roboter, die sowohl vorwärts als auch rückwärts fahren können. Die Idee hinter beiden Ansätzen ist, dass sich der Weg aus Strecken und Kreisbögen zusammen setzt. Abbildung 5.6 verdeutlicht diese Grundidee an einem einfachen Beispiel. Der Weg setzt sich aus einer Strecke und aus 2 Kreisbögen zusammen. Die Krümmung der Kreisbögen ist abhängig von der Geschwindigkeit des Roboters (vgl. Kapitel 2.1.3). Die Übergänge zwischen der Strecke und den Kreisbögen sind jeweils tangential.

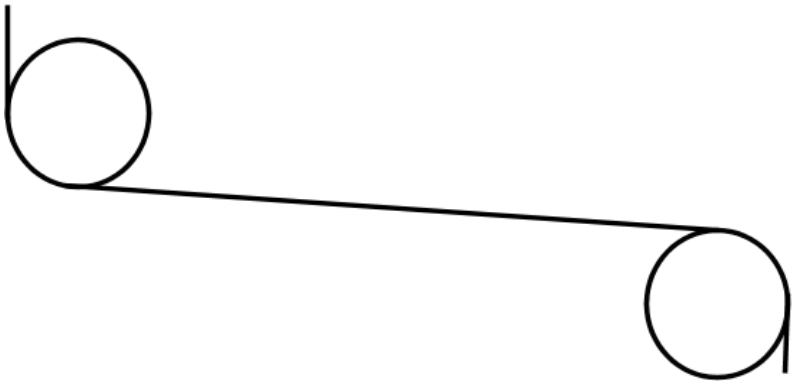

Abbildung 5.6: Die S-Kurven Methode (Quelle: Tobias Jäger)

Der Algorithmus ist um eine Kollisionsvermeidung erweiterbar. Um jedes Hindernis wird ein Kreis gelegt, dessen Radius wie beim Start- bzw. Zielkreis berechnet wird. Schneidet die Strecke den Kreis des Hindernisses, wird die zu fahrende Strecke in zwei Strecken und einem Kreisbogen aufgeteilt. Vom ersten Kreisbogen wird der Kreis um das Hindernisse tangential angefahren und soweit befahren, bis er wieder tangential in Richtung des letzten Kreisbogens verlassen werden kann. Weitere Hindernisse werden analog behandelt.

Für das in Kapitel 6 beschriebene Testsystem wurde dieser Ansatz ohne Kollisionsvermeidung erstmalig im Rahmen der Projektgruppe 416 (s. [Pro03, S. 132] und [BFH+03]) implementiert. Später wurde der Algorithmus von studentischen Hilfskräften neu implementiert. Die aktuelle Implementierung umfasst ca. 4000 Zeilen C++-Code.

5.7 Vector Field Histogram+-Methode

Bei der Vector Field Histogram+ Methode (VFH+, s. [UB98]) handelt es sich um ein Verfahren für eine lokale Kollisionsvermeidung bei mobilen Robotern, das von Ulrich und Borenstein aus dem Vorläufer Vector Field Histogram (VFH, s. [BK91]) entwickelt wurde und z.B. zusammen mit dem Dreh-Fahr-Dreh Algorithmus (s. Kapitel 5.4) eingesetzt werden kann. Konzipiert wurde der Algorithmus in Verbindung mit Ultra-Schall Sensoren, kann jedoch auch eingesetzt werden, wenn aus anderen Quellen die Positionen der Hindernisse bekannt sind.

Im ersten Schritt wird ein *Primary Polar Histogram* erstellt. Hierzu wird das Gebiet um dem Roboter in Kreissektoren aufgeteilt und im *Primary Polar Histogram* wird für jeden Sektor die Hindernisdichte in diesem Sektor gespeichert. Sektoren ohne Hindernisse erhalten den Wert Null. Zur einfacheren Implementierung wird die Größe des Roboters nicht berücksichtigt. Stattdessen werden die Hindernisse um die Größe des Roboters vergrößert.

Aus dem *Primary Polar Histogram* wird das *Binary Polar Histogram* erstellt. Das *Binary Polar Histogram* zeigt, welche Sektoren frei und welche blockiert sind. Die Entscheidung, ob ein Sektor blockiert ist, wird auf Basis der Werte des Primary Polar Histogram getroffen. Unterhalb eines einstellbaren Schwellenwertes gelten die jeweiligen Sektoren als frei, über diesem Schwellenwert als blockiert.

Im nächsten Schritt wird das *Masked Polar Histogram* unter Berücksichtigung der kinematischen und dynamischen Eigenschaften des Roboters (s. Kapitel 2.1) erstellt. Es

wird angenommen, dass der Roboter keine Kurve mit einer Krümmung größer als ein bestimmter Wert fahren kann (s. Kapitel 2.1.3) und dass diese Krümmung abhängig von der Geschwindigkeit des Roboters ist. In der Abbildung 5.7 werden die Kurven mit ma-

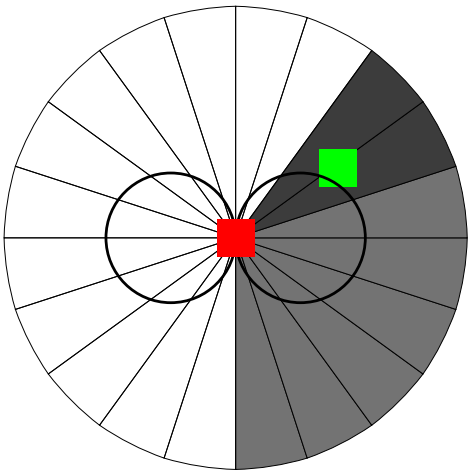

Abbildung 5.7: Blockierte Segmente im VFH+-Algorithmus

ximal möglicher Krümmung durch Kreise dargestellt. Sollte einer dieser beiden Kreise ein Hindernisse schneiden, werden alle Sektoren jenseits des jeweiligen Hindernisses auch als blockiert markiert, da ein Versuch, einen dieser Sektoren zu erreichen, mit einer Kollision zwischen dem Roboter und dem Hindernis enden würde. In der Abbildung sind diese Sektoren hellgrau markiert. Die dunkelgrau markierten Sektoren sind durch das Hindernis blockiert.

In einem letzten Schritt entscheidet eine Kostenfunktion, welche nicht blockierten Sektoren der Roboter befahren soll. In die Kostenfunktion gehen unter anderem die Faktoren ein, in welche Richtung der Roboter aktuell fährt und in welche Richtung das Ziel liegt, und ist je nach Aufgabenstellung anpassbar.

5.8 Zusammenfassung

Tabelle 5.1 fasst die Eigenschaften der verschiedenen Anfahrtsalgorithmen zusammen.

	Straßen-karten-Methode	Segmen-tierung des Raumes	Potentialfeld-Methode	Dreh-Fahr-Dreh	Dreh-Fahr-Dreh mit VFH+	CMU	S-Kurve
Kollisions-vermeidung	√	√	√	-	√	√	√
Ausrichtung am Ziel	-	-	-	√	√	√	√
Ausrichtung und Geschwindigkeit am Ziel	-	-	-	-	-	√	√
Berücksichtigung der Robotereigen-schaften	-	-	-	√	√	-	√
Implementierungs-aufwand	○	○	○	-	○	-	+
Rechenaufwand	+	+	-	-	○	-	+
Speicherbedarf	+	+	-	-	-	-	-
Vollständigkeit	√	√	-	-	-	-	-

√: vorhandene Eigenschaft - : fehlende Eigenschaft
+: hoher Aufwand/Bedarf ○: mittlerer Aufwand/Bedarf - : niedriger Aufwand/Bedarf

Tabelle 5.1: Übersicht über die Anfahrtsalgorithmen

6 Testumgebung

> „By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team." [Rob06a]
>
> „Soccer robotics is an emerging field that combines artificial intelligence and mobile robotics with the popular sport of soccer." [KKKS04]

Als Testumgebung dieser Arbeit kommt das bereits am Lehrstuhl Informatik I vorhandene System für das Roboterfußball-Projekt zum Einsatz.

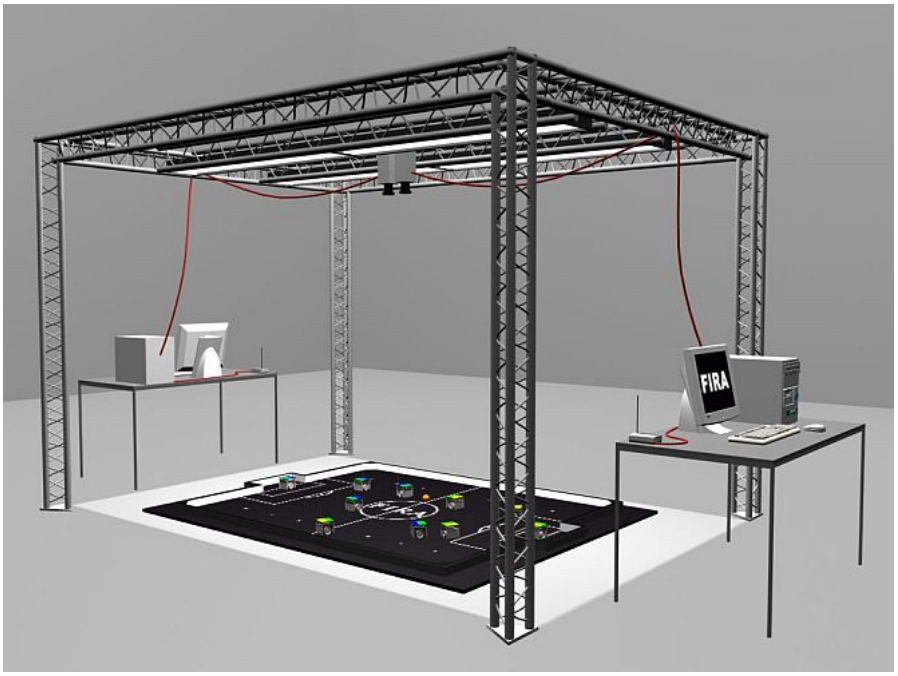

Abbildung 6.1: Typischer Aufbau beim Roboterfußball (Quelle: Marco Wickrath)

Abbildung 6.1 zeigt den typischen Aufbau beim Roboterfußball. Die Steuerung der Roboter erfolgt zentral über ein Hostsystem. Dieses Hostsystem erfasst das Spielfeld mit Hilfe einer oder mehrerer Kameras, die über dem Spielfeld montiert sind. Zur Erleichterung der Erkennung der Roboter sind diese mit Farbmarkierungen ausgestattet. Nachdem die Bildverarbeitung die Position der Roboter bestimmt hat, wird für jeden Roboter die nächste Aktion festgelegt. Entsprechend dieser Aktionen werden via Funk Befehle an die Roboter gesendet. Die Roboter setzen diese Befehle danach um.

In den folgenden Abschnitten werden der Roboter, der Funk und die Bildverarbeitung ausführlich dargestellt.

6.1 Der Roboter

Der am Lehrstuhl I eingesetzte Roboter wurde unter dem Aspekt entwickelt, dass er im Roboterfußball in der FIRA MiroSot Liga eingesetzt werden kann. In den Voraussetzungen für diese Liga liegt es begründet, dass der Roboter eine kubische Form mit einer Kantenlänge von 7,5 cm hat. Um den Ball besser führen zu können, verfügt der Roboter über eine Ballführung an der Vorderseite. Er verfügt über zwei Radencoder, einen Beschleunigungssensor und einen Kompasssensor, von denen zur Zeit nur die beiden Radencoder genutzt werden.

6.1.1 Prozessor

In dem verwendeten Roboter kommt als Prozessor der TMS320F2812 von Texas Instruments (s. [TI05]) zum Einsatz. Dieser Prozessor ist speziell für Steuerungsaufgaben konzipiert und kann mit bis zu 150 MHz betrieben werden. Er verfügt über 18 KiW Speicher und 128 KiW Flashspeicher.

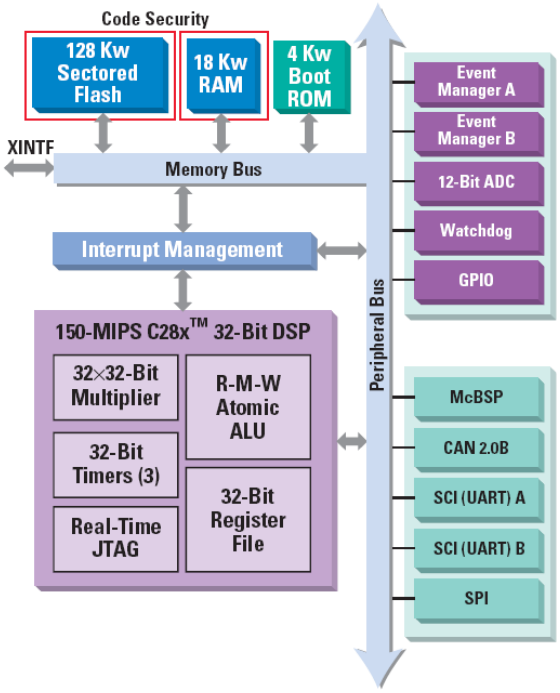

Abbildung 6.2: Schematischer Aufbau des TMS320F2812 (Quelle: [TI02, S. 2])

6.1.2 Mathematische Operationen

Der TMS320F2812 verfügt nicht über eine Gleitkommaeinheit. Aus diesem Grund sind Berechnungen von Gleitkommazahlen relativ langsam, wie die Messungen aus Anhang A zeigen.

Texas Instruments empfiehlt deshalb die Nutzung von Festkommazahlen und stellt eine Software-Bibliothek mit dem Namen IQ-Math (s. [TI03]) zur Verfügung, die die Nutzung von Festkommazahlen erleichtern soll.

Bei Festkommazahlen ist immer eine feste Anzahl von Bits für die Vor- und Nachkommastellen reserviert. Gebräuchliche Standards sind: Q N.M Format, Komplementdarstellung und 2er-Komplementdarstellung.

Fehleranfällig ist die Bestimmung, wieviele Vor- bzw. Nachkommastellen für eine Berechnung notwendig sind. Ist die Anzahl der Vorkommastellen zu klein, kommt es zu Überläufen, weil die Zahlen sich mit der Anzahl an Stellen nicht mehr darstellen lassen. Ist die Anzahl der Nachkommastellen zu klein, sind die Berechnung ggf. zu ungenau.

Bei Gleitkommazahlen gibt es keine feste Anzahl an Vor- und Nachkommastellen. Stattdessen ist die Anzahl an Bits für den Exponent und für die Mantisse festgelegt. Der Standard IEEE-754 ([AI85]) beschreibt den Aufbau einer Gleitkommazahlen. Eine 32-Bit Gleitkommazahl ist nach diesem Standard folgendermaßen aufgebaut:

Bits	Zweck
31	Vorzeichen (v)
30-23	Exponent (e)
22-0	Mantisse (M)

Für die Interpretation dieser Darstellung wird eine Fallunterscheidung vorgenommen, wobei Einträge, die mit * markiert sind, für den jeweiligen Fall nicht beachtet werden:

Exponent	Mantisse	Interpretation
$= 255$	$\neq 0$	Keine Zahl
$= 255$	$= 0$	$(-1^v) * \infty$
$0 < e < 255$	*	$(-1^v) * 2^{e-127} * (1.M)$
0	$\neq 0$	$(-1^v) * 2^{-126} * (0.M)$
0	0	$(-1^v) * 0$

Der Wertebereich der Gleitkommazahl ist wesentlich größer als bei Festkommazahlen. Jedoch vergrößert sich der Abstand zwischen zwei darstellebaren Zahlen, je größer die darzustellende Zahl wird. [TI04] führt als Beispiel die Addition 10.000000000 + 0.000000238 an. In der benutzten Gleitkommadarstellung gibt es zwischen 10.000000000 und 10.000000950 keine weitere, darstellbare Zahl, so dass sich als Summe 10.000000000 ergibt.

Während der darstellbare Zahlenbereich auf Grund der festen Anzahl von Vor- und Nachkommastellen bei Festkommazahlen konstant ist, ist er bei Gleitkommazahlen dynamisch. Auch ist der Abstand zwischen zwei darstellbaren Zahlen bei Festkommazahlen konstant, während er bei Gleitkommazahlen abhängig von der darzustellenden Zahl ist.

Vor der Benutzung der IQ-Math Bibliothek muss deshalb der für das Problem passende Datentyp (s. Tabelle 6.1) ausgewählt werden. Hierbei ist zu beachten, dass der Wertebereich des Datentypes so gewählt sein muss, dass das Ergebnis sowie alle Zwischenergebnisse innerhalb des Wertebereiches liegen. Gleichzeitig muss die Genauigkeit ausreichend sein.

Datentyp	kleinste darstellebare Zahl	größte darstellbare Zahl	Genauigkeit
IQ1	-1073741824.000000000	1073741823.500000000	0.500000000
IQ2	-536870912.000000000	536870911.750000000	0.250000000
IQ3	-268435456.000000000	268435455.875000000	0.125000000
IQ4	-134217728.000000000	134217727.937500000	0.062500000
IQ5	-67108864.000000000	67108863.968750000	0.031250000
IQ6	-33554432.000000000	33554431.984375000	0.015625000
IQ7	-16777216.000000000	16777215.992187500	0.007812500
IQ8	-8388608.000000000	8388607.996093750	0.003906250
IQ9	-4194304.000000000	4194303.998046875	0.001953125
IQ10	-2097152.000000000	2097151.999023438	0.000976563
IQ11	-1048576.000000000	1048575.999511719	0.000488281
IQ12	-524288.000000000	524287.999755859	0.000244141
IQ13	-262144.000000000	262143.999877930	0.000122070
IQ14	-131072.000000000	131071.999938965	0.000061035
IQ15	-65536.000000000	65535.999969482	0.000030518
IQ16	-32768.000000000	32767.999984741	0.000015259
IQ17	-16384.000000000	16383.999992371	0.000007629
IQ18	-8192.000000000	8191.999996185	0.000003815
IQ19	-4096.000000000	4095.999998093	0.000001907
IQ20	-2048.000000000	2047.999999046	0.000000954
IQ21	-1024.000000000	1023.999999523	0.000000477
IQ22	-512.000000000	511.999999762	0.000000238
IQ23	-256.000000000	255.999999881	0.000000119
IQ24	-128.000000000	127.999999940	0.000000060
IQ25	-64.000000000	63.999999970	0.000000030
IQ26	-32.000000000	31.999999985	0.000000015
IQ27	-16.000000000	15.999999993	0.000000007
IQ28	-8.000000000	7.999999996	0.000000004
IQ29	-4.000000000	3.999999998	0.000000002
IQ30	-2.000000000	1.999999999	0.000000001

Tabelle 6.1: Datentypen der IQ-Math Bibliothek

Die folgenden Codezeilen zeigen, wie eine normale Gleitkommaberechnung bei der Benutzung der IQ-Math in C bzw. C++ aussehen.

Gleitkommaberechnung mit floats:

```
01| float a,b,c;
02|
03| a = b + c * sin( 1.1 );
```

Implementierung in C bei Benutzung von IQ-Math:

```
01| _iq a,b,c;
02|
03| a = _IQmpy( b + c, _IQsin(_IQ(1.1));
```

Implementierung in C++ bei Benutzung von IQ-Math:

```
01| iq a,b,c;
02|
03| a = b + c * IQsin(IQ(1.1));
```

6.1.3 Speicher

Der Prozessor verfügt insgesamt über 18 KiW SARAM (Single Access Random Access Memory), das aus einem Bereich der Größe 2 KiW und aus zwei Bereichen der Größe 8 KiW besteht. Laut [TI02, S. 2] erreichen nur Programme, die aus diesen Speicherbereichen ausgeführt werden, die maximale Geschwindigkeit von 150 MIPS. Der eingesetze Roboter verfügt außerdem über 512 KiW externes SRAM (Static Random Access Memory), das über das „External Interface" (s. [TI05, S. 35]) des Prozessors angebunden ist. Programme, die in diesem Bereich abgelegt sind, laufen jedoch langsamer. Da genaue Informationen nicht verfügbar waren, entschied sich der Autor, die Geschwindigkeit zu messen. Code, der aus dem externen RAM ausgeführt wird, läuft um den Faktor 20 langsamer, als wenn der gleiche Code aus dem internen Speicher ausgeführt wird. Anhang A zeigt die genauen Ergebnisse.

Der Roboter verfügt außerdem über 128 KiW Flashspeicher. Programme, die aus diesem nicht-flüchtigen Speicher ausgeführt werden, erreichen 110 bis 120 MIPS.

6.2 globale Bildverarbeitung

Im Laufe des Projekts wurden mehrere Bildverarbeitungen entwickelt. Die ursprüngliche Version der sich im Einsatz befindlichen Bildverarbeitung, wurde von Norman Weiss in Rahmen seiner Diplomarbeit ([WJ04]) entwickelt. Im Laufe der Zeit wurde diese Version von studentischen Hilfskräften und Projektgruppen (s. [Pro03, S. 42ff], [Pro05, S. 9ff], [Pro06, S. 56ff]) weiterentwickelt.

Mit Hilfe einer oder zweier Kameras, die über dem Spielfeld montiert sind, bestimmt die Bildverarbeitung die Position der eigenen und der gegnerischen Roboter auf dem Spielfeld. Um die Erkennung der Roboter zu erleichtern sind diese mit Farbmarkierungen auf der Oberseite ausgestattet.

Die Erkennung von Objekten läuft in mehreren Schritten ab:

- Im Bild der Kamera (s. Abbildung 6.3) wird nach zusammenhängenden Farbflächen (s. Abbildung 6.4) gesucht.
- In Abhängigkeit von der Anzahl der Farbmarkierungen auf dem Roboter, wird eine oder mehrere zusammenliegende Farbmarkierungen einem Roboter zugewiesen.
- Die Konfiguration der Roboter wird unter Berücksichtigung von Verzerrungen (s. Kapitel 3.4 und [Pro06, S. 56]) berechnet.

Abbildung 6.3: Kamerabild

Abbildung 6.4: Zusammenhängende Farbflächen im Kamerabild

6.3 lokale Bildverarbeitung

Simon Schulz entwickelte im Rahmen seiner Diplomarbeit ein Bildverarbeitungssystem für den Roboter. Das System besteht aus 4 Kameras mit einer Auflösung von je 640x480 Bildpunkten. Ein Prozessor und ein FPGA (field programmable gate array, programmierbarer Logikbaustein) übernehmen die Bildverarbeitung. Während der FPGA XC3S400 aus der Spartan 3-Serie der Firma Xilinx die Ansteuerung der Kameras und die Einteilung der Bildpunkte in die verschiedenen Farbklassen übernimmt, kommt für die Suche von Farbflächen ein TMS320F2812 zum Einsatz. Außerhalb des Spielfeldes werden Farbmarkierungen aufgestellt, mit dessen Hilfe die Bildverarbeitung die Position des Roboter bestimmen kann. Die errechnete Position wird über eine serielle Schnittstelle an den Prozessor übermittelt.

Dem Autor dieser Arbeit stand ein Prototyp dieses Bildverarbeitungssystemes zeitweise zur Verfügung. Abbildung 6.5 zeigt, wie der Roboter um das prototypische Bildverarbeitungssystem erweitert wurde. Die endgültige Version des Bildverarbeitungssystem, das komplett in den Roboter integriert ist, und weitergehende Informationen finden sich in der Diplomarbeit von Simon Schulz ([Sch06]).

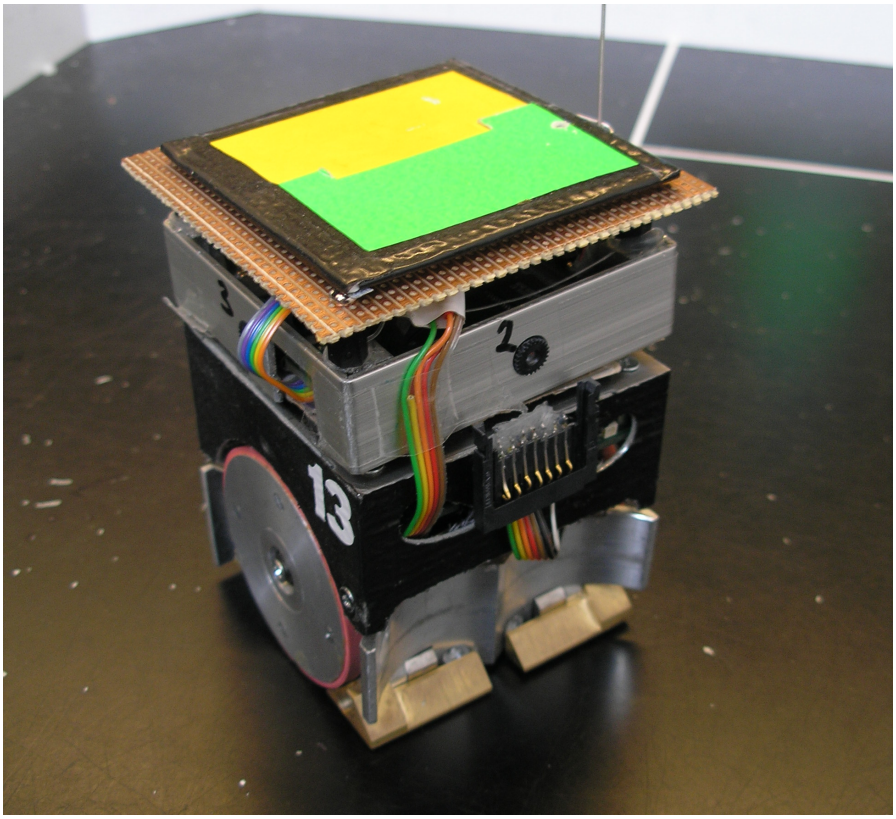

Abbildung 6.5: Roboter mit dem lokalen Bildverarbeitungssystem

6.4 Simulator

Die Hauptaufgabe des in das Hostsystem integrierten Simulators war das Testen der Roboterfußball-Strategie. Die Simulation des Roboterverhaltens stützt sich auf Teile der Software, die auf dem Roboters läuft. Die physikalischen Gegebenheiten werden durch sehr vereinfachte Modelle simuliert. Insgesamt differiert das Verhalten zwischen Realität und Simulation im Bereich der Simulation von Kollisionen und der Simulation des Ballverhaltens so sehr, dass der Simulator zum Testen der Strategie des Roboterfußballes nicht geeignet ist. Diverse Änderungen am Hostsystem führten dazu, dass der Simulator mittlerweile nicht mehr funktionsfähig ist.

6.5 Funk

Für die Kommunikation mit dem Roboter stehen zwei verschiedene Funksysteme zur Verfügung. Das erste System, „Bim", basiert auf Transceivern der Firma Radiometrix, während das zweite System, „Yab", eine Entwicklung der Projektgruppe 449 ist. Beide Systeme werden über eine serielle Verbindung angesprochen und sind für eine bidirektionale Übertragung ausgelegt, jedoch nur im halbduplex Modus. Das bedeutet, dass ein Modul entweder Senden oder Empfangen kann. Beides gleichzeitig ist jedoch nicht möglich.

6.5.1 Bim

Die Bim Module senden je nach Version auf einer von vier verschiedenen Frequenzen (418, 433, 869, 914 MHz) und haben eine Netto-Übertragungsrate von bis zu 160 KiBit/s. Da das Verhältnis zwischen gesendeten Nullen und gesendeten Einsen gleich sein muss, ist eine Codierung der Daten notwendig. Es bietet sich an, eine Manchesterkodierung zu nutzen. Hierbei wird für jedes Bit zwei Bits übertragen. Eine 1 wird z.B. durch die Übertragung von 01 repräsentiert und statt einer 0 wird 10 gesendet. Durch diese Verdoppelung der Daten können nur bis zu 80 KiBit/s Nutzdaten übertragen werden. Da auch ältere Module mit niedrigeren Übertragungsraten im Gebrauch sind, bleiben 19200 Bits/s für die Übertragung von Daten.

Weitere Informationen sind in den Datenblättern ([Rad04c, Rad04a, Rad04b]) des Herstellers zu finden.

6.5.2 Yab

Die Yab Module wurden in Rahmen der Projektgruppe 449 als Ersatz für die Bim Module entwickelt. Sie sind Pin-kompatibel zu den Bim Modulen, senden im Bereich von 2,4 bis 2,5 GHz und haben eine Datenübertragungsrate von bis zu 1 MiBit.

Weitere Informationen finden sich im Endbericht der Projektgruppe 449 ([Pro05, S. 97]).

Teil II

Realisierung

7 Roboter Software

Die Implementierungen dieser Diplomarbeit sollten auf dem in Kapitel 6 vorgestellen Roboter erfolgen. Programmiert wird dieser Roboter mit der Entwicklungsumgebung „Code Composer Studio" von Texas Instruments. Da sich der Roboter bereits im Rahmen des Roboterfußball-Projektes im Einsatz befindet und somit schon Quellcode vorhanden ist, musste entschieden werden, welche Teile des bestehenden Quellcodes für die zu lösende Aufgabe übernommen werden können. Außerdem musste entschieden werden, ob die Implementierung in Assembler, C oder in C++ stattfinden sollte. Assembler schied aufgrund des Aufwands und der Fehleranfälligkeit aus. C++ hat den Vorteil, dass sich implementiere Klassen leicht in den Simulator übernehmen lassen. Als Nachteil wurde erwartet, dass C++ Code etwas langsamer als C Code wäre.

Es wurde entschieden, dass die Implementierung in C++ erfolgen solle. Von dem bestehenden Code sollten die Routinen für die Regelung und den Hardwarezugriff übernommen werden. Kapitel 7.1 beschreibt den Entwurf und den Beginn der C++ Implementierung. Aufgrund von diversen Problemen, die in dem Unterkapitel dokumentiert sind, wurde diese Implementierung verworfen. Kapitel 7.2 beschreibt den anschließenden Entwurf und die Implementierung in C.

7.1 C++-Implementierung

Abbildung 7.1 zeigt das geplante Klassendiagramm. Nach Programmierung der ersten Klassen zeigten sich bei Tests diverse Probleme. Die Ausführungsgeschwindigkeit war nicht, wie erwartet, ein wenig, sondern deutlich geringer. Messungen ergaben einen Geschwindigkeitseinbruch von bis zu 60% im Vergleich zur vorhandenen Implementierung in C. Hinzu kamen Probleme mit der Speicherverwaltung. Während bei der Implementierung in C jede Funktion durch eine Compileranweisung in einem anderen Speicherbereich verschoben werden kann (s. Kapitel 6.1.3), besteht diese Möglichkeit bei C++ Klassen nicht. Alle implementieren Klassen müssen somit in einem der zusammenhängenden Speicherbereiche passen, was den verfügbaren Speicherplatz erheblich verkleinert. Aus diesen Gründen hat sich der Autor entschieden doch eine Implementierung in C vorzunehmen.

7.2 C-Implementierung

Abbildung 7.2 zeigt, aus welchen Modulen die C-Implementierung besteht. Die in Kapitel 2 beschriebenen Elemente eines mobilen Roboters finden sich auch in diesem Diagramm wieder. Während die Module Anfahrt, Regelung und Hardware die Bewegung des Roboter ermöglichen, ermöglichen die Module Weltmodell, Sensorik und Hardware die Wahrnehmung der Umgebung. Die Elemente Kommunikation und Handeln finden sich in den gleichnamigen Modulen.

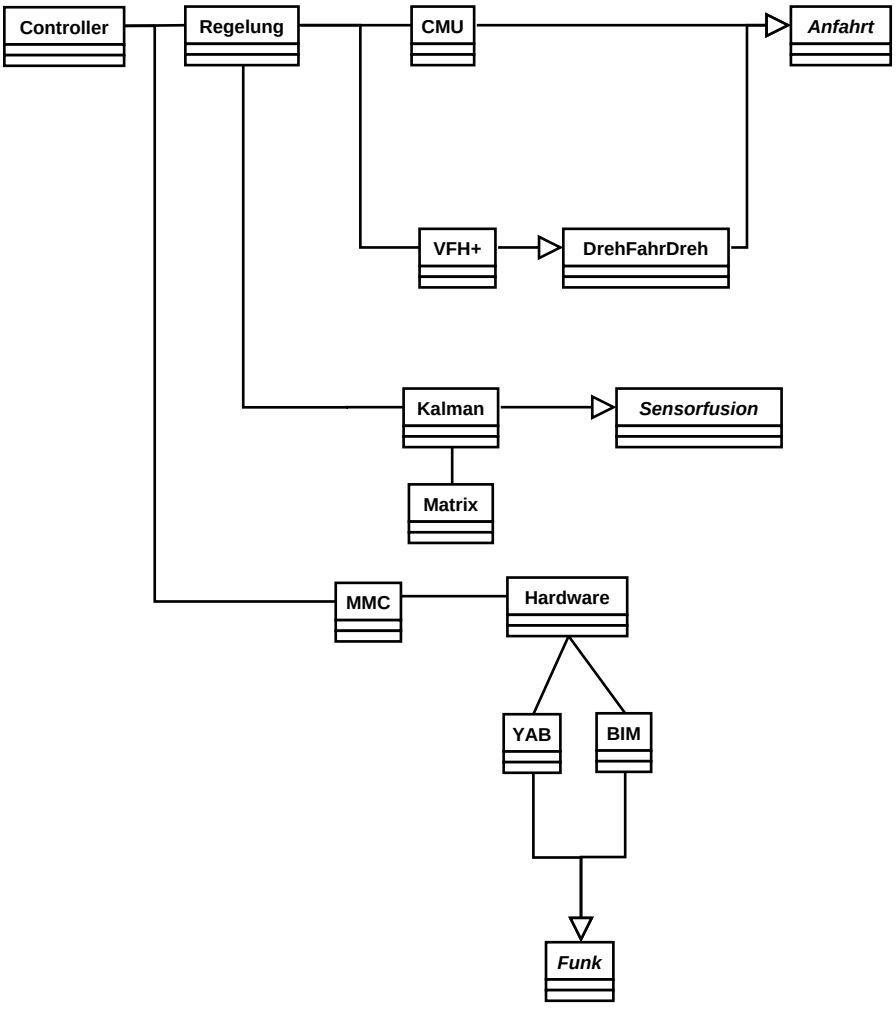

Abbildung 7.1: Klassendiagramm der Robotersoftware (C++ Entwurf)

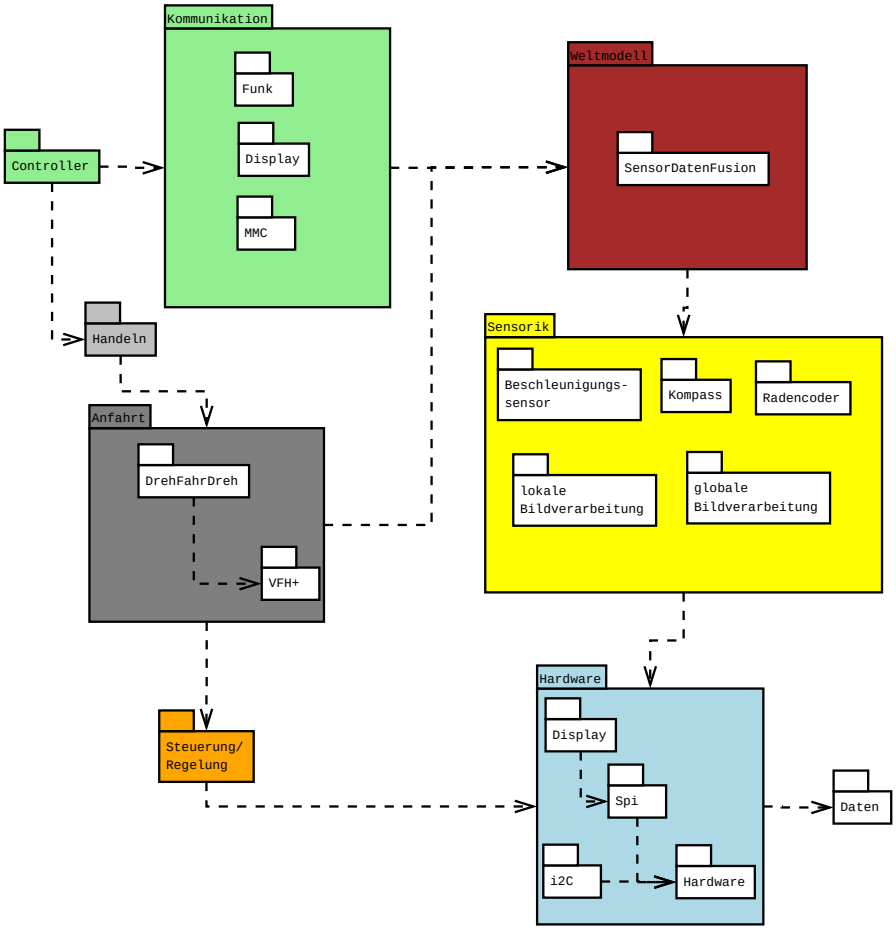

Abbildung 7.2: Module und Untermodule der Robotersoftware
Die Pfeile zeigen Abhängigkeiten zwischen den Modulen auf. Direkte Abhängigkeiten werden aus Gründen der besseren Übersicht nicht eingezeichnet, sofern bereits eine transitive Abhängigkeit besteht.

7.2.1 Controller

Dieses Modul sorgt dafür, dass jede Millisekunde der ausgewählt Befehl aufgerufen wird. Abbildung 7.3 zeigt die Abhängigkeiten der Befehle. Jeder Befehl ruft einen Befehl einer niedrigeren Ebene auf. Die unterste Ebene bildet die Regelung, die dafür sorgt, dass die gewählten Radgeschwindigkeiten eingeregelt werden. Hierzu werden die Motoren über die PWM-Ausgänge des Prozessors angesteuert.

7.2.2 Hardware

Dieses Modul enthält sämtliche Routinen, die für einen Zugriff auf die Hardware notwendig sind. Der Großteil der in diesem Modul genutzten Routinen stammt aus bereits vorhandender Software, mit Ausnahme der im folgenden dokumentierten Modulen.

7.2.2.1 SPI

Dieses Modul dient der Ansteuerung von Geräten, die mit Hilfe des Serial Peripheral Interface (SPI) an den Prozessor gekoppelt sind. Das SPI wurde von der Firma Motorola entwickelt und ist ein synchroner serieller Datenbus, an dem alle Geräte parallel angeschlossen werden. Für die Übertragung von Daten sind bei diesem Bus 3 Datenleitung ausreichend. Eine Leitung übernimmt den Datenversand (MOSI, Master out Slave in), eine weitere den Datenempfang (MISO, Master in Slave out) und eine ist für die Taktung des Busses zuständig (SCKL, Serial Clock). Um Kollisionen auf dem Bus zu vermeiden, existiert für jedes angeschlossene Gerät eine Steuerleitung (CS, Chip Select). Mit Hilfe dieser Steuerleitungen kann der Prozessor festlegen, welches Gerät gerade Daten empfangen bzw. versenden darf.

Der TMS320F2812 verfügt über eine hardwaremäßig Unterstützung des SPI Busses. Die Grundfunktionalität dieses Modules wurde aus einem Beispiel von Texas Instruments entnommen und an die Erfordernisse dieser Arbeit angepasst.

Weitere Informationen finden sich in [TI05, S. 81f].

7.2.2.2 I^2C

Ähnlich wie das SPI Modul dient dieses Modul der Ansteuerung von Geräten, die mit Hilfe des I^2C-Bus (Inter-Integrated Circuit) an den Prozessor gekoppelt sind. Der I^2C-Bus ist ebenfalls ein synchroner serieller Datenbus, an dem alle Geräte parallel angeschlossen werden. Für die Übertragung der Daten kommen 2 Datenleitungen zum Einsatz. Eine Leitung für den Takt (SCL, engl. serial clock line) und eine Leitung für die Datenübertragung (SDA, engl. serial data line). Die Datenrate liegt zwischen 100 KiBit/s und 3,4 MiBit/s. Kollisionen werden vermieden, indem jedes angeschlossene Gerät eine eindeutige Kennung trägt, die 7 oder 10 Bit lang ist, und der Prozessor bestimmt, welches Gerät senden oder empfangen kann.

Da bei dem TMS320F2812 keine hardwaremäßige Unterstützung für den I^2C Bus existiert, muss die gesamte Steuerung der I^2C Kommunikation inkl. Generierung des Taktes softwaremäßig erfolgen.

Um eine Datenübertragung zu beginnen, stellt der Prozessor die sogenannte „Start Condition" her. Hierbei wechselt die Datenleitung vom Zustand 1 zum Zustand 0, während die Taktleitung auf 1 bleibt.

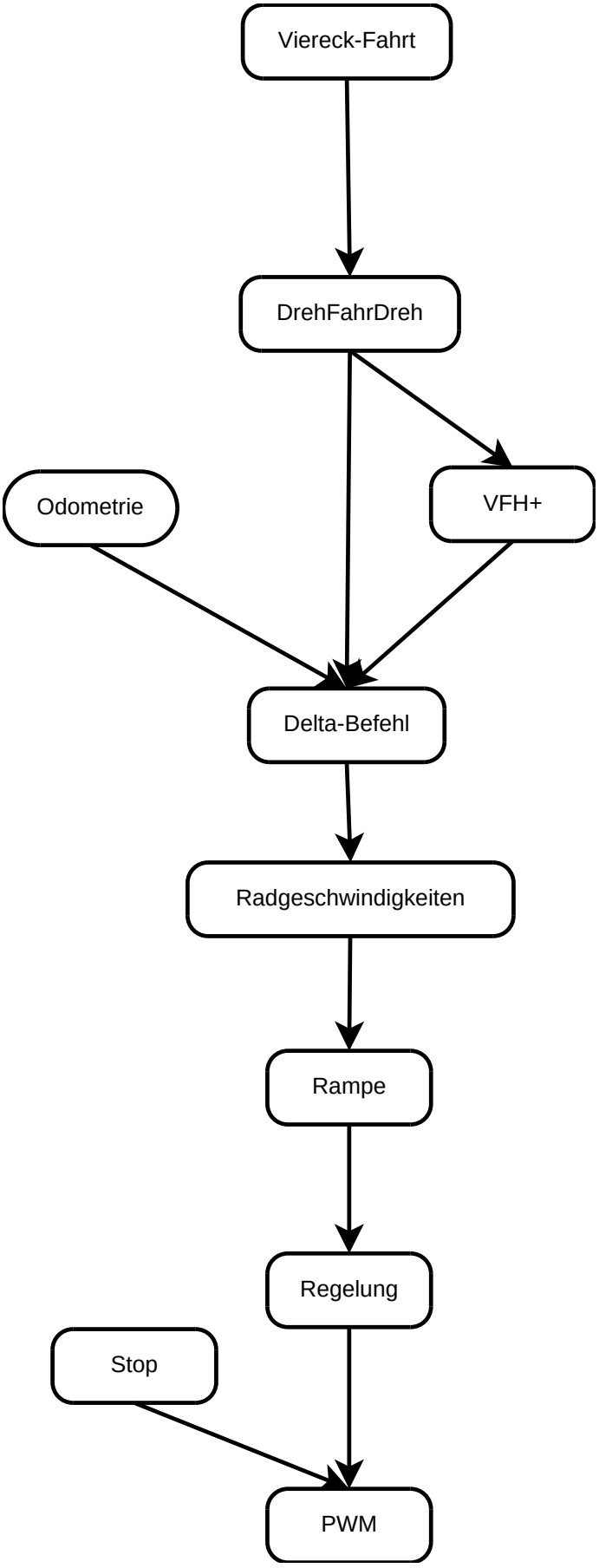

Abbildung 7.3: Abhängigkeiten der Befehle

Danach wird die 7 Bit lange Kennung des Gerätes gesendet, dass angesprochen werden soll. Ein Bit wird gesendet, indem die Taktleitung auf 0 gezogen wird, die Datenleitung entsprechend des zu sendenden Bits gesetzt wird und danach die Taktleitung auf 1 gezogen wird. Nach den 7 Adressbits wird ein weiteres Bit gesendet, um die Richtung der anschließenden Datenübertragung festzulegen. Bei einer 0 übermittelt der Prozessor Daten während er bei einer 1 Daten empfängt. Nach diesen 9 Bits empfängt der Prozessor ein Bit, das sogenannte ACK. Dieses Bit wird vom dem angesprochenen Gerät generiert, und gibt an, dass es sich angesprochen fühlt und bereit ist. Datenübertragungen zum Prozessor verlaufen analog. Der Prozessor generiert den Takt, das angesprochene Gerät ändert die Datenleitung und nach 8 Bits sendet der Prozessor ein ACK.

Weitere Informationen finden sich in [Phi01].

In der Software des Testsystemes existieren bereits Routinen zur Ansteuerung des I^2C-Busses, diese wurden jedoch nicht genutzt, da sie viele Funktionen hatten, die für diese Arbeit nicht notwendig waren und zu viel Speicher belegt hätten. Stattdessen wurden die notwendigen Routinen neu implementiert.

7.2.2.3 Display

Als Display kommt das LCD-Display der Firma Sharp aus der LS020xxx-Serie zum Einsatz, das über den SPI Bus mit dem Prozessor verbunden ist. Das Display bietet eine Auflösung von 176x132 Bildpunkten bei einer Farbtiefe von 16 Bit unter Benutzung des RGB-565 Formats. Die Ansteuerung übernimmt eine von Christian Kranz (s. [Kra05]) geschriebene Software, die von Simon Schulz sowie dem Autor dieser Arbeit an den TMS320F2812 angepasst wurde. Dieses Modul stellt nur einen rudimentären und sehr hardwarenahen Zugriff auf das Display zur Verfügung. Für den Zugriff auf das Display sollten deshalb die Funktionen aus dem Modul Kommunikation/Display (s. Kapitel 7.2.5.2) genutzt werden.

7.2.3 Weltmodell

Dieses Modul sorgt dafür, dass aus den Informationen der unterschiedlichen Sensoren, ein Weltmodell berechnet wird.

Ausführliche Informationen finden sich in Kapitel 10.

7.2.4 Sensorik

Dieses Modul stellt Routinen zum Zugriff auf die verschiedenen Sensoren zur Verfügung.

7.2.4.1 Radencoder

Die beiden Quadratur-Radencoder (s. Kapitel 3.2.1, [Fau06b]) sind in die Motoren (s. [Fau06a]) eingebaut und erzeugen pro Wellenumdrehung des Motors 512 Impulse. Da der TMS320F2812 hardwaremäßig über eine Dekodierunglogik für Quadratur-Signale (s. [TI05, S. 64]) verfügt, kann die Anzahl der Impulse für jeden Radencoder aus je einer Speicherzelle ausgelesen werden.

7.2.4.2 Kompass

Zur Ausrichtungsbestimmung kommt der magnetoresistive (s. Kapitel 3.2.2) Kompass HMC6352 (s. [Hon06]) der Firma Honeywell zum Einsatz. Der Chip ist über einen I^2C Bus (s. Kapitel 7.2.2.2) an den Prozessor angebunden und kann bis zu 20-mal in der Sekunde die Ausrichtung bestimmen. Der Sensor verfügt über einen sogenannten „Continuous Mode". Einmal in diesem Modus, sendet der Kompasssensor, eine Taktung des I^2C-Busses durch den Prozessor vorausgesetzt, kontinuierlich Daten.

Zur Abfrage des Kompasssensors wurde eine nebenläufige Implementierung gewählt, die die Ausführung des Codes nicht länger als notwendig verzögert. Innerhalb des Timer-Interrupt (s. Kapitel 7.2.2) wird pro Aufruf eine Flanke des Takts erzeugt. Je nachdem ob Daten gesendet oder empfangen werden, wird außerdem die Daten-Leitung gesetzt oder ausgelesen.

Nach dem Empfang einer neuen Messung wird diese in die globale Datenstruktur geschrieben und beim nächsten routinemäßigen Aufruf der Sensordatenfusion in die Berechnung mit einbezogen.

7.2.4.3 Beschleunigungssensor

Der Roboter verfügt über den Beschleunigungssensor LIS3L02DQ (s. [STM05]) der Firma ST-Microelectronics, der über den SPI Bus (s. Kapitel 7.2.2.1) angebunden ist. Laut Datenblatt kann der Sensor auf drei Achsen bei 1500 Messungen pro Sekunde Beschleunigungen zwischen -2 g und 2 g messen.

Auf eine nebenläufige Implementierung wie beim Kompasssensor wurde verzichtet, da die Zeit, die zur Abfrage der Messwerte nötig ist, klein genug ist, um die Ausführung des restlichen Programmes nicht merklich zu verzögern.

7.2.4.4 lokale Bildverarbeitung

Das lokale Bildverarbeitungssystem sendet die Position des Roboter mit Hilfe der seriellen Schnittstelle. Dieses Modul empfängt die Daten und rechnet sie in das genutzte Koordinatensystem um.

7.2.4.5 globale Bildverarbeitung

Dieses Modul verarbeitet die Informationen der globalen Bildverarbeitung, die per Funk an den Roboter übertragen werden. Die Hauptaufgabe ist es die übermittelten Informationen von dem auf dem Hostsystem genutzten Koordinatensystem auf das auf dem Roboter genutzten umzurechnen.

7.2.5 Kommunikation

In diesem Modul sind sämtliche Routinen zur Kommunikation vereint. Dazu gehören Routinen zur Übertragung von Daten per Funk, die Speicherung von Daten auf die MMC-Karte und die Ansteuerung des Displays.

7.2.5.1 Funk

Dieses Modul verarbeitet alle mit Hilfe der Funkmodule empfangenen Daten. Je nach eingesetzten Funkmodul wird zuerst die Manchesterkodierung dekodiert, die für die Datenübertragung bei den Bim Modulen notwendig ist. Beginnend mit dem Startbyte eines jeden Datenpakets werden insgesamt 72 Bits gespeichert und anschließend ausgewertet. Die ersten 8 Bits spezifizieren den Befehl. Die weiteren Daten im Datenpaket sind befehlsabhängig. Im folgenden werden die einzelnen Befehle und der Aufbau des Datenpaket beschrieben.

Stop
Dieser Befehl stoppt den Roboter.

Bits	Beschreibung
0-7	0

Radgeschwindigkeiten
Dieser Befehl lässt den Roboter mit den übermittelten Radgeschwindigkeiten fahren.

Bits	Beschreibung
0-7	1
8	Vorzeichen für die linke Radgeschwindigkeit (0=positiv)
9-23	linke Radgeschwindigkeit
24	Vorzeichen für die rechte Radgeschwindigkeit (0=positiv)
25-39	rechte Radgeschwindigkeit

Odometrie
Dieser Befehl startet einen Odometrielauf (s. Kapitel 9.1).

Bits	Beschreibung
0-7	2=Odometrielauf im Uhrzeigersinn, 3=gegen den Uhrzeigersinn

Position anfahren
Veranlasst den Roboter, die übermittelte Position mit dem Anfahrtsalgorithmus (s. Kapitel 11.1) anzufahren.

Bits	Beschreibung
0-7	4
8-23	X Position in mm
24-39	Y Position in mm
40-54	Ausrichtung an der Zielposition in 0,1°
55	Kollisionsvermeidung (1=aktiviert)

Daten des globalen Sensors: Roboterposition
Übermittelt die von der Bildverarbeitung erkannte Position an den Roboter.

Bits	Beschreibung
0-7	5
8-21	X Position in mm
22-35	Y Position in mm
36-50	Ausrichtung in 0,1°

Latenzzeittest: Bildverarbeitung
Startet den Latenzzeittest für die Bildverarbeitung (s. Kapitel 9.3.1).

Bits	Beschreibung
0-7	6

Daten des globalen Sensors: Hindernisse
Übermittelt die von der Bildverarbeitung erkannten Hindernisse an den Roboter.

Bits	Beschreibung
0-7	7
8-23	Nummer des Hindernisses
24-39	X-Position in mm
40-55	Y-Position in mm

Viereckfahrt
Startet eine Viereckfahrt, wie in Kapitel 8.1 beschrieben.

Bits	Beschreibung
0-7	8
8	Kollisionsvermeidung (1=aktiv)

Sensorennutzung
Legt fest, welche Sensoren zur Erstellung des Weltmodells genutzt werden sollen.

Bits	Beschreibung
0-7	9
8	Kompasssensor nutzen (0/1)
9	Beschleunigungssensor nutzen (0/1)
10	globale Bildverarbeitung nutzen (0/1)
11	lokale Bildverabeitung nutzen (0/1)

7.2.5.2 Display

Die in diesem Modul zusammengefassten Routinen ermöglichen die Ausgabe von Text sowie das Zeichnen von einfachen geometrischen Formen, wie z.B. Rechtecken. Die implementierten Routinen schreiben nicht sofort auf das Display, sondern zuerst in ein Array. Da der Zugriff auf das Array wesentlich schneller erfolgen kann als auf das Display, wird durch diesen Mechanismus sichergestellt, dass die Programmausführung nicht verzögert wird. Immer wenn der Prozessor keine anderen Aufgaben auszuführen hat, werden die Daten aus dem Array Pixel um Pixel an das Display weitergeleitet. Die Aktualisierungsgeschwindigkeit des Displays ist somit direkt abhängig von der Auslastung des Prozessors.

7.2.5.3 MMC

Diese Routinen dienen dem Zugriff auf eine Speicherkarte vom Typ MMC. Die MMC-Karte dient zur Speicherung von Debug-Informationen. Für einen Zeitraum werden in jedem Regelungstakt Informationen über den Zustand des Roboters protokolliert. Da die Speicherung von Daten auf die MMC-Karte die Programmausführung zu lange stoppen würde, werden die Daten zuerst in das SRAM gespeichert und erst nach Ablauf des Protokollierungszeitraumes auf die MMC-Karte gespeichert. Zu den gespeicherten Informationen gehören z.B. die von den Sensoren gelieferten Werte, die berechnete Position des Roboter, der Befehl, der gerade ausgeführt wird, sowie die Soll- und Istgeschwindigkeiten der Räder. Nach Notwendigkeit können weitere Informationen protokolliert werden. Um den Speicherbedarf und den Implementierungsaufwand gering zu halten, wurde auf ein Dateisystem auf der MMC-Karte verzichtet. Stattdessen werden die Daten sequenziell, beginnend mit dem ersten Block der MMC-Karte, gespeichert. Eine Auswertung der auf der MMC-Karte gespeicherten Daten kann auf einen normalen PC mit einen entsprechenden Kartenleser erfolgen. Hierzu wurde ein Programm unter Linux geschrieben, das die Rohdaten von der MMC-Karte ausliest und in eine Textdatei schreibt. Die Daten dieser Datei können dann z.B. mit Hilfe des Programmes „gnuplot" grafisch dargestellt werden.

Die benutzten Routinen zum Zugriff auf die MMC-Karte basieren auf den Routinen von Ulrich Radig (s. [Rad06]) und wurden an den TMS320F2812 angepasst. Die MMC-Karte wird über einen SPI BUS (s. Kapitel 7.2.2.1) an den Prozessor gekoppelt.

7.2.6 Handeln

In diesem Modul werden alle implementierten Beispielhandlungen zusammengefasst.

7.2.7 Anfahrt

Aufgabe dieses Moduls ist es, den Roboter von der aktuellen Konfiguration in die Zielkonfiguration zu bringen.

Ausführliche Informationen finden sind im Kapitel 11.1.

7.2.8 Steuerung/Regelung

Das Modul wurde in Teilen aus den bestehenden Sourcen übernommen. An grundlegenden Befehlen kennt die Regelung die Befehle Radgeschwindigkeiten und Delta-Winkel.

Im Modus Radgeschwindigkeiten versucht die Regelung die gewünschten Radgeschwindigkeiten einzuregeln.

Im Modus Delta-Winkel gibt es als Vorgaben eine Geschwindigkeit sowie einen Winkel. Der Winkel gibt eine Korrektur der Ausrichtung an, die schnellstmöglich eingeregelt werden soll. In Kombination mit einer Geschwindigkeit von 0 ergeben sich Ausrichtungsänderungen im Stand, während bei einer Geschwindigkeit ungleich 0 die Ausrichtungsänderungen während der Fahrt vollführt werden.

Um eine einwandfreie Regelung zu gewährleisten, muss unbedingt sicher gestellt sein, dass die Routinen der Regelung genau jede Millisekunde aufgerufen werden.

8 Hostsystem Software

Das Hostsystem erfüllt drei verschiedene Aufgaben: Steuerung der globalen Bildverarbeitung und des Robotes sowie Plattform für den Simulator.

Für diese Funktionalität wurde das bestehende Hostsystem in weiten Teilen entkernt. Erhalten blieben die Bildverarbeitung, ein Teil der Benutzeroberfläche und die OpenGL-Ausgabe. Abbildung 8.1 zeigt die Benutzeroberfläche. Die markierten Bereiche werden in den folgenden Unterkapiteln erläutert.

8.1 Robotersteuerung

Mit den Bedienelementen aus den Bereiche 1 bis 3 (s. Abbildung 8.1) können verschiedene Handlungen aufgerufen und beeinflusst werden. Mit Hilfe der Bedienelemente aus Bereich 1 kann festgelegt werden, welche Sensoren zur Erstellung des Weltmodells auf dem Roboter genutzt werden. Die Änderungen werden dem Roboter via Funk mitgeteilt (s. Kapitel 7.2.5.1). In Bereich 3 können Handlungen gestartet werden, die zur Evaluation der Sensoren dienen. Diese Handlungen sind im Kapitel 9 beschrieben. In Bereich 2 sind alle Handlungen zusammengefaßt, die dem Testen des gesamten Systems dienen. Die einzelnen Handlungen werden im folgenden beschrieben.

Stop

Der Roboter bleibt schnellstmöglich stehen.

Viereck-Fahrt

Bei dieser Handlung fährt der Roboter kontinuierlich ein Viereck ab. Abbildung 8.2 zeigt ein Beispiel für eine solche Fahrt. Hierbei wurde die Roboterposition zehnmal pro Sekunde eingezeichnet.

Position anfahren

Bei dieser Handlung kann über das Hostsystem ein Zielpunkt angegeben werden, der vom Roboter mit Hilfe der Wegplanung (s. Kapitel 11.1) angefahren wird. Der Zielpunkt wird durch den in Bereich 4 dargestellten transparenten Roboter festgelegt. Außerdem ist einstellbar, ob die Kollisionsvermeidung für diese Fahrt genutzt werden soll.

8.2 Simulator

Da ein Testen auf dem Roboter zeitaufwändig ist, wurde ein Simulator entwickelt, um die Entwicklungszeit zu verkürzen.

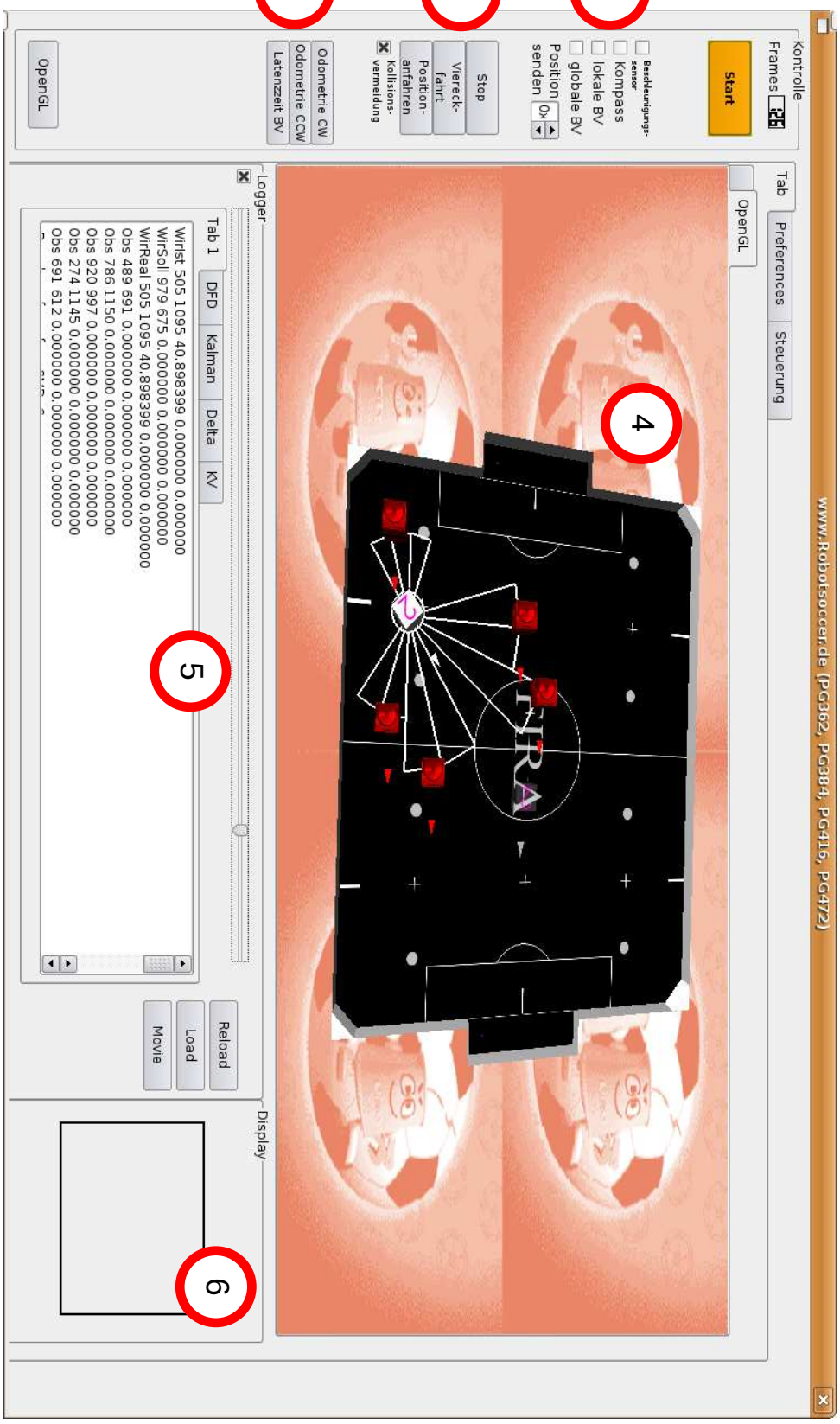

Abbildung 8.1: Benutzeroberfläche des Hostsystems

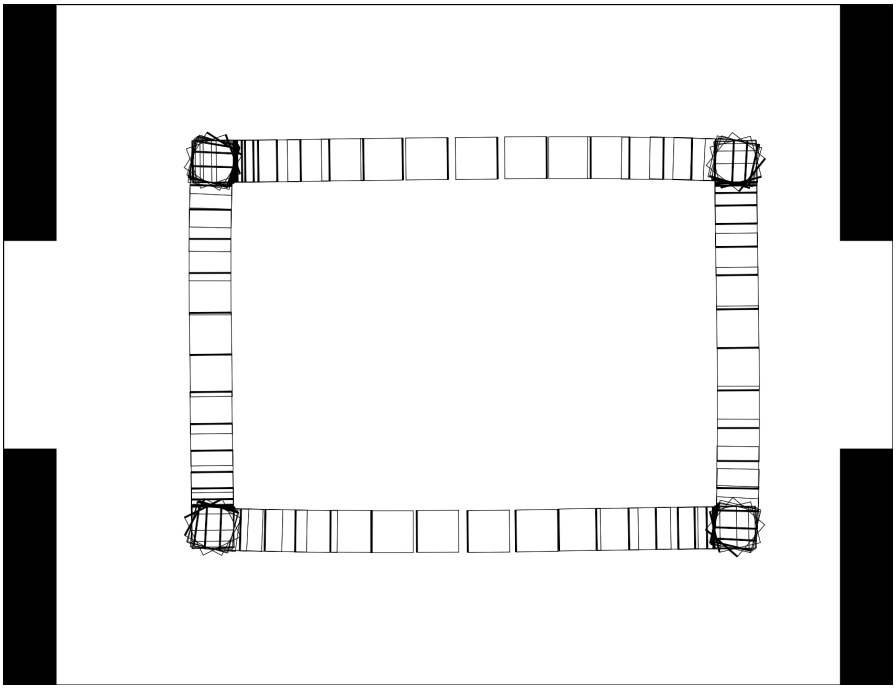

Abbildung 8.2: Beispiel für eine Viereck-Fahrt

Bei der Konzeptionierung des Simulator war die Hauptpriorität, dass ein Großteil der Software des Roboters ohne Änderungen im Simulator laufen sollte und dass die Befehle oberhalb der Regelung getestet werden können. Im Gegensatz zum Simulator des Roboterfußball-Projektes hatten andere Punkte wie Kollisionserkennung oder Reflektionen des Balles an der Bande oder an einem Roboter deutlich niedrigere Priorität.

Gegen Ende der Diplomarbeit wurde der Simulator nochmals deutlich erweitert. Es wurden Funktionen hinzugefügt, die Messungen der Sensoren inkl. Latenzzeit und Messabweichungen simulieren und das Robotermodell wurde so erweitert, dass es Odometriefehlern unterliegt.

Simulation

Um die DSP-Hardware zu simulieren, wird im Simulator das Modul Hardware, in dem alle direkten Hardwarezugriffe vereint sind, ausgetauscht. Quellcode zur Initialisierung des Roboters konnte ersatzlos entfallen. Statt über Funk zu kommunizieren, erfolgt die Kommunikation zwischen Simulator und der globalen Bildverarbeitung über eine Datenstruktur. Statt die Daten an die serielle Schnittstelle zu schicken, schreibt das Funkmodul die Daten in eine Datenstruktur aus der die Routinen der DSP-Sourcen diese auslesen.

Bei der Simulation der Motoren wurde ein einfaches Modell gewählt. Unter der Annahme, dass die benutzte Rampe verhindert, dass der Roboter in einen unkontrollierbaren Zustand gerät, wird die Bewegung mit Hilfe des dynamischen Modells (s. Kapitel 2.1.2) berechnet. Als Geschwindigkeit werden die im jeweils vorherigen Regelschritt gesetzten Geschwindigkeiten genutzt.

Für jeden Roboter existieren im Simulator zwei unterschiedliche Konfigurationen. Die erste Konfiguration beschreibt die wahre Position des Roboters. Die zweite Konfiguration beschreibt die Position, so wie sie im Roboter mit Hilfe der Sensorfusion berechnet wird.

Kollisionen

Um Kollisionen zwischen zwei Robotern zu erkennen, wird angenommen, dass die Roboter eine Kreisform haben. Diese Kreisform ist so gewählt, dass der Roboter komplett von ihr aufgenommen werden kann. Der Durchmesser entspricht der größten Ausdehnung des Roboters, also $\sqrt{2 * 7,5^2}$ cm. Um eine Kollision zu erkennen, muss überprüft werden, ob sich die beiden Kreise, die die Roboter repräsentieren, schneiden. Dies ist durch eine einfache Bestimmung des Abstandes möglich.

Um eine Kollision zwischen Roboter und Bande zu bestimmten, wird angenommen, dass der Roboter keine Ausdehnung hat. Stattdessen wird angenommen, dass sich die Banden für Roboter $0,5 * \sqrt{2 * 7,5^2}$ cm und für den Ball $0,5 * 4,26$ cm, also um die Hälfte des Balldurchmessers, weiter innerhalb des Spielfeld befinden. Eine Kollision mit einer Bande hat stattgefunden, wenn der Bewegungsvektor, der aus der vorherigen Position und der aktuellen Position konstruiert wird, die hereingezogene Bande schneidet. Diese Überprüfung ist durch eine einfache Schnittpunktberechnung von zwei Strecken möglich.

Sensoren

Bei der Simulator der Messwerte von Sensoren werden die Latenzzeit und die Messabweichung berücksichtigt. Um die Messung eines Sensors zu simulieren, wird im ersten Schritt aus der Datenstruktur die wahre Konfiguration des Roboter unter Berücksichtigung der Latenzzeit herausgesucht. Anschließend werden die Messwerte entsprechend der in Kapitel 9 berechneten Varianzen verfälscht.

8.3 Bildverarbeitung

Die Bildverarbeitung wurde zum Beginn der Diplomarbeit aus dem Roboterfußball-Projekt übernommen. Bislang für den Roboterfußball ausgelegt, war die Bildverarbeitung unter der Voraussetzung programmiert, dass es gleich viele eigene wie gegnerische Roboter zu erkennen gab. Da in dieser Arbeit die gegnerischen Roboter als Hindernisse genutzt werden sollten, musste an dieser Stelle die Bildverarbeitung geändert werden, um eine flexible Anzahl an Hindernisse zu erkennen. Diese eigentlich kleine Änderung, führte zu Änderungen an 17 Klassen.

Zum Beginn der Diplomarbeit stand für diese Arbeit keine der normalerweise eingesetzten Firewire-Kameras zur Verfügung. Deshalb wurde die Bildverarbeitung so erweitert, dass sie auch USB-Kameras nutzen kann. Hierzu wurden Routinen für den Zugriff auf die video4linux-Schnittstelle programmiert. Diese Schnittstelle ermöglicht unter Linux den Zugriff auf eine Vielzahl von Kameras, TV-Karten und ähnlichen Geräten.

Da an der Oberfläche für die Steuerung der Bildverarbeitung keine Änderungen vorgenommen wurden, sei für weitere Informationen auf [WJ04], [Pro03, S. 42ff], [Pro05, S. 9ff] und [Pro06, S. 56ff] verwiesen.

8.4 OpenGL-Ausgabe

OpenGL (Open Graphics Library) beschreibt eine plattform- und programmiersprachenunabhängige Schnittstelle zur Erstellung von 3D-Grafiken. Im Hostsystem wird mit Hilfe von OpenGL im Bereich 4 das Spielfeldes inkl. der Roboter und Hindernisse dargestellt. Im Rahmen dieser Arbeit wurden die Darstellungsmöglichkeiten der Ausgabe

erweitert. Es können 3 zusätzliche geometrische Formen eingezeichnet werden: Kreissektoren, um die VFH-Kollisionsvermeidung zu visualisieren, Ellipsen, um die Varianzen in der Position darzustellen, und Linien um z.B. die Viereckfahrt darzustellen. Bereich 6 wird zur Simulation des Displays (s. 7.2.2.3) genutzt. Da diese Anzeige jedoch die Simulationsdauer erheblich steigert, ist die Anzeige nur optional aktivierbar.

8.5 Protokollierung

Zur Fehlersuche ist es notwendig, die Simulation Schritt um Schritt zu analysieren. Aus diesem Grund wurde die Funktionalität der Protokollierungfunktion (im Projekt „Logger" genannt) erweitert. Es werden alle relevanten Daten der Simulation (Konfiguration des Roboters, Matrizen des Kalmanfilter, Parameter der Befehle, Odometrie, ...) als Textdatei gespeichert. Mit Hilfe der Benutzerelemente in Bereich 5 kann diese Datei später analysiert werden. Für jeden Schritt werden die Daten in verschiedene Kategorien unterteilt und getrennt dargestellt. In der OpenGL Ansicht (Bereich 4) wird die jeweils aktuelle Konfiguration des Roboters und der Hindernisse dargestellt. Außerdem werden sowohl die belegten Kreissektoren der Kollisionsvermeidung als auch die Varianz dargestellt. Die Varianz wird durch eine Ellipse um dem Roboter dargestellt, deren Größe abhängig von den Varianzwerten für die X- und Y-Position ist.

9 Evaluation der Sensoren

Die Evaluation stützt sich auf eine Teilmenge der in Kapitel 3.1 vorgestellten Kriterien. In Hinblick auf die Sensorfusion (s. Kapitel 10) liegt ein besonderes Augenmerk auf der Eignung der Sensoren zur Erstellung eines Weltmodells und somit auf den Kriterien Auflösung, Messabweichung und Latenzzeit.

Die Evaluation des Kompasssensors wurde nicht vollendet, da die vom Kompasssensor zurückgelieferten Messwerte trotz Kalibrierung keinen Bezug zur Ausrichtung hatten.

9.1 Radencoder

Für das Testen der Radencoder und Odometrie (vgl. Kapitel 3.2.1) hat sich die sogenannte UMBmark-Methode (University of Michigan Benchmark) von Borenstein und Feng (s. [BF94]) durchgesetzt. Die Basis dieses Tests bildet ein Viereck, dessen Kanten vom Roboter abgefahren werden. Zu Beginn steht der Roboter an einer Ecke des Vierecks. Die genaue Roboterposition wird bestimmt und an den Roboter übermittelt. Anschließend fährt der Roboter die erste Kante ab, dreht sich am Ende der Kante um 90° und fährt die nächste Kante ab. Diese Befehlsabfolge wird solange wiederholt, bis der Roboter alle vier Kanten abgefahren hat und wieder die erste Ecke erreicht hat. Hierbei ist es nicht nötig, dass der Roboter genau die Startposition erreicht. Da die Odometrie, nicht jedoch die Steuerung und Regelung, getestet werden soll, ist es ausreichend, wenn der Roboter in der Nähe der Startposition stoppt.

Anschließend wird die Endposition ermittelt, sowie die Position ausgelesen, die der Roboter mit Hilfe der Odometrie berechnet hat. Der Abstand zwischen diesen beiden Positionen ist der Odometriefehler. Um zu verhindern, dass sich zwei unterschiedliche Fehlerquellen gegenseitig aufheben, wird das Viereck sowohl im als auch gegen den Uhrzeigersinn abgefahren. Um nicht-systematische Odometrie Fehler zu minimieren, wird die ganze Strecke mit relativ niedriger Geschwindigkeit abgefahren.

Um die Odometrie des benutzten Roboters zu testen, wurden insgesamt 8 Fahrten durchgeführt, vier im und vier gegen den Uhrzeigersinn. Das Viereck hatte hierbei, bedingt durch die Spielfeldgröße, eine Kantenlänge von einem Meter.

Abbildung 9.1 zeigt für jede Fahrt die Differenz zwischen der errechneten und der tatsächlichen Position. In der Abbildung ist die tatsächliche Position jeweils durch den Nullpunkt repräsentiert und die Messpunkte geben jeweils den relativen Abstand zischen tatsächlicher und berechneter Position an. Auffällig ist die Clusterbildung. Während die Größe der Cluster ein Maß für die Nicht-Systematischen Fehler ist, ist der Abstand zwischen den Mittelpunkten der Cluster und dem Nullpunkt ein Maß für die systematischen Fehler.

Borenstein und Feng stellen in ihrer Arbeit ein Verfahren zur Reduzierung von systematischen Fehlern vor. Da in den Odometrie-Gleichungen nur die Werte für den Radabstand und die Anzahl der Radencoder-Impulse pro mm veränderbar sind, konzentrieren sich

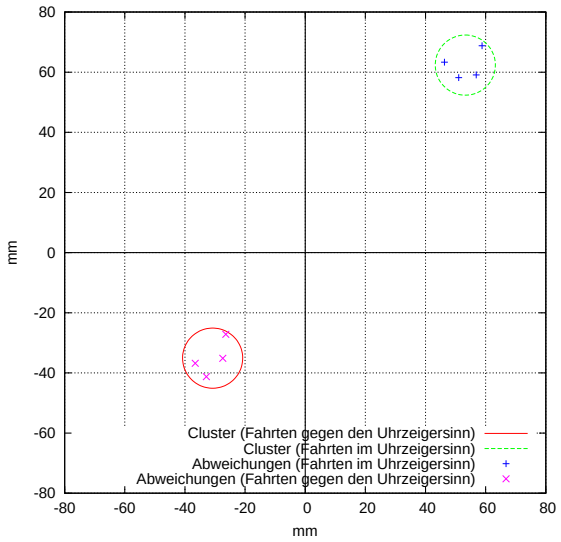

Abbildung 9.1: Abweichungen zwischen errechneter (Nullpunkt) und tatsächlicher Position

die Autoren auf die Korrektur dieser Werte. Die gesamte Korrektur stützt sich jedoch nur auf die vom Roboter berechnete Position und die tatsächlichen Eckpunkte des gefahrenden Vierecks. Da jedoch bei dem eingesetzten Roboter eine Protokollierung der Daten der Radencoder für jeden Regelungschritt möglich ist (vgl. Kapitel 7.2.5.3), hat sich der Autor dieser Arbeit gegen die von Borenstein und Feng vorgestellte Methode zur Korrektur entschieden.

Stattdessen wird von jeder Fahrt die tatsächliche Startkonfiguration, die tatsächliche Zielkonfiguration und die Anzahl der Radencoder-Impulse in jedem Reglerschritt protokolliert. Mit Hilfe dieser Informationen können die Odometrie-Berechnungen simuliert werden. Die drei veränderbaren Parameter (Radabstand, Impulse pro mm für jedes Rad) können so in einer Simulation optimiert werden.

Zur Optimierung kommt eine (1+1)-Evolutionsstrategie zum Einsatz. Die Fitnessfunktion summiert die Abstände zwischen errechneter und tatsächlicher Position für alle 8 Fahrten auf. Abbildung 9.2 zeigt den Verlauf der Optimierung. Deutlich wird, dass nach 32 Durchläufen der Odometriefehler bereits deutlich reduziert ist und nach 89 Durchläufen keine signifikante Verbesserung mehr zu erreichen ist. Die folgende Tabelle zeigt die Parameter vor und nach der Optimierung.

	vor der Optimierung	nach der Optimierung
Radabstand	67,000	65,640
Impulse pro mm, links	69,818	69,989
Impulse pro mm, rechts	69,818	70,045
maximaler Fehler (in mm)	90,47	23,68
maximaler Ausrichtungsfehler (in °)	8,02°	5,73°

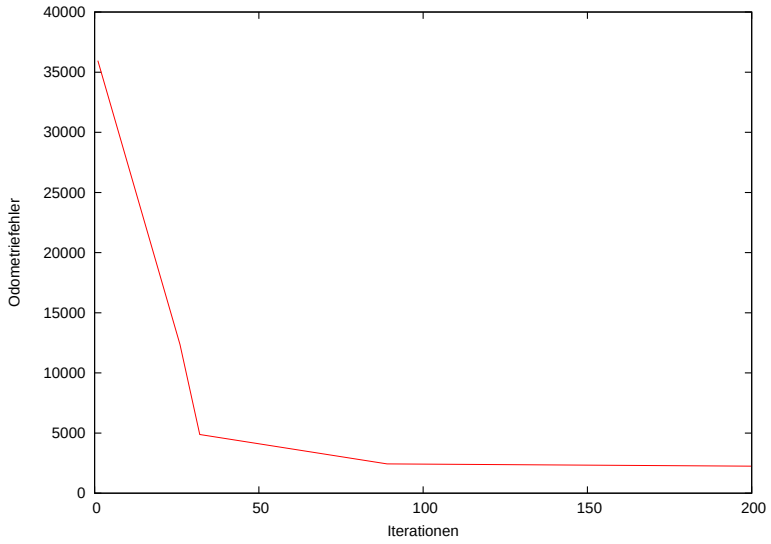

Abbildung 9.2: Verlauf der Optimierung der Odometrieparamter

Abbildung 9.3 zeigt die Odometriefehler der Testfahrten, wenn die optimierten Parameter zur Berechnung genutzt werden. Um den Odometriefehler weiter zu senken, wäre eine Untersuchung notwendig, welcher der in Kapitel 3.2.1 aufgeführten Punkte bei den systematischen Fehlern dominiert. Basierend auf dieser Untersuchung müssten dann Maßnahmen ergriffen werden, um diesen systematischen Fehler zu senken. Da die erreichte Genauigkeit der Odometrie für die Aufgabenstellung ausreichend ist, wird diese Untersuchung nicht durchgeführt.

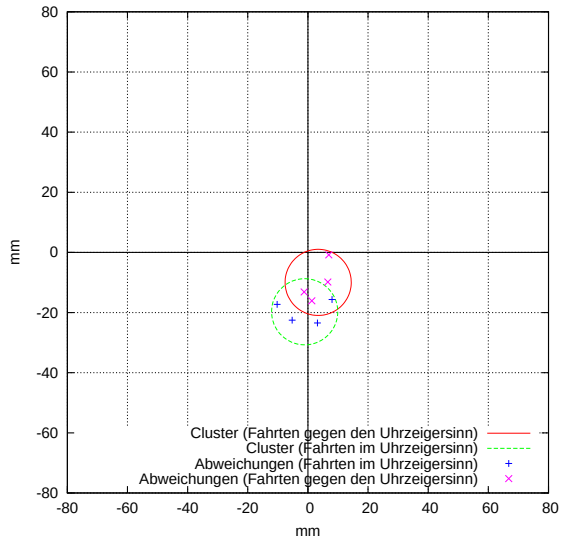

Abbildung 9.3: Abweichungen zwischen errechneter (Nullpunkt) und tatsächlicher Position nach der Optimierung

9.2 Beschleunigungssensor

Um die Eignung des Beschleunigungssensors (s. Kapitel 7.2.4.3) für die Erstellung des Weltmodells zu evaluieren, wurden verschiedene Tests durchgeführt. Abbildung 9.4 zeigt die Ergebnisse des ersten Tests: Der Roboter beschleunigt auf eine Geschwindigkeit von $10\frac{cm}{s}$ und hält diese Geschwindigkeit. Nach insgesamt 3 Sekunden bremst er auf eine Geschwindigkeit von 0 ab. Die Messungen entlang der Z-Achse unterliegen der Erdbeschleunigung, weshalb die Beschleunigung auf der Z-Achse im Ruhezustand bei 1 g bzw. $9,81\frac{m}{s^2}$ liegt. Die Ausschläge entlang der Z-Achse lassen sie durch das Hin- und Herkippen des Roboters während der Fahrt erklären. Nur jeweils einer der beiden unter dem Roboter angebrachten Schleifer hat Kontakt zum Boden. Bei Beschleunigungen kann der eine Schleifer den Bodenkontakt verlieren, während der andere Bodenkontakt erhält. Dieses führt zu einer leichten Kippbewegung, die auf der Z-Achse sichtbar wird.

Auf der Y-Achse werden die Beschleunigungen in Fahrtrichtung deutlich. Der Beschleunigungssensor detektiert zum Beginn der Fahrt eine Beschleunigung, die aufgrund der Anordnung des Beschleunigungssensor im Roboter im Diagramm mit negativen Wert eingezeichnet ist. Später gibt es einige kurze Bremsvorgänge, um die gewünschte Geschwindigkeit zu halten. Zum Ende der Fahrt wird der Roboter durch eine starke Bremsung zum Halten gebracht. Versuche, die Momentangeschwindigkeit des Roboters durch Integration der Beschleunigungskurve zu bestimmen, führten nicht zum Erfolg, da hierfür die Messwerte des Beschleunigungssensor zu ungenau sind.

Abbildung 9.5 zeigt eine Fahrt, bei der der Roboter durch eine Kollision mit einer Bande gestoppt wurde. Abbildung 9.6 zeigt den Zeitraum der Kollision in einer vergrößerten Ansicht. Deutlich fällt die Bremsung von $13,7\frac{m}{s^2}$ (Y-Achse; Zeitpunkt ungefähr 11450) auf. Da ein solcher Wert durch normales Beschleunigen bzw. Bremsen ohne Kollision nicht erreicht werden kann, kann der Beschleunigungssensor zur Kollisionserkennung eingesetzt werden.

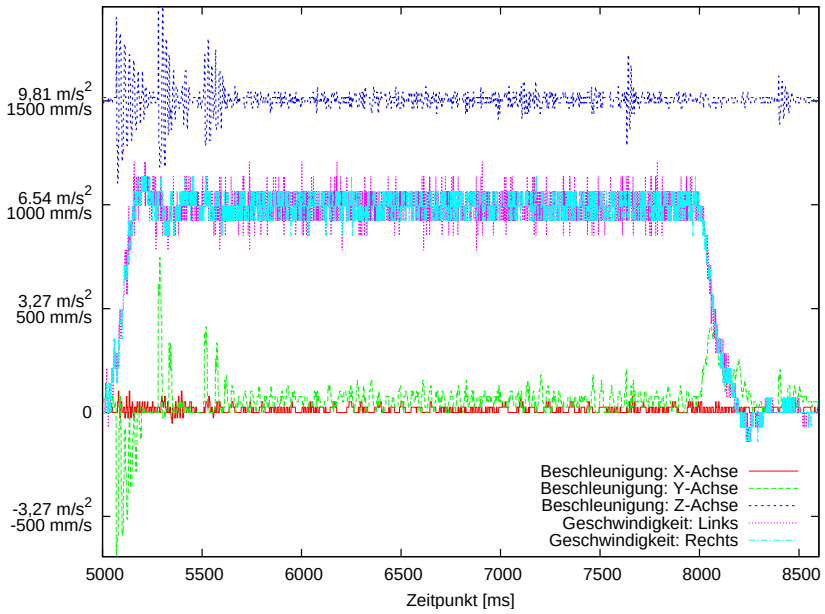

Abbildung 9.4: Beschleunigungskurven einer Fahrt

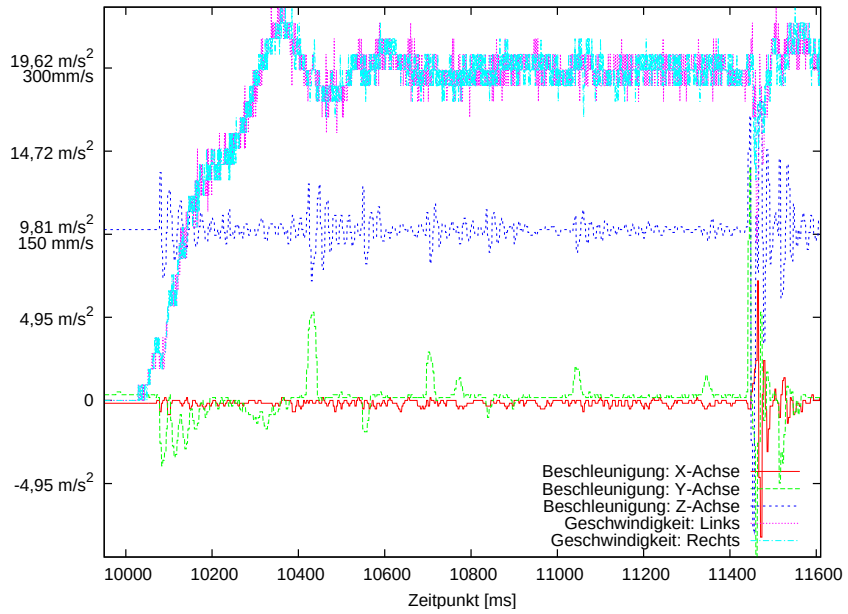

Abbildung 9.5: Beschleunigungskurven einer Fahrt mit einer Kollision

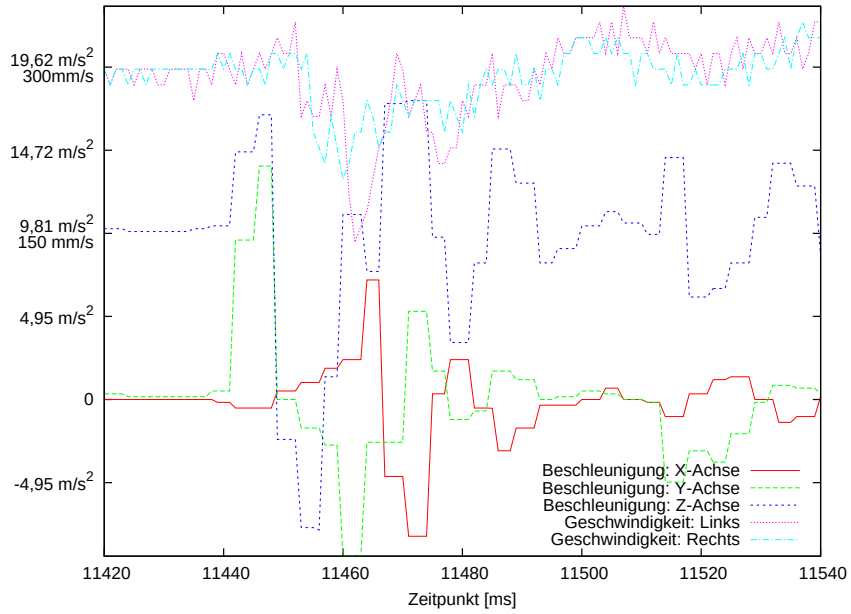

Abbildung 9.6: Beschleunigungskurven während einer Kollision. (Ausschnittsvergrößerung der Abbildung 9.5)

9.3 globale Bildverarbeitung

Im folgenden werden Messungen der Latenzzeit und der Messabweichung durchgeführt, da beide Werte für die Sensorfusion unabdingbar sind.

9.3.1 Latenzzeit

Um die Latenzzeit der globalen Bildverarbeitung zu messen, wurde folgender Versuchsaufbau gewählt: Der Roboter führt im Stand regelmäßige Richtungsänderungen aus und ermittelt dabei mit Hilfe der Odometrie seine Ausrichtung. Die Bildverarbeitung sendet via Funk regelmäßig die ermittelte Roboterausrichtung an den Roboter. Beide Ausrichtung werden protokolliert. Abbildung 9.7 zeigt den Kurvenverlauf der bei-

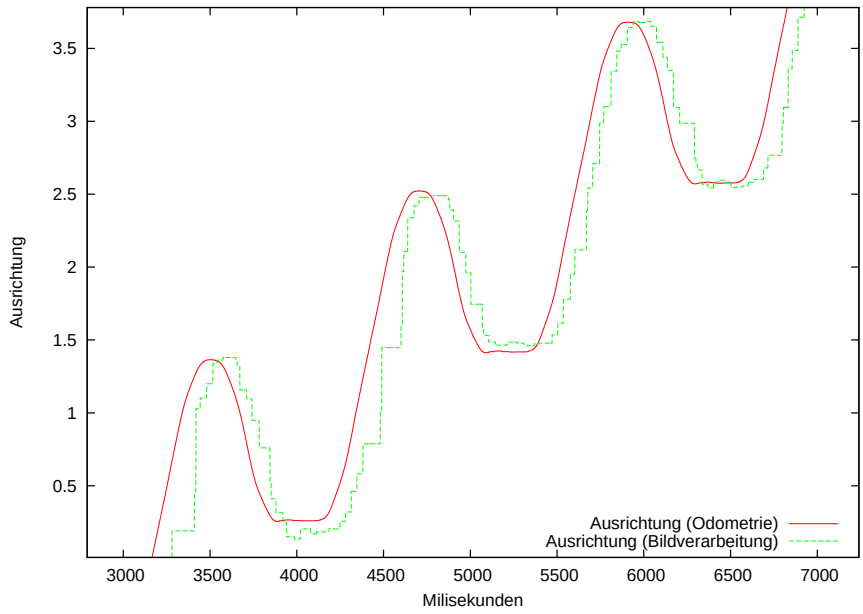

Abbildung 9.7: Durch die Odometrie und durch die Bildverarbeitung ermittelte Ausrichtung

den ermittelten Ausrichtung. Die Verschiebung der beiden Kurven zueinander liegt in der Latenzzeit der Bildverarbeitung und des Funks begründet. Bei der Benutzung der Bim-Module ergibt sich somit eine Latenzzeit von 82 ± 3 Millisekunden. Da die Bearbeitungszeiten der Kamerabilder auf dem Hostsystem nicht konstant sind, kommt es zu Schwankung in der Latenzzeit.

Bei der Benutzung der Yab-Module ergibt sich eine Latenzzeit von ca. 140 Milisekunden. Bis zum Abschluss dieser Arbeit konnte nicht endgültig geklärt werden, wie diese erhebliche Differenz zu begründen ist und deshalb wurden ausschließlich die Bim-Module genutzt.

9.3.2 Messabweichungen

Um die zufälligen und die systematischen Messabweichungen zu messen, wurde der Roboter an verschiedenen Stellen des Spielfelds positioniert und die Messergebnisse (X-,

Y-Position und Ausrichtung) von mehreren Minuten gespeichert. Anschließend wurde die Streuung der Messwerte bestimmt. Abbildung 9.8 zeigt die Positionen der einzelnen

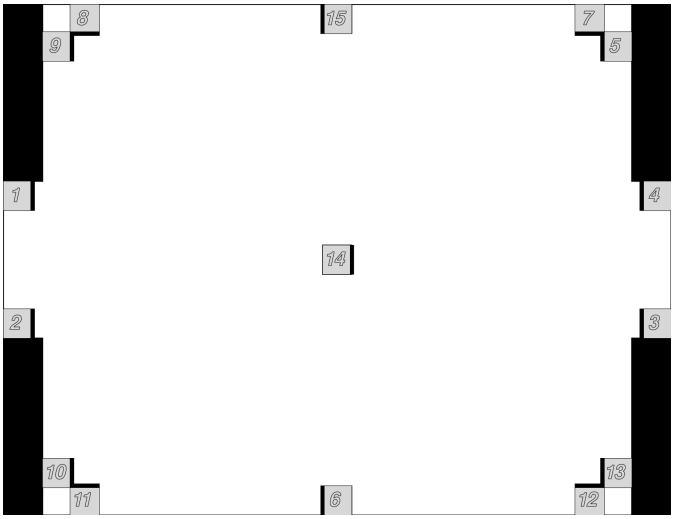

Abbildung 9.8: Die Positionen der einzelnen Messungen.

Messpunkte In Abbildung 9.9 ist für jede Messung die Differenz zwischen tatsächlicher

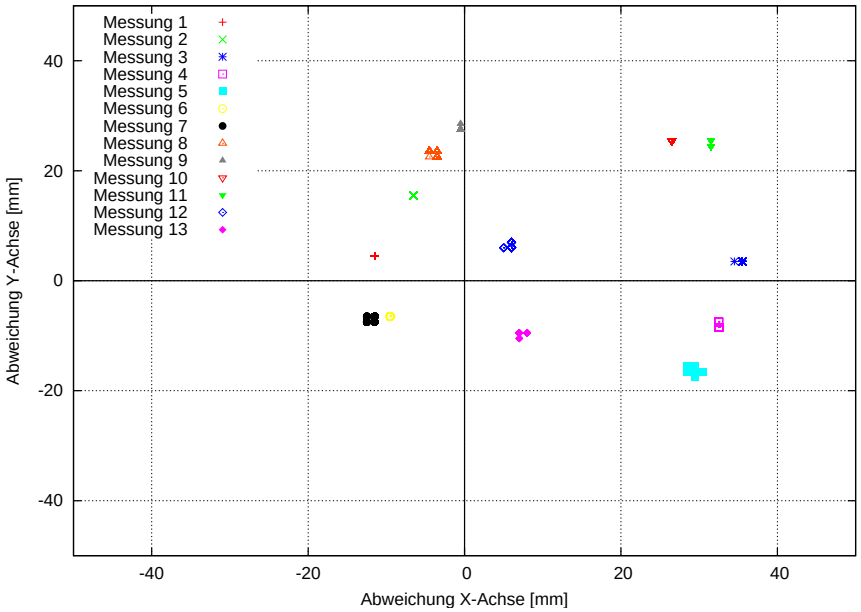

Abbildung 9.9: Messabweichungen der globalen Bildverarbeitung

und von der Bildverarbeitung ermittelte Position eingezeichnet. Die ermittelte Position wird hierzu jeweils so verschoben, dass sich die tatsächliche Position auf den Nullpunkt befindet. An jeder Position wurden mehrere Messungen durchgeführt, die in der Grafik mit den gleichen Symbolen eingezeichnet sind. Die zufälligen Messabweichungen sind an den Clustern von gleichen Symbolen erkennbar, während der Abstand vom Nullpunkt ein Maß für die systematische Messabweichung ist.

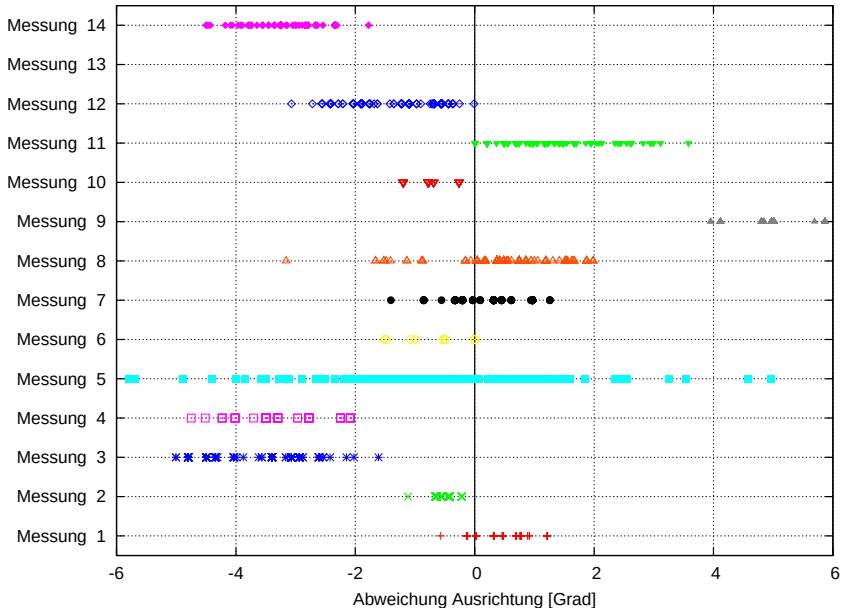

Abbildung 9.10: Abweichungen in der Ausrichtungerkennung der Bildverarbeitung

Abbildung 9.10 zeigt die Messabweichungen in der Ausrichtung. Während die verschiedenen Messungen in der Grafik vertikal abgetragen sind, sind die Messabweichungen horizontal abgetragen.

Auf Basis dieser Messungen wurde die Standardabweichung bestimmt, die für die Sensorfusion benötigt wird:

Parameter	Standardabweichung
x	21,21 mm
y	18,35 mm
θ	2,27°

9.4 lokales Bildverarbeitungssystem

Das lokale Bildverarbeitungssystem konnte nur teilweise getestet werden, da der für die Tests präparierte Roboter vier Wochen vor dem Ende dieser Diplomarbeit irreparable beschädigt wurde. Auf Basis von Daten, die Simon Schulz dem Autor dieser Arbeit zur Verfügung gestellt hat, konnten jedoch die Messabweichungen bestimmt werden. Abbildung 9.11 zeigt die Messergebnisse von drei Messungen an drei unterschiedlichen Positionen. Die Ellipsen repräsentieren die Standardabweichungen für die jeweilige Messung.

Auf Basis dieser drei Messungen wurde die Standardabweichung für das lokale Bildverarbeitungssystem bestimmt:

Parameter	Standardabweichung
x	45,23 mm
y	109,95 mm

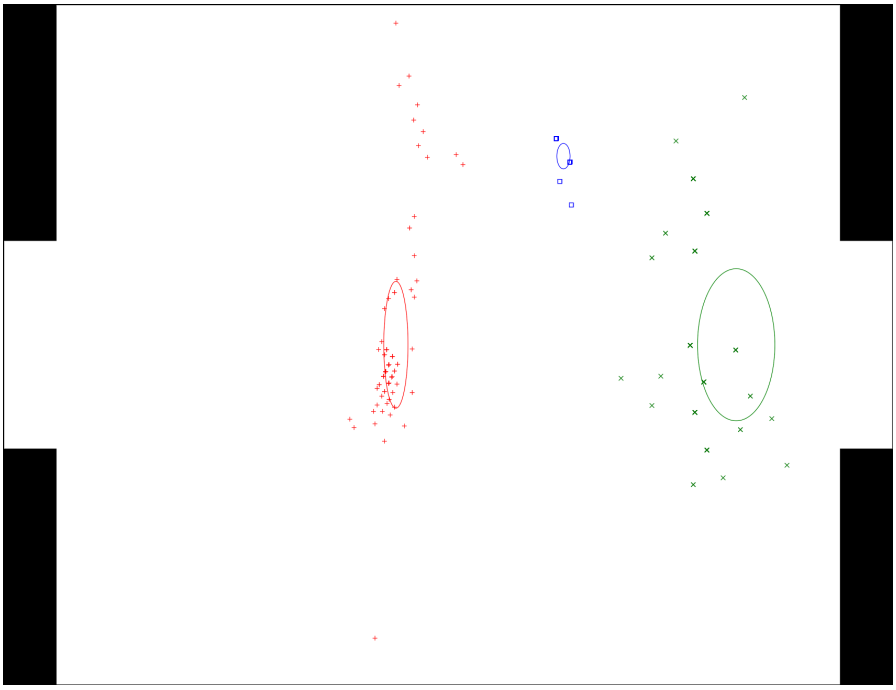

Abbildung 9.11: Messabweichungen der lokalen Bildverarbeitungen

Diese Werte basieren auf Messungen zum Anfang der Testphase des Bildverarbeitungssystem. Innerhalb dieser Testphase konnte Simon Schulz die Genauigkeit des Bildverarbeitungssystem noch steigern. Für weitere Informationen sei auf [Sch06] verwiesen.

10 Sensorfusion

Im folgenden werden die Kriterien für die Auswahl des Algorithmus zur Sensorfusion beschrieben. Anschließend wird die konkrete Implementierung inkl. Bestimmung der Modellgüte, Kompensation der Latenzzeiten und die notwendigen Optimierungen erläutert. Das Kapitel schließt mit verschiedenen Tests der Sensorfusion.

10.1 Auswahl

Aus den in Kapitel 4 vorgestellten Algorithmen zur Sensorfusion wurde der Kalmanfilter ausgewählt.

Beim Multihypothesen Tracking (s. Kapitel 4.1.2) wird für jede Vermutung ein Kalmanfilter eingesetzt, weshalb die Rechenzeit um ein Vielfaches über der eines einzelnen Kalmanfilters liegt. Der Speicherplatzbedarf von Gitterbasierte Verfahren (s. Kapitel 4.1.3) ist bei einer Genauigkeit von 0,5 cm größer, als der vorhandene Speicher. Bei den Partikelfiltern (s. Kapitel 4.1.4) muss für jeden Partikel die Wahrscheinlichkeiten angepasst werden. Der Rechenzeitbedarf ist somit wesentlich höher als beim Kalmanfilter.

Der Kalmanfilter benötigt von allen vorgestellten Verfahren am wenigsten Speicherplatz und Rechenzeit. Seine Beschränkung auf eine Vermutung stellt bei den genutzten Sensoren keine Einschränkung da.

10.2 Zustandsänderungen

Zentrales Element des Kalmanfilter ist die Beschreibung der Zustandesänderung. Da es sich um ein nicht-lineares dynamisches System handelt, kommt der erweiterte Kalmanfilter zum Einsatz.

Die Systemgleichung hat die Form:

$$x_k = f\left(x_{k-1}, u_k, w_{k-1}\right) \tag{10.1}$$

Der Zustandsvektor $x_k = \begin{pmatrix} x & y & \theta & v_l & v_r \end{pmatrix}^{\mathrm{T}}$ enthält die aktuelle Position (x, y) bezüglich des Spielfelds, die Ausrichtung des Roboters (θ), sowie die linke und rechte Radgeschwindigkeit (v_l, v_r). $u_k = \begin{pmatrix} \Delta v_l & \Delta & v_r \end{pmatrix}$ ist der Regelungseingriff. Hierbei geben Δv_l und Δv_r die Geschwindigkeitsänderungen der Räder zwischen Zeitpunkt k und k-1 an.

Als Systemgleichung ergibt sich:

$$x_k \quad = \quad f\left(x, y, \theta, v_l, v_r, \Delta v_l, \Delta v_r\right) \tag{10.2}$$

$$= \begin{pmatrix} f_1(x, y, \theta, v_l, v_r, \Delta v_l, \Delta v_r) \\ f_2(x, y, \theta, v_l, v_r, \Delta v_l, \Delta v_r) \\ f_3(x, y, \theta, v_l, v_r, \Delta v_l, \Delta v_r) \\ f_4(x, y, \theta, v_l, v_r, \Delta v_l, \Delta v_r) \\ f_5(x, y, \theta, v_l, v_r, \Delta v_l, \Delta v_r) \end{pmatrix} \tag{10.3}$$

$$= \begin{pmatrix} x - \frac{v_l+v_r}{2} \sin\left(\theta - \frac{\pi}{2}\right) \\ y - \frac{v_l+v_r}{2} \cos\left(\theta - \frac{\pi}{2}\right) \\ \theta + \frac{v_r-v_l}{w} \\ v_l + \Delta v_l \\ v_r + \Delta v_r \end{pmatrix} \tag{10.4}$$

Die Gleichungen für f_1, f_2 und f_3 ergeben sich aus den in Kapitel 3.2.1 aufgestellten Gleichungen unter Berücksichtigung des benutzten Koordinatensystems.

Bei der Implementierung offenbarte dieser Ansatz jedoch mehrere Probleme. Zum einen ist der Regelungseingriff sehr schwer zu bestimmen. Am Ende eines Durchlaufs der Regelungsschleife werden die Motoren mit einem neuen PWM-Wert angesteuert. Um den Regelungseingriff zu bestimmen, muss aus der PWM-Änderung auf die Geschwindigkeitsänderung geschlossen werden. Für diese Abhängigkeit ist jedoch kein Modell vorhanden und die Erstellung eines solchen Modells würde den Rahmen dieser Diplomarbeit weit übersteigen.

Außerdem ist der Prozessor trotz der in Kapitel 10.7 beschriebenen Optimierungen nicht schnell genug, um die für eine Kalman Berechnung notwendigen Matrix Operationen in deutlich weniger als einer Millisekunde auszuführen.

Um das erste Problem zu lösen, gibt es in der Literatur den Vorschlag, die aus den Odometrie Daten berechneten Werten als Regelungseingriff und nicht als eine Messung zu benutzen. Da die Laufzeit einer $n \times n$ Multiplikation mit $O\left(n^3\right)$ abschätzbar ist, bietet es sich an, den Zustandsvektor zu verkleinern, um die Laufzeit der Kalman Berechnung zu reduzieren. Beide Punkte zusammengenommen führten zum folgenden Zustandsvektor und zur folgenden Systemgleichung:

$$x_k = \begin{pmatrix} x & y & \theta \end{pmatrix} \tag{10.5}$$

$$x_k = f(x, y, \theta, \Delta x, \Delta y, \Delta\theta) \tag{10.6}$$

$$= \begin{pmatrix} f_1(x, y, \theta, \Delta x, \Delta y, \Delta\theta) \\ f_2(x, y, \theta, \Delta x, \Delta y, \Delta\theta) \\ f_3(x, y, \theta, \Delta x, \Delta y, \Delta\theta) \end{pmatrix} \tag{10.7}$$

$$= \begin{pmatrix} x + \Delta x * \cos\left(-\left(\theta + \frac{\pi}{2}\right)\right) - \Delta y * \sin\left(-\left(\theta + \frac{\pi}{2}\right)\right) \\ y - \Delta y * \cos\left(-\left(\theta + \frac{\pi}{2}\right)\right) - \Delta x * \sin\left(-\left(\theta + \frac{\pi}{2}\right)\right) \\ \theta + \Delta\theta \end{pmatrix} \tag{10.8}$$

Die Werte Δx, Δy und $\Delta\theta$ werden durch die Odometrie berechnet und beschreiben die relative Bewegung entlang der X- bzw. Y-Achse und die relative Ausrichtungsänderung.

10.3 Vorhersageschritt

Mit Gleichung 10.8 kann $\hat{x}_k^-$ aus dem Vorhersageschritt des erweiterten Kalmanfilter berechnet werden. Die Varianz wird nach Gleichung 4.8 durch $\hat{P}_k^- = A_k P_{k-1} A_k^T + Q_{k-1}$ berechnet.

$$A_k = \left.\frac{\partial f}{\partial x}\right|_{x=\hat{x}_k^-} \tag{10.9}$$

$$= \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial \theta} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial \theta} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial \theta} \end{pmatrix} \tag{10.10}$$

$$= \begin{pmatrix} 1 & 0 & \Delta y * \cos\left(\theta + \frac{\pi}{2}\right) - \Delta x * \sin\left(\theta + \frac{\pi}{2}\right) \\ 0 & 1 & \Delta x * \cos\left(\theta + \frac{\pi}{2}\right) + \Delta y * \sin\left(\theta + \frac{\pi}{2}\right) \\ 0 & 0 & 1 \end{pmatrix} \tag{10.11}$$

Zur Bestimmung von Q_{k-1} sei auf das Kapitel 10.5 verwiesen.

10.4 Korrekturschritt

Im Korrekturschritt ist der lineare Kalmanfilter ausreichend, da bei den eingesetzten Sensoren der Zusammenhang zwischen der jeweiligen z-Matrix und $\hat{x}_k$ linear ist. Für jeden Sensor wird der Korrektur-Schritt mit den dazugehörigen z-, H- und R-Matrizen wiederholt.

Tabelle 10.1 führt die für den jeweiligen Korrektur-Schritt eingesetzen Matrizen auf. Die Werte für die R-Matrizen sind dem Kapitel 9 zu entnehmen.

Sensor	z-Matrix	H-Matrix
Kompasssensor	(θ)	$\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$
lokale Bildverarbeitung	$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
globale Bildverarbeitung	$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Tabelle 10.1: Übersicht über die im Korrektur-Schritt eingesetzen Matrizen

Bevor der Korrektur-Schritt ausgeführt werden kann, muss besonders Augenmerk auf Winkel geworfen werden. Sei der Zustandswinkel 1° und der Messwinkel 359°. Passende Varianzen vorausgesetzt, wird als neuer Zustandswinkel 0° erwartet. Ohne Korrektur der Winkel käme aber als neuer Zustandswinkel 179° heraus. Ist der Abstand zwischen Messwinkel und Zustandswinkel größer als 180°, wird in Abhängigkeit vom Zustandswinkel entweder 360° zum Messwinkel addiert oder vom Messwinkel 360° subtrahieren. Im konkreten Fall wird statt 359° -1° als Messwinkel angenommen. Am Ende des Korrekturschrittes muss ggf. der Zustandswinkel normalisiert werden, damit sich der Winkel im Bereich $[0°; 360°[$ befindet.

Beim Hinzufügen von neuen Sensoren muss nur der Korrekturschritt erweitert werden. Im wesentlichen muss hierfür nur die H- und R-Matrix für den jeweiligen Sensor bestimmt werden.

10.5 Maß für die Modellgüte

Die Matrix Q_{k-1} modelliert die Ungewissheit bezüglich der Modellgüte. Im folgenden wird ein einfaches Modell für die Berechnung diese Matrix präsentiert.

Da die Zustandsänderung durch die Odometrie berechnet wird, wird die Modellgüte durch den Odometriefehler dominiert. Zur Vereinfachung des Modells wird angenommen, dass die Kovarianzen für die X-, Y-Position und Ausrichtung allein von der zurückgelegten Strecke der beiden Räder (Δs) abhängig sind. Unter der Annahme, dass die Funktionen $f_x(\Delta s)$, $f_y(\Delta s)$ und $f_\theta(\Delta s)$ die Kovarianzen für die X- und Y-Position sowie für die Ausrichtung berechnen, gilt:

$$Q_{k-1} = \begin{pmatrix} f_x(\Delta s) & 0 & 0 \\ 0 & f_y(\Delta s) & 0 \\ 0 & 0 & f_\theta(\Delta s) \end{pmatrix} \tag{10.12}$$

Sei v_x, v_y die Varianz in X- und Y-Richtung bezgl. dem Koordinatensystem des Roboters und v_θ die Varianz der Ausrichtung. Unter den oben gemachten Annahmen, gilt:

$$\begin{aligned} v_x &= c_x * \Delta s \\ v_y &= c_y * \Delta s \\ v_\theta &= c_\theta * \Delta s \end{aligned}$$

Die Funktionen f_x und f_y müssen die Varianz aus dem Koordinatensystem des Roboters in das globale transformieren:

$$f_x(\Delta s) = \left| v_x * \sin\left(-\theta - \frac{\pi}{2}\right)\right| + |v_y * \sin(-\theta - \pi)| \tag{10.13}$$

$$f_y(\Delta s) = \left| v_x * \cos\left(-\theta - \frac{\pi}{2}\right)\right| + |v_y * \cos(-\theta - \pi)| \tag{10.14}$$

Für $f_\theta(\Delta s)$gilt:

$$f_\theta(\Delta s) = v_\theta = \Delta s * c_\theta \tag{10.15}$$

Zur Bestimmung der Konstanten c_x,c_y und c_θ wurde ein (1+1)-EA eingesetzt. Hierbei wurden die Odometriefahrten im Simulator solange mit verschiedenen Werten für c_x, c_y und c_θ wiederholt, bis die Ergebnisse der simulierten Kovarianzmatrix möglichst gut zu den Beobachtungen der Odometriefahrten passten. Für die drei Werte wurde ermittelt:

$$c_x = 7,0 * 10^{-6} \tag{10.16}$$

$$c_y = 1,7 * 10^{-5} \tag{10.17}$$

$$c_\theta = 1,42 * 10^{-4} \tag{10.18}$$

Desweiteren müssen in diesem Modell die vom Beschleunigungssensor detektierten Kollisionen berücksichtigt werden. Bei einer Kollision sind keine gesicherten Aussagen über

den Modellfehler möglich. Aus diesem Grund wird der Modellfehler möglichst groß angenommen, was da zuführt, dass bei den folgenden Berechnungen dem aktuellen Zustand nur minimale Genauigkeit zugesprochen wird und das Weltmodell sich auf Basis neuer Sensorinformationen neu aufbaut.

Insgesamt ergibt sich für Q_{k-1}:

$$Q_{k-1} = \begin{cases} \begin{pmatrix} v_{max} & 0 & 0 \\ 0 & v_{max} & 0 \\ 0 & 0 & v_{max} \end{pmatrix} \text{ falls Kollision} \\ \\ \begin{pmatrix} f_x(\Delta s) & 0 & 0 \\ 0 & f_y(\Delta s) & 0 \\ 0 & 0 & f_\theta(\Delta s) \end{pmatrix} \text{ sonst} \end{cases} \tag{10.19}$$

Der Wert für v_{max} ist abhängig von dem genutzten Zahlenformat und der Spielfeldgröße zu wählen.

Da die Werte für den Modellfehler pro Regelungsschritt kleiner als die Genauigkeit des genutzten Zahlenformats sein können, müssen die Werte für die zurückgelegte Strecke solange aufsummiert werden, bis der Modellfehler Größen erreicht, die mit dem genutzten Zahlenformat darstellbar sind.

Die folgende Tabelle zeigt die Standardabweichungen, die aus den Messungen ermittelt wurden (s. Kapitel 9.1), sowie die mit den obengenannten Werte simulierten.

Parameter	Standardabweichung Messung	Standardabweichung Simulation
x	21,21 mm	22,88 mm
y	18,35 mm	44.33 mm
θ	2,27°	1,84°

In [Kel00] wird ein komplexeres Modell vorgestellt, welches die Güte des Modells erhöhen könnte. Da die Güte des Modells für die Aufgabenstellung ausreichend ist, wird auf eine Implementierung verzichtet.

10.6 Kompensation der Latenzzeiten

Zur Kompensation der Latenzzeiten kommt das in Kapitel 4.2.2 beschriebe Verfahren „Extrapolation der Beobachtungen" zum Einsatz. Hierfür werden jeweils die letzten Konfigurationen des Roboters in einer Datenstruktur gespeichert. Die Anzahl der zu speichernden Konfigurationen hängt von der höchsten Latenzzeit der genutzten Sensoren ab.

Um die Latenzzeit eines Sensors zu kompensieren, wird die Änderung der Konfiguration innerhalb der Latenzzeit mit Hilfe der gespeicherten Konfigurationen berechnet und die Sensordaten entsprechend geändert. Anschließend werden die geänderten Daten der Sensorfusion zugeführt.

Schränkungen in der Latenzzeit werden durch eine Vergrößerung der Varianzen berücksichtigt. Hierbei wird die Bewegung innerhalb der Latenzzeitschrankung bestimmt und die Varianz entsprechend angepaßt.

10.7 Optimierung

Da der Kalmanfilter einen erheblichen Teil der verfügbaren Rechnenzeit benötigt, wurde Wert auf eine gute Optimierung gelegt.

Als Konsequenz aus den Benchmarks (s. Anhang A) wird die IQ-Math Bibliothek (s. Kapitel 6.1.2) genutzt. Zum Einsatz kommt das Format IQ24. Der Wertebereich dieses Formates ist $\left[-128 : 128 - 6 * 10^{-8}\right]$ bei einer Genauigkeit von $6 * 10^{-8}$. Damit der gewählte Wertebereich ausreichend ist, werden die Positionsdaten der Konfiguration in Metern und der Winkel in Gradmaß gespeichert.

Die nächsten Optimierungen setzen bei den Matrix-Berechnungen an. Da die vom Compiler generierten FOR-Schleifen wenig effizient sind und der Compiler kein Loop-Unrolling unterstützt, wurde ein Programm geschrieben, das optimierten Quellcode für jede benötigte Matrix-Multiplikation erzeugt. Die Multiplikation wird ohne Nutzung einer Schleife und direkt in den IQ-Math Zahlenformat durchgeführt. Die nachfolgenden Codezeilen zeigen beispielhaft eine generierte Funktion, die eine $(1 \times 3) * (3 \times 3)$ Multiplikation durchführt.

```
01| void matrix_mul1x3x3x3(matrixiq* a, matrixiq* b, matrixiq* nach)
02| {
03|   nach->zeilen=1;
04|   nach->spalten=3;
05|   nach->inhalt[0*3+0]=(_IQ24mpy(a->inhalt[0*3+0],b->inhalt[0*3+0]) +
06|                        _IQ24mpy(a->inhalt[0*3+1],b->inhalt[1*3+0]) +
07|                        _IQ24mpy(a->inhalt[0*3+2],b->inhalt[2*3+0]));
08|   nach->inhalt[0*3+1]=(_IQ24mpy(a->inhalt[0*3+0],b->inhalt[0*3+1]) +
09|                        _IQ24mpy(a->inhalt[0*3+1],b->inhalt[1*3+1]) +
10|                        _IQ24mpy(a->inhalt[0*3+2],b->inhalt[2*3+1]));
11|   nach->inhalt[0*3+2]=(_IQ24mpy(a->inhalt[0*3+0],b->inhalt[0*3+2]) +
12|                        _IQ24mpy(a->inhalt[0*3+1],b->inhalt[1*3+2]) +
13|                        _IQ24mpy(a->inhalt[0*3+2],b->inhalt[2*3+2]));
14| }
```

Um die Berechnung von Multiplikationen mit transponierten Matrizen zu beschleunigen, wurde auch für diese Art von Multiplikationen Funktionen nach dem obrigen Muster generiert, ohne dass in einem Zwischenschritt erst eine transponierte Matrix berechnet werden muss.

10.8 Evaluation

Um die korrekte Funktionsweise des Kalmanfilters zu testen, wurden verschiedene Tests durchgeführt. Exemplarisch werden hier die Ergebnisse einiger Tests präsentiert.

Abbildung 10.1 zeigt die Standardabweichungen für eine Viereck-Fahrt bei ausschließlicher Nutzung der Radencoder. Erwartungsgemäß steigen die Werte für die Standardabweichungen am Ende der Viereck-Fahrt in Bereiche der Messungen aus Kapitel 9.1.

Um die Auswirkungen eines Kompasssensors zu testen, wurde ein fiktiver Wert von $1°$ für die Standardabweichung angenommen. Abbildung 10.2 zeigt die Ergebnisse einer Viereck-Fahrt. Deutlich sichtbar ist, dass der Einsatz eines Kompasssensors den Anstieg der Standardabweichung abschwächt.

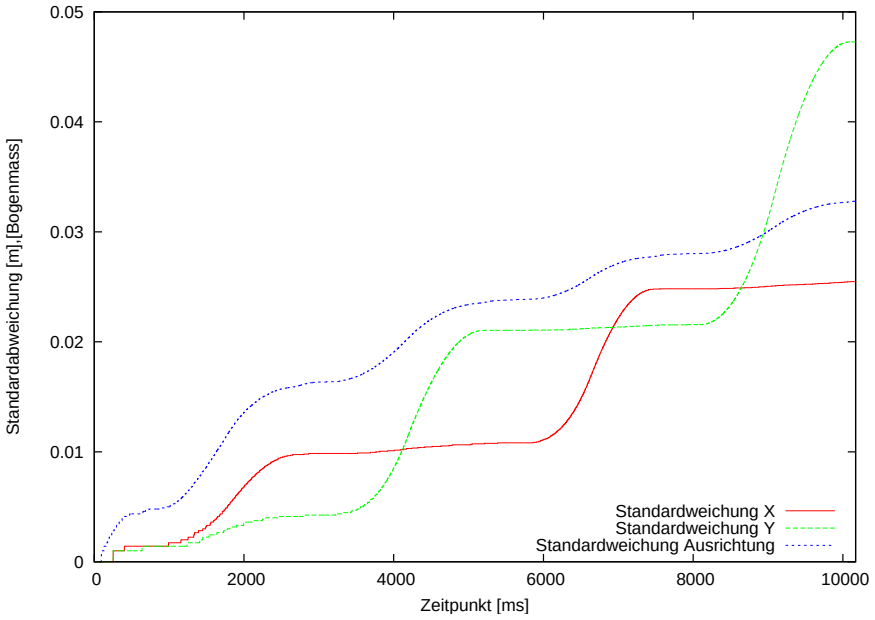

Abbildung 10.1: Entwicklung der Standardabweichungen bei einer Viereck-Fahrt

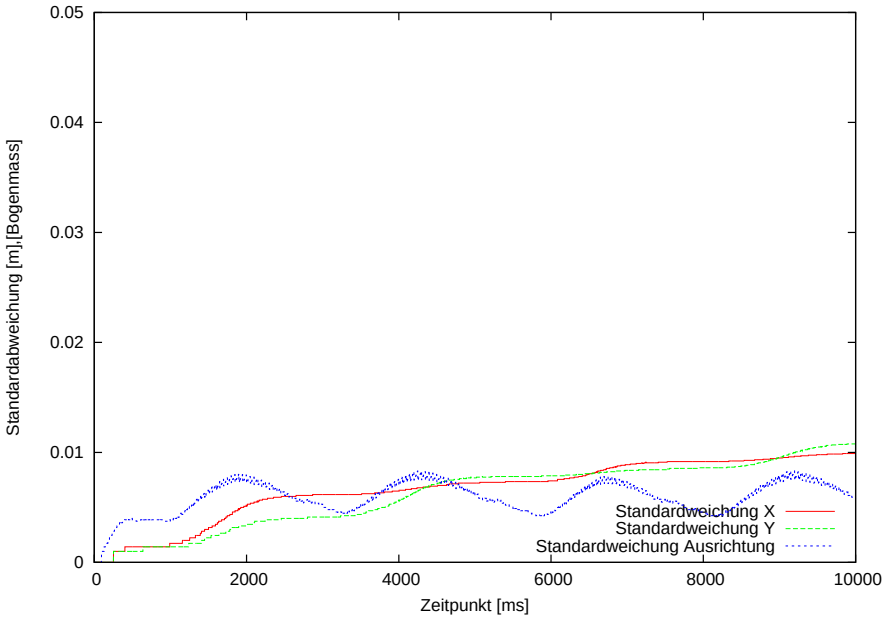

Abbildung 10.2: Standardabweichungen bei einer Viereck-Fahrt bei Benutzung eines Kompasssensors

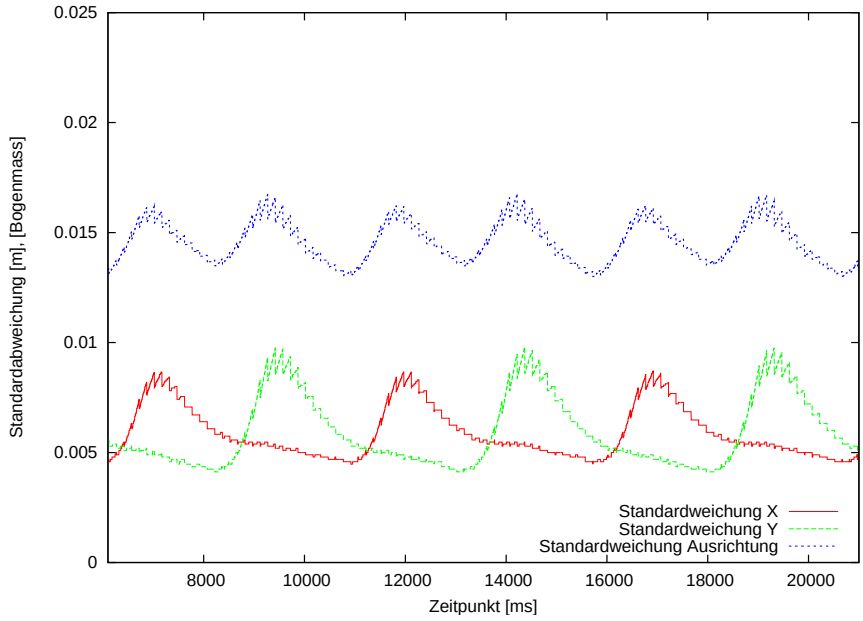

Abbildung 10.3: Standardabweichungen bei einer Viereck-Fahrt bei Benutzung des globalen Bildverarbeitungssystem

Abbildung 10.3 zeigt die Standardabweichungen bei Nutzung des globalen Bildverarbeitungssystem für eine Viereck-Fahrt. Deutlich erkennbar ist, dass die Standardabweichung in Abhängig von der Fahrtrichtung deutlich schwankt. Je nachdem, ob der Roboter entlang der X-Achse oder entlang der Y-Achse fährt, wächst mal die Standardabweichung für die X-Position schneller und mal die für die Y-Position. Außerdem ist gut erkennbar, wie die Messungen der Position durch das globale Bildverarbeitungsystem die Standardabweichung in regelmäßigen Abständen senkt.

Abbildung 10.4 zeigt die Kombination von dem globalen Bildverarbeitungssystem und dem Kompasssensor.

Die Tests zeigen das erwartete Verhalten einer Sensorfusion. Je mehr Sensorinformationen vorhanden sind, desto genauer wird das Weltmodell. Es zeigte sich auch, dass Sensoren, wie der Kompasssensor, die nur eine Koordinate des Weltmodells bestimmen können, die Genauigkeit des Weltmodell erhöhen können.

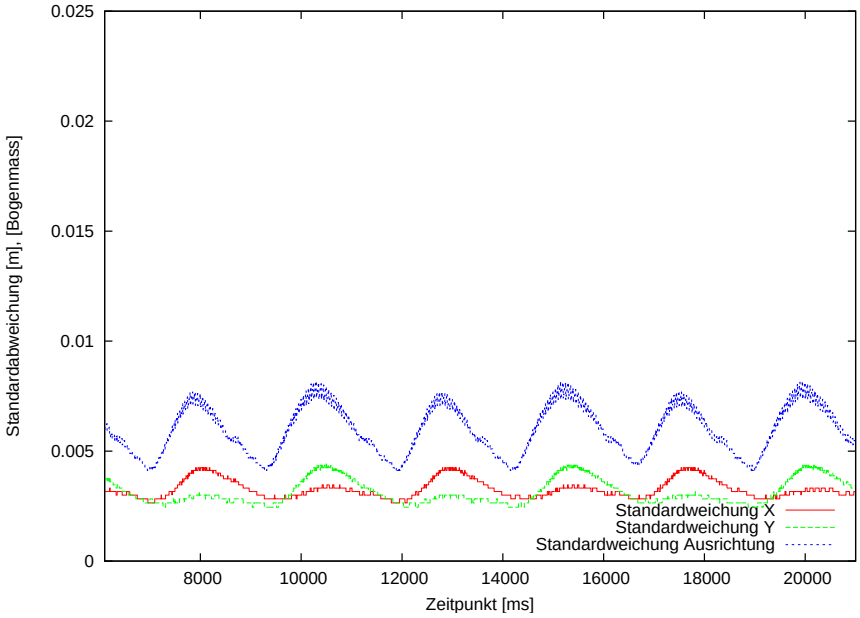

Abbildung 10.4: Standardabweichungen bei einer Viereck-Fahrt bei Benutzung des globalen Bildverarbeitungssystem und des Kompasssensors

11 Wegplanung

In diesem Kapitel wird beschrieben, nach welchen Kriterien der Algorithmus zur Wegplanung ausgesucht wurde. Anschließend wird die konkrete Implementierung der Wegplanung erläutert.

11.1 Auswahl

Basierend auf den Ergebnissen aus Kapitel 5.8 wird in diesem Kapitel untersucht, welcher Algorithmus am besten zur Lösung der Aufgabenstellung geeignet ist.

Die Methoden Straßenkarten (s. Kapitel 5.1), Segmentierung des Raumes (s. Kapitel 5.2) und Potentialfeld (s. Kapitel 5.3) kamen nicht in die nähere Auswahl, da es sich um allgemeine Methoden zur Wegplanung handeln, die nicht ohne größere Modifikationen auf das aktuelle Problem anwendbar sind. Die CMU-Methode und die S-Kurven-Methoden wurden bereits im Rahmen des Roboterfußball-Projektes implementiert. Da die CMU-Methode die Einschränkungen der differentiellen zweirädrigen Roboter nicht beachtet, kommt der Roboter nur dann am gewünschten Zielpunkt an, wenn seine Geschwindigkeit sehr niedrig ist. Negativ an der S-Kurven-Methode ist der hohe Speicherplatzbedarf für die Implementierung und der hohe Rechenzeitbedarf. Beides sind knappe Ressourcen auf dem Roboter, weshalb auch diese beide Methoden ausscheiden.

Da eine Kollisionsvermeidung erforderlich ist, bleibt die Dreh-Fahr-Dreh-Methode in Verbindung mit der Kollisionsvermeidung VFH+. Diese Kombination berücksichtigt die Einschränkungen der differentiellen zweirädrigen Roboter, benötigt wenig Speicherplatz für die Implementierung und der Rechenzeitbedarf ist gering.

11.2 Implementierung

Die Dreh-Fahr-Dreh-Methode wird mit Hilfe des Delta-Winkel Befehles realisieren. Am Anfang der Fahrt wird der Winkel in Richtung des Zielpunktes berechnet. Anschließend dreht sich der Roboter im Stand um diesen Winkel. Ist der Roboter in Richtung des Zielpunktes ausgerichtet, beschleunigt er und fährt in Richtung des Ziels. Die Fahrrichtung wird hierbei durch die Kollisionsvermeidung so angepaßt, dass es zu keinen Kollisionen mit Hindernissen kommt. Am Zielpunkt mit einer Geschwindigkeit von 0 angekommen, dreht sich der Roboter ggf. in die gewünschte Zielausrichtung.

Im folgenden werden einige Punkte des Anfahrtsalgorithmuses ausführlicher erläutert:

Die Implementierung der Kollisionsvermeidung basiert auf VFH+ (s. Kapitel 5.7), jedoch wurde der Algorithmus an die Erfordernisse des Testsystems angepasst.

Der Raum um dem Roboter wird in 20 Kreissektoren unterteilt. Für jeden Sektor wird der Abstand zum nächsten Hindernisse bzw. zur nächsten Bande bestimmt. Im nächsten

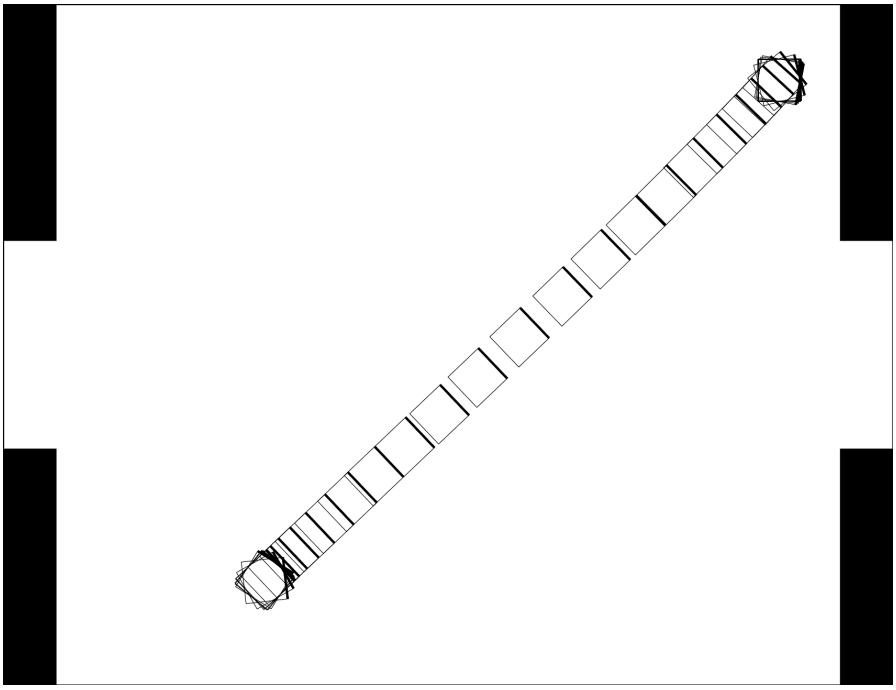

Abbildung 11.1: Beispiel für einen berechneten Weg

Schritt wird bestimmt, welche Sektoren befahrbar sind. Da im Testsystem mit wenigen Hindernissen zurechnen ist, wurde die folgende Strategie gewählt:

Alle Sektoren werden als befahrbar markiert, außer sie erfüllen eins der folgenden Kriterien:

- Abstand zum Zielpunkt ist größer als der Abstand zum nächsten Hindernis (Roboter oder Bande) im jeweiligen Sektor
- Der Sektor ist durch die dynamischen Eigenschaften des Roboters in Verbindung mit blockierenden Hindernisse nicht ohne Kollision erreichbar

Abbildung 11.2 zeigt ein entsprechendes Beispiel. Eingezeichnet sind die Sektoren, die durch Hindernisse blockiert sind. Der Übersichtshalber wurden die Sektoren, die durch die Bande blockiert sind, nicht eingezeichnet. Da die Hindernisse um die Größe des Roboters und um den Sicherheitsabstand vergrößert werden, ist ein Teil der Sektoren als blockiert markiert, obwohl kein Roboter in ihnen steht.

Auf Basis dieser Daten wird nach dem Sektor gesucht, der als befahrbar markiert ist und der am ehesten in Richtung des Zieles führt. Ausgehend von dem Sektor, in dem das Ziel liegt, werden deshalb jeweils die benachbarten Sektoren durchsucht. Diese Suche wird solange auf die benachbarten Sektoren ausgedehnt, bis ein befahrbarer Sektor gefunden wird. Danach richtet sich der Roboter mittig auf diesen Sektor aus und nachdem er ausgerichtet ist, beschleunigt er. Während der Fahrt wird nach dem oben beschriebenen Verfahren der zu befahrende Sektor kontinuierlich neu bestimmt. Die Robotergeschwindigkeit wird gemäß der in Kapitel 11.3 entwickelten Formel bestimmt. Diese Formel sorgt dafür, dass der Roboter die Zielposition mit einer Geschwindigkeit von 0 erreicht.

An der Zielposition angekommen, dreht der Roboter sich mit Hilfe des Delta-Winkel

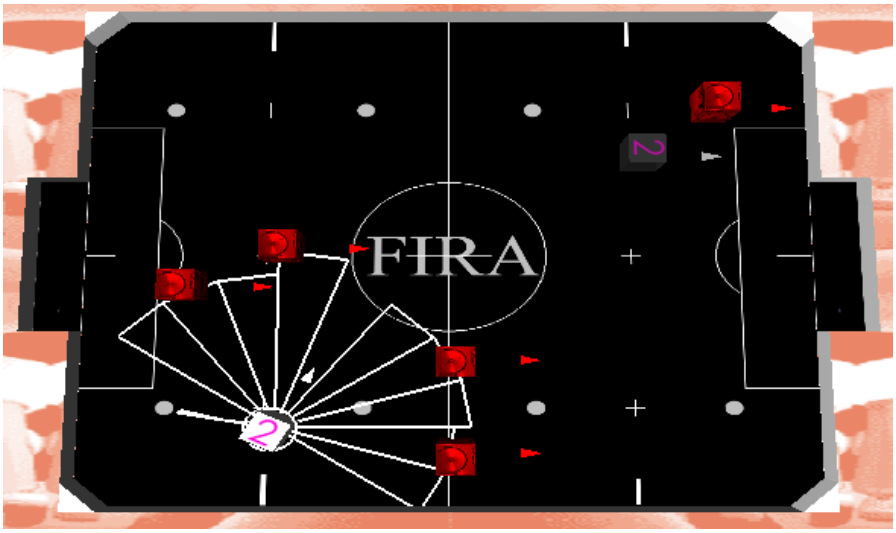

Abbildung 11.2: Mit Hilfe von VFH+ als belegt markierte Sektoren

Befehles in die gewünschte Ausrichtung. Ein Flag in der Datenstruktur des Roboters signalisiert der Handlung, dass der Roboter am Zielpunkt angekommen ist.

11.3 Geschwindigkeitsbestimmung

Bei der Berechnung der maximalen Geschwindigkeit sind verschiedene Faktoren zu beachten. Die Geschwindigkeit des Roboters v_0 darf nur so groß sein, dass er noch in der Lage ist, vor einem Hindernis oder am Zielpunkt zum Stehen zu kommen. Der Weg, den der Roboter zum Abbremsen braucht, muss also kleiner als der Abstand zum Hindernis bzw. zum Zielpunkt sein.

Sei a die Beschleunigung des Roboters. Die Zeit t, die notwendig ist um von einer Geschwindigkeit v_0 bei einer Beschleunigung von a auf 0 abzubremsen, ist durch $t = \frac{v_o}{a}$ bestimmbar und die in dieser Zeit zurückgelegte Strecke s durch:

$$s = \int_0^t v_0 - ax\,dx \tag{11.1}$$

$$= \frac{v^2}{2a} \tag{11.2}$$

Es folgt:

$$v(s) = \sqrt{2as} \tag{11.3}$$

Mit Hilfe von $v(s)$ kann die Geschwindigkeit so in Abhängigkeit von einer Strecke berechnet werden, dass der Roboter die Strecke mit der höchstmöglichen Geschwindigkeit befährt, gleichzeitig aber in der Lage ist, am Ende der Strecke noch auf eine Geschwindigkeit von 0 abzubremsen.

Die Entfernung s ist in der Wegplanung das Minimum aus der Entfernung zum Zielpunkt und dem Abstand zum nächsten Hindernis.

11.4 Evaluation

Um die Wegplanung zu testen, wurden verschiedene Tests mit unterschiedlichen Start- und Zielkonfigurationen sowie verschiedenen Positionen der Hindernisse durchgeführt.

Abbildung 11.1 zeigt einen Test, in welchem nur die Wegplanung ohne die Kollisionsvermeidung getestet wurde. Abbildungen 11.3 und 11.4 zeigen Beispiele für die Wegplanung

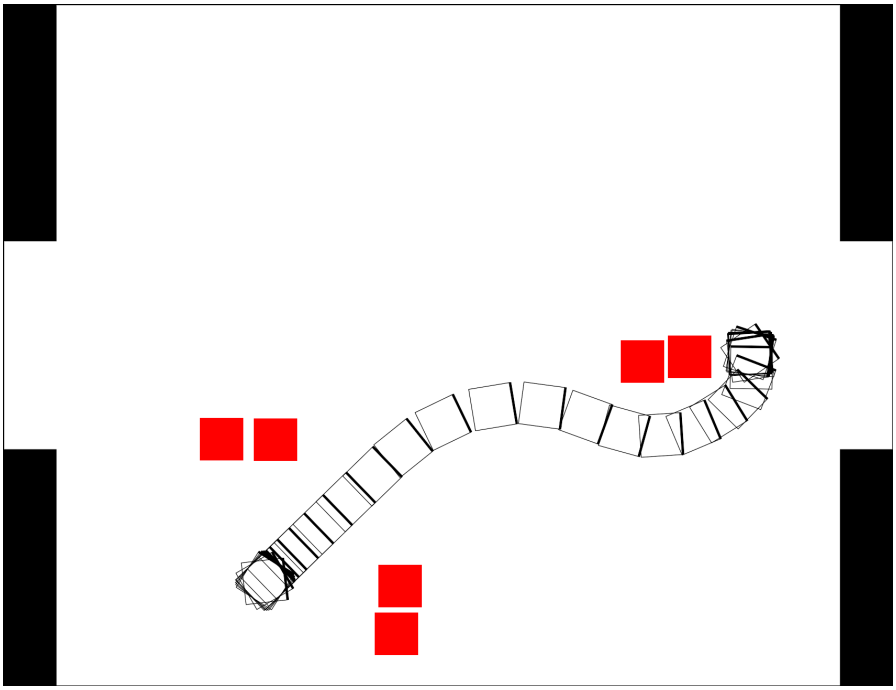

Abbildung 11.3: Beispiel für einen berechneten Weg inkl. Kollisionsvermeidung

inklusive Kollisionsvermeidung.

Die Tests zeigen, dass der Roboter seine Zielposition in Rahmen der Genauigkeit des Weltenmodells erreicht, wobei Hindernissen sicher umfahren werden. Bedingt durch den Algorithmus werden Lücken zwischen zwei Hindernissen teilweise nicht als solche erkannt. Das Problem könnte reduziert werden, indem die Anzahl der Kreissektoren erhöht wird. Da dies eine höhere Rechenzeit nach sich ziehen würde, wurde auf diese Änderung verzichtet und in Kauf genommen, dass der Algorithmus teilweise kleinere Umwege fährt.

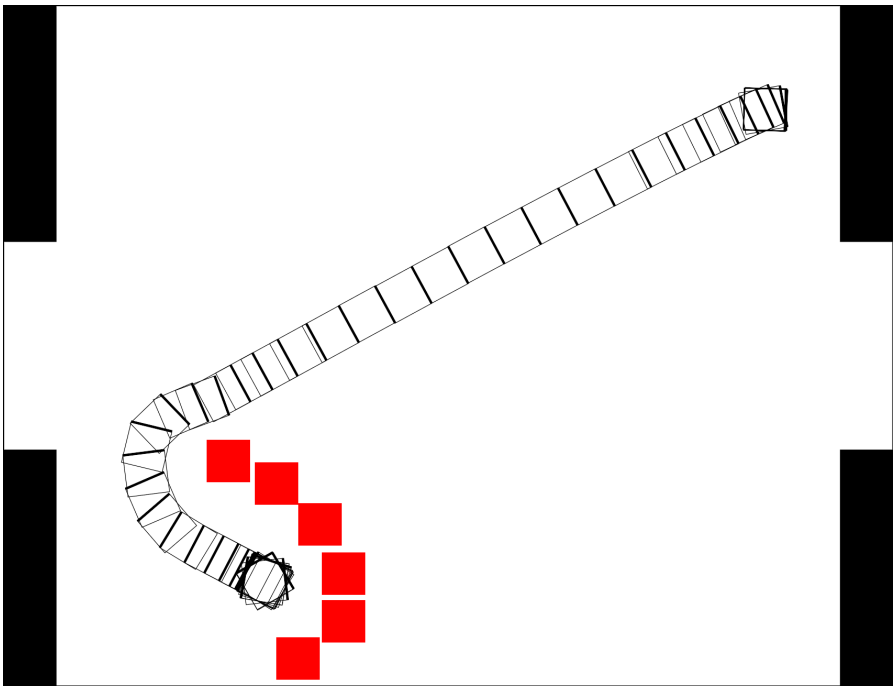

Abbildung 11.4: Beispiel für einen berechneten Weg inkl. Kollisionsvermeidung

12 Fazit und Ausblick

In den folgenden Unterkapiteln wird diese Arbeit noch einmal zusammengefasst und die Ergebnisse präsentiert. Abschließend wird ein Ausblick auf mögliche zukünftige Arbeiten gegeben, die diese Arbeit fortsetzen könnten.

12.1 Zusammenfassung

Ziel dieser Diplomarbeit war es, die Autonomie des am Lehrstuhl Informatik I eingesetzten Roboter in den Bereichen Sensorik und Wegplanung zu erhöhen. Hierzu wurden im ersten Teil die für diese Arbeit notwendigen Grundlagen erarbeitet. Es wurde auf mobile Roboter und Sensoren eingegangen; Algorithmen zur Fusion von Sensordaten und zur Wegplanung wurden präsentiert. Außerdem wurde das vorhandene Testsystem vorgestellt.

Der zweite Teil dieser Diplomarbeit beschäftigte sich mit der Realisierung der Aufgabenstellung. Es wurden Sensoren, die der relativen und absoluten Positionsbestimmung dienen, bezüglich den Kriterien Messabweichung und Latenzzeit getestet. Mit Hilfe des Kalmanfilters werden die Messwerte der verfügbaren Sensoren zu einem Weltmodell zusammengeführt. Basierend auf diesem Weltmodell ist der Roboter in der Lage mit Hilfe des Dreh-Fahr-Dreh Algorithmus von einer Position zu einer anderen Position zu fahren. Als Kollisionsvermeidung kommt optional die Vector-Field-Histogramm+-Methode zum Einsatz. Da diese Diplomarbeit keine konkrete Aufgabe für den Roboter vorgesehen hat, wurden zwei einfache Handlungen zum Testen implementiert. Bei der ersten Handlung fährt der Roboter ein Viereck auf dem Spielfeld ab. Die zweite Handlung veranlasst den Roboter eine Position auf dem Spielfeld anzufahren. Bei beiden Handlungen ist ein menschlicher Eingriff nur zum Starten der Handlung notwendig. Nach dem Starten erledigt der Roboter die ihm gestellte Aufgabe autonom.

12.2 Ergebnisse

Der zur Sensorfusion benutzte Kalmanfilter ermöglicht auch unter den gegeben Einschränkungen bzgl. Rechenleistung und Speicher eine effektive Erstellung eines Weltmodells. Die notwendige Reduzierung des Zustandsvektors auf die X- und Y-Position sowie die Ausrichtung hat keine negativen Auswirkungen. Zum Einsatz kommen das globale und lokale Bildverarbeitungssystem, der Kompasssensor, der Beschleunigungssensor und die Radencoder. Um weitere Sensoren zu integieren ist hauptsächlich die Bestimmung der H- und R-Matrix notwendig. Aufgrund von Problemen mit der Hardware konnte das Zusammenspiel aller Sensoren jedoch nicht in der Realität getestet werden, sondern nur im Simulator. In der Realität konnte abschließend nur das Zusammenspiel von Radencodern und dem globalen Bildverarbeitungssystem getestet werden. Beide Tests waren

erfolgreich und lieferten ein konsistentes Weltmodell. Die Daten des Weltmodell werden zur Wegplanung genutzt.

Die Wegplanung basiert auf dem Dreh-Fahr-Dreh Algorithmus. Mit Hilfe dieses Algorithmus fährt der Roboter zuverlässig und präzise von einer Position zu einer anderen. Die Genauigkeit der Anfahrt ist direkt proportional zur Genauigkeit des Weltmodells. Sind Hindernisse auf dem Spielfeld, wird dieser Algorithmus durch die Kollisionsvermeidung Vector-Field-Histogram+ ergänzt. Die aktuelle Implementierung der Kollisionsvermeidung arbeitet für stationäre Hindernisse zuverlässig. Bei einer großen Anzahl von Hindernissen (>10) findet der Algorithmus, bedingt durch den reaktiven Ansatz, unter Umständen keinen freien Sektor in Richtung des Zieles.

12.3 Ausblick

Im Rahmen des Roboterfußballs plant die FIRA eine Liga, in der autonome, mobile Roboter zum Einsatz kommen sollen. Diese Arbeit kann als Grundlage für Roboter dienen, die in dieser Liga spielen sollen. Bis zum einsatzbereiten Roboter sind jedoch noch einige Erweiterungen notwendig. Das Weltmodell muss um den Ball, die eigenen und die gegnerischen Roboter erweitert werden. Es bietet sich an, für jedes Objekt einen separaten Kalmanfilter zu nutzen. Die Wegplanung inkl. Kollisionsvermeidung müsste mit beweglichen Hindernissen getestet werden. Um einen Ball zu spielen, müsste eine Wegplanung implementiert werden, die an der Zielposition eine vorgegebene Ausrichtung und Geschwindigkeit erreichen kann. Anschließend kann eine Strategie für den Roboterfußball programmiert werden.

Insgesamt bleibt zweifelhaft, ob diese Erweiterung auf diesem System möglich sind. Ein Großteil der Leistungsfähigkeit des TMS320F2812 geht durch die langsame Anbindung des SRAMs und durch den Compiler verloren. Die Optimierungen des Compilers scheinen nicht auf der Höhe der Zeit zu sein. Der Multiply-Accumulate Befehl des DSP wird nicht genutzt, Schleifen sind wenig effizient und konstante Ausdrücke im Quellcode werden teilweise erst zur Laufzeit berechnet. Für Variabelen und Programmcode kann jeweils nur ein zu nutzender Speicherbereich angegeben werden. Ist der jeweilige Speicherbereich voll belegt, muss mit Hilfe von Compileranweisungen für jede zusätzliche Funktion oder Variable festgelegt werden, in welchem Speicherbereich sie liegen soll. Insgesamt haben diese Fehler und Probleme mit der Hardware die Implementierung erheblich verzögert. Ein anderer Prozessor mit einer besseren Speicheranbindung und einem besseren Compiler wäre somit wünschenswert.

Da die Zugriffe auf die Hardware in einem Modul gekapselt sind, sollte es mit wenigen Änderungen möglich sein, diese Implementierung auch auf anderen Plattformen zum Laufen zu bringen, sofern ein C-Compiler verfügbar ist.

Teil III

Anhang

A Benchmark TMS320F2812

A.1 Ausführungsgeschwindigkeiten

Wie in Kapitel 6 beschrieben, verfügt der Roboter über 512 KiW externen SRAM. Um die Geschwindigkeit des SRAM im Vergleich zum internen SARAM zu ermitteln, wurde ein Benchmark implementiert. Als Basis dient ein einfacher Bubble-Sort Algorithmus, der 100 Zahlen eines Arrays sortiert. Das Array wir bei jeden Durchlauf mit den gleichen Pseudo-Zufallszahlen initialisiert. Da sowohl der Code als auch das Array im internen oder externen Speicher liegen können, mussten vier Varianten getestet werden. Die Angaben in der Tabelle sind die Laufzeiten in Millisekunden.

	Daten intern	Daten extern
Code intern	1,3	5,8
Code extern	27,1	31,5

A.2 Benötigte Rechnenzeit

Wie in Kapitel 6 beschrieben verfügt der Prozessor über keine Einheit zur Berechnung von Gleitkommazahlen. Die Berechnung von Gleitkommazahlen muss deshalb softwäremäßig durch die Integer-Einheit erfolgen. Die folgende Tabelle zeigt die benötigte Zeit für verschiedene Berechnungen.

	Gleitkommazahlen (float)	IQ24
10.000 Sinus-Aufrufe	145,5	4,8
10.000 Divisionen	30,3	6,4
10.000 Additionen	25,1	0,4

Literaturverzeichnis

[AI85] American National Standards Institute ; Institute of Electrical and Electronic Engineers: IEEE standard for binary floating-point arithmetic. ANSI/IEEE Std 754-1985, 1985

[Asi86] Asimov, Isaac: Alle Robotergeschichten. Bergisch Gladbach : Lübbe, 1986

[Bay63] Bayes, Thomas: An essay towards solving a problem in the doctrine of chances. In: Philosophical Transactions of the Royal Society of London 53 (1763), S. 370–418

[BF94] Borenstein, J. ; Feng, L.: UMBmark - A Method for Measuring, Comparing, and Correcting Dead-reckoning Errors in Mobile Robots / University of Michigan. 1994. – Forschungsbericht

[BFH+03] Bursch, S. ; Fennecker, C. ; Hildebrand, L. ; Reusch, B. ; Sölter, M.: Path planning for mobile robots using the S-curve algorithm. In: FIRA World Congress Austria 2003, 2003

[BK91] Borenstein, J. ; Koren, Y.: The Vector Field Histogram - Fast Obstacle Avoidance For Mobile Robots. In: IEEE Transactions on Robotics and Automation 7 (1991), Nr. 3, S. 278–288

[BP04] B. Putz, P. K. M. Han H. M. Han: Implementation Of Acceleration Sensors Improving Dynamic Behaviour Of a Mobile Mini Robot. In: IAV2004-Preprints, 5th IFAC/EURON SYMPOSIUM ON INTELLIGENT AUTONOMOUS VEHICLES, 2004

[BRF96] Borenstein, J. ; R., Everett H. ; Feng, L.: 'Where am I?' Systems and Methods for Mobile Robot Positioning / The University of Michigan. 1996. – Forschungsbericht

[DM00] Dudek, Gregory ; Michael, Jenkin: Computational Principles of Mobile Robotics. Cambridge : Cambridge University Press, 2000

[Doy95] Doyle, Alexander B.: Algorithms and Computational Techniques for Robot Path Planning, University of Wales, Diss., 1995

[Dub57] Dubins, L. E.: On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. In: American Journal of Mathematics 79 (1957), S. 497–516

[Eve95] Everett, H. R.: Sensors for Mobile Robots: Theory and Application. South Avenue : A. K. Peters, Ltd., 1995

[Fau06a] Faulahber (Hrsg.): DC-Micromotors Series 2232. Faulahber, 2006. http://www.faulhaber-group.com/uploadpk/e_2232SR_DFF.pdf

[Fau06b] Faulhaber (Hrsg.): Magnetic Encoders. Faulhaber, Feb 2006. http://www.faulhaber-group.com/uploadpk/e_IE2512_DFF.pdf

[FHL+03] Fox, D. ; Hightower, J. ; Liao, L. ; Schulz, D. ; Borriello, G.: Bayesian Filters for Location Estimation. In: IEEE Pervasive Computing Magazine July-September 2003 (2003), S. 24–33

[Fir06a] Federation of International Robot-soccer Association. http://www.fira.net. Version: Oct 2006

[Fir06b] Micro Robot World Cup Soccer Tournament Rules. http://www.fira.net/soccer/mirosot/overview.html. Version: Oct 2006

[Fra04] Fraden, Jacob (Hrsg.): Handbook of modern sensors. 3. ed. New York : Springer, 2004

[Fri06] Frinken, Volkmar: Ein System zur 3D-Rekonstruktion von Hindernissen für Roboter der FIRA-MiroSot Klasse, Universität Dortmund, Fachbereich Informatik, Diplomarbeit, 2006

[Hon06] Honeywell International Inc. (Hrsg.): HMC6352 Digital Compass Solution. Plymouth: Honeywell International Inc., Jan 2006. http://www.ssec.honeywell.com/magnetic/datasheets/HMC6352.pdf

[Kal60] Kalman, R.: A New Approach to Linear Filtering and Prediction Problems. In: Transactions of the ASME–Journal of Basic Engineering 82 (1960), Nr. Series D, S. 35–45

[Kel00] Kelly, Alonzo: Some Useful Results for Closed-Form Propagation of Error in Vehicle Odometry / The Robotics Institute, Carnegie Mellon University. 2000 (CMU-RI-TR-00-20). – Forschungsbericht

[Kha86] Khatib, O: Real-time obstacle avoidance for manipulators and mobile robots. In: Int. J. Rob. Res. 5 (1986), Nr. 1, S. 90–98. – ISSN 0278–3649

[KKKS04] Kim, Jong-Hwan ; Kim, Dong-Han ; Kim, Yong-Jae ; Seow, Kiam-Tian: Soccer Robotics. Springer, 2004

[Klu01] Klute, Thomas: Konzeption und Aufbau eines teilautonomen Fußballroboters, Universitiy of Dortmund, Germany, Diplomarbeit, 2001

[Kor91] Koren, Borenstein J. Y.: Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation. In: Proceedings of the IEEE International Conference on Robotics and Automation Sacramento (1991), April 7-12, S. pages 1398–1404

[Kra05] Kranz, Christian: Using the Siemens S65-Display, Oct 2005. http://www.superkranz.de/christian/S65_Display/DisplayIndex.html

[May79] Maybeck, Peter S.: Mathematics in Science and Engineering. Bd. 141: Stochastic models, estimation, and control. New York, 1979

[Mey06] Meyers Großes Taschenlexikon in 24 Bänden. Zehnte Auflage. Mannheim : Bibliographisches Institut, 2006

[Moh01] Mohr, Markus: Realisierung der Sterung eines autonomen Kleinroboters, Universtiät Dortmund, Fakultät Marschinenbau, Diplomarbeit, 2001

[Neg03] Negenborn, Rudy: Robot Localization and Kalman Filters – On finding your position in a noisy world, University of Utrecht, the Netherlands, Diplomarbeit, 2003

[Nol90] Nolting, Wolfgang: Grundkurs Theoretische Physik 2. Analytische Mechanik. Wiesbaden : Vieweg, 1990

[Phi01] Philips Semiconductors (Hrsg.): THE I2C-BUS SPECIFICATION. Version 2.1. Philips Semiconductors, Jan 2001

[Pro01] Projektgruppe 362: Endbericht der Projektgruppe 362 "Roboterfußball"/ Universität Dortmund, FB Informatik. 2001. – Forschungsbericht

[Pro03] Projektgruppe 416: Endbericht der Projektgruppe 416 / Universität Dortmund, FB Informatik. 2003. – Forschungsbericht

[Pro05] Projektgruppe 449: Endbericht der Projektgruppe 449 / Universität Dortmund, FB Informatik. 2005. – Forschungsbericht

[Pro06] Projektgruppe 472: Endbericht der Projektgruppe 472 "VisiRobs"/ Universität Dortmund, FB Informatik. 2006. – Forschungsbericht

[Rad04a] Radiometrix (Hrsg.): 433MHz high speed FM radio transceiver module. England: Radiometrix, Nov 2004

[Rad04b] Radiometrix (Hrsg.): 869/914MHz high speed FM radio transceiver module. England: Radiometrix, Nov 2004

[Rad04c] Radiometrix (Hrsg.): Low Power UHF Data Transceiver Module. England: Radiometrix, Nov 2004

[Rad06] Radig, Ulrich: AVR-MMC/SD - Speicherriese für einen Zwerg, Jan 2006. `http://www.ulrichradig.de/home/index.php/avr/mmc-sd`

[RAG04] Ristic, Branko ; Arulampalam, Sanjeev ; Gordon, Neil: Beyond the Kalman Filter Particle Filters for Tracking Applications. Boston : Artech House, 2004

[Rob04] RoboCup 2003: Robot Soccer World Cup VII. Bd. 3020. Springer, 2004 (Lecture Notes in Computer Science)

[Rob05] RoboCup 2004: Robot Soccer World Cup VIII. Bd. 3276. Springer, 2005 (Lecture Notes in Computer Science). – ISBN 3–540–25046–8

[Rob06a] RoboCup. `http://www.robocup.org`. Version: Oct 2006

[Rob06b] RoboCup 2005: Robot Soccer World Cup IX. Bd. 4020. Springer, 2006 (Lecture Notes in Computer Science)

[RS90] Reeds, J. A. ; Shepp, L. A.: Optimal paths for a car that goes both forwards and backwards. In: Pacific Journal of Mathematics 145 (1990), Nr. 2, S. 367–393

[Sch05] Schmitz, Norbert: Satellitenortung und Sensorfusion zur Lokalisierung von Fahrzeugen in unstrukturierter Umgebung, Technischen Universität Kaiserslautern, Diplomarbeit, 2005

[Sch06] Schulz, Simon: Entwicklung eines kompakten, DSP- und FPGA-basierten Bildverarbeitungssystems mit Unterstützung mehrerer Kameras zur Navigation mobiler Roboter, Universität Dortmund, Fachbereich Informatik, Diplomarbeit, 2006

[SN04] Siegwart, R. ; Nourbakhsh, I.: Introduction to Autonomous Mobile Robots. Cambridge : Bradford Book, 2004

[STM05] STMicroelectronics (Hrsg.): Inertial Sensor: 3-Axis - 2g Digital Output Linear Accelerometer. STMicroelectronics, May 2005. `http://www.stmicroelectronics.com/stonline/products/literature/ds/10175.pdf`

[Stö04] Stöcker, Horst: Taschenbuch der Physik. Frankfurt : Deutsch (Harri), 2004

[TI02] Texas Instruments (Hrsg.): TMS320F2810 and TMS320F2812 32-Bit Fixed-Point Flash DSPs Product Bulletin. Texas Instruments, 2002

[TI03] Texas Instruments (Hrsg.): IQmath Library – A Virtual Floating Point Engine. 1.4d. Texas Instruments, March 2003

[TI04] Texas Instruments (Hrsg.): Online Training Course: IQ Math on the Texas Instruments TMS320C28x DSP. Texas Instruments, 9 2004

[TI05] Texas Instruments (Hrsg.): TMS320F2810, TMS320F2811, TMS320F2812, TMS320C2810, TMS320C2811, TMS320C2812 Digital Signal Processors Data Manual. Texas Instruments, Okt 2005

[UB98] Ulrich, Iwan ; Borenstein, Johann: VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. In: IEEE International Conference on Robotics and Automation, 1998

[VBA+99] Veloso, Manuela ; Bowling, Michael ; Achim, Sorin ; Han, Kwun ; Stone, Peter: The CMUnited-98 Champion Small-Robot Team. In: Lecture Notes in Computer Science 1604 (1999), S. 77

[VSHA98] Veloso, Manuela ; Stone, Peter ; Han, Kwun ; Achim, Sorin: The CMUnited-97 Small-Robot Team. In: Kitano, Hiroaki (Hrsg.): RoboCup-97: Robot Soccer World Cup I. Berlin : Springer Verlag, 1998, S. 242–256

[WJ04] Weiss, Norman ; Jesse, Norbert: Towards Local Vision in Centralized Robot Soccer Leagues: A Robust and Flexible Vision System Also Allowing Varying Degrees of Robot Autonomy. In: Proceedings of the 5th International Conference on Simulated Evolution and Learning (SEAL04) and 2004 FIRA Robot World Congress, 2004