Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Informatik

Studienarbeit

Selbstlokalisierung humanoider Roboter im RoboCup

vorgelegt von

Naja v. Schmude
am 28. September 2011

Betreuer: Hamid Reza Moballegh und Daniel Seifert

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Studienarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Berlin, den 28. September 2011

Inhaltsverzeichnis

1 Einleitung

Die folgende Arbeit beschäftigt sich mit der Selbstlokalisierung humanoider Roboter im RoboCup. Im Speziellen wurde die Selbstlokalisierung der *FUmanoids* überarbeitet. Bevor sich möglichen Lösungsansätzen in Kapitel 2 und 3 gewidmet und die konkrete Implementierung im 4. Kapitel erläutert wird, soll zunächst die Problemstellung erklärt und das Szenario beschrieben werden. Die Arbeit endet mit einer experimentellen Auswertung in Kapitel 5 und liefert einen Ausblick für zukünftige Entwicklungen im Bereich der Lokalisierung.

1.1 Selbstlokalisierung

Das Lokalisierungsproblem bezeichnet die Problemstellung, dass ein Roboter seine Position in einer bekannten oder unbekannten Umgebung selbstständig bestimmen können soll. Dies ist eine notwendige Fähigkeit für autonom agierende Roboter. Je nach Einsatzgebiet und verwendeten Sensoren ist dies unterschiedlich schwierig zu lösen. Bei autonomen Fahrzeugen wird z.B. durch die Verwendung von GPS und bekanntem Kartenmaterial dieses Problem auf ein Minimum reduziert. Bei Bergungsrobotern, die in unwegsamem und unbekanntem Gebiet navigieren, ist es schwieriger. Hier kann nicht auf Kartenmaterial o.Ä. zurückgegriffen werden und GPS hilft demnach nicht weiter. Oft kommen Kameras und Laserscanner zum Einsatz. Zusätzlich zur Lokalisierung innerhalb der bereits erschlossenen Umgebung muss eine Karte von noch unbekannten Gebieten angelegt werden. Dies wird auch als *Simultaneous Localization and Mapping* bezeichnet.

Im RoboCup Soccer ist durch das Spielfeld eine klar definierte Umgebung gegeben, sodass der Roboter durch seine Sensorik – beim RoboCup sind dies meist Kameras – die Umgebung wahrnehmen und mit der bekannten Anordnung auf dem Spielfeld in Übereinstimmung bringen kann. Wenn eine Übereinstimmung gefunden wurde, kann daraus

sofort die Position des Roboters auf dem Spielfeld extrahiert werden. Mit genau dieser Fragestellung, wie Sensordaten und bekannte Umwelt in Übereinstimmung gebracht werden können, beschäftigt sich die vorliegende Arbeit.

1.2 RoboCup

Der RoboCup ist eine international agierende Initiative zur Förderung der künstlichen Intelligenz im Bereich Robotik. Bei der Gründung 1997 war das Hauptziel des RoboCup, bis zum Jahr 2050 ein autonom agierendes Roboter-Team zu entwickeln, das fähig ist die dann amtierende menschliche FIFA-Weltmeistermannschaft zu schlagen.

Neben der dazu gegründeten *RoboCup Soccer Liga*, in der es um Fussball spielen in unterschiedlichen Variationen geht, existieren heute auch Ligen für Haushaltsroboter (*RoboCup @Home*), Bergungsroboter (*RoboCup Rescue*) und zur Förderung von Schülerinnen und Schülern der *RoboCup Junior*.

Der RoboCup Soccer Bereich teilt sich in unterschiedliche Ligen auf:

RoboCup Soccer Humanoid Liga In dieser Liga treten nach humanoiden Maßstäben gebaute, also menschenähnliche, Roboter gegeneinander an. Es sind nur menschenähnliche Bewegungen und Sensoren erlaubt. Die Roboter agieren autonom und dürfen nur mittels WLAN kommunizieren. Es existieren drei Unterklassen, die jeweils die Größe der Roboter begrenzen: In der *Kid-Size* spielen Roboter mit einer Größe von $30cm$ bis $60cm$, in der *Teen-Size* von $100cm$ bis $120cm$ und in der *Adult-Size* ist jede Größe zwischen $130cm$ und $160cm$ zulässig.

RoboCup Standard Plattform Liga In der Standard Plattform Liga treten alle Teams mit baugleichen Robotern an, sodass sich hier nur auf die Software konzentriert werden muss. Zurzeit werden die humanoiden Nao-Roboter von Aldebaran verwendet. Das Setup ist ähnlich wie bei der humanoiden Kid-Size Liga.

RoboCup Middle Size Liga Hier treten bis zu sechs Roboter gegeneinander an. Die Roboter sind nicht menschenähnlich, sondern fahren auf Rädern autonom.

RoboCup Small Size Liga Die fahrenden Roboter in dieser Liga agieren nicht autonom sondern werden zentral gesteuert. Dies ist die älteste RoboCup Soccer Liga.

1.2.1 Humanoide Kid-Size Liga

In der humanoiden Kid-Size Liga treten jeweils drei Roboter gegeneinander an. Zur Unterscheidung der Teams tragen die Kontrahenten Cyan- bzw. Magentafarbene Markierungen. Das Spielfeld ist zur Zeit 6×4 Meter groß. Die Tore sind $150cm$ breit und zur besseren Unterscheidbarkeit in blau bzw. gelb gefärbt. Um die Lokalisierung zusätzlich zu vereinfachen, existieren zwei Säulen am Rand der Mittellinie[14].

Die Regeln werden jedoch von Jahr zu Jahr angepasst und gehen dabei von diesem sehr speziellen Szenario hin zu einem normalem Fussballfeld. So ist z.B. abzusehen, dass in den kommenden Jahren nicht nur die Säulen wegfallen werden, sondern zusätzlich auch beide Tore weiß werden, wie es bei der Middle Size schon länger der Fall ist. Auch eine Vergrößerung des Spielfeldes ist nicht auszuschließen. Diese Veränderungen machen es notwendig, dass die Lokalisierung der Roboter ständig weiterentwickelt wird und mit den weniger werdenden *Landmarks* (d.h. markante, eindeutig auf dem Spielfeld zu identifizierende Merkmale wie Tore oder Säulen) zurecht kommen muss.

Eine Ansicht des Spielfelds ist in Abbildung 1.1 zu sehen.

Abbildung 1.1: Spielfeld der humanoiden Kid-Size Liga (Quelle: [14])

1.2.2 Team FUmanoids

Das Team *FUmanoids* wurde 2006 als Nachfolgeprojekt der *FU-Fighters* an der Freien Universität Berlin gegründet. Im Gegensatz zu den FU-Fighters, die in der Small Size und

Middle Size Liga sehr erfolgreich gespielt haben, sind die FUmanoids in der humanoiden Kid-Size Liga aktiv. Das Team wird als studentisches Projekt geführt, in dem ca. 15 Studenten aktiv mitwirken. 2009 und 2010 konnte das Team im RoboCup den zweiten Platz erreichen.

Zur Weltmeisterschaft 2011 in Istanbul wurde eine neue Generation von Robotern entworfen. Diese sind $60cm$ groß bei einem Gewicht von über $4kg$. Für die Beweglichkeit sorgen 21 Dynamixel-Motoren, jeweils sieben pro Bein, drei pro Arm und ein Motor dient zur Kopfsteuerung.

Als Hauptsensor dient eine Logitech Quickcam 9000 Pro USB-Kamera, die mit einer Fischaugenlinse bestückt wurde, um ein Sichtfeld von 180×90 Grad zu erreichen. Zusätzlich ist der Roboter mit einem 5-Achsen Inertialsensor (IMU) ausgestattet, um die Lage der Kamera im Raum zu bestimmen. Als Recheneinheit dient ein IGEPv2 Board mit $1GHz$ Rechenleistung.

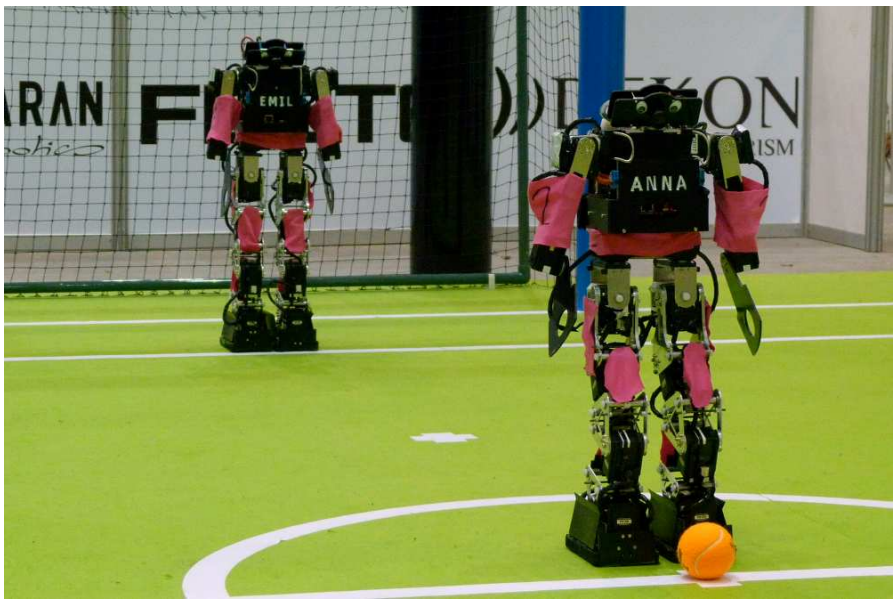


Abbildung 1.2: Roboter Anna und Emil aus der 2011er Familie der FUmanoids

2 Grundlagen zur Selbstlokalisierung mobiler Roboter

2.1 Lokalisierungsarten

Bei mobilen Robotern kann zwischen drei verschiedenen Arten von Lokalisierung unterschieden werden.

Lokale Lokalisierung Die lokale Lokalisierung kann als Positionstracking bezeichnet werden. Initial ist die Position des Roboters bekannt und im Folgenden müssen lediglich die Bewegungen des Roboters verfolgt werden. Eine unimodale Modellierung (z.B. über eine Gaußverteilung) ist in solch einem Szenario ausreichend.

Globale Lokalisierung Bei der globalen Lokalisierung ist die initiale Position des Roboters unbekannt. Daher reicht in diesen Fällen eine unimodale Modellierung nicht aus, denn durch existierende Symmetrien o.ä. in der Welt müssen mehrere Hypothesen gleichzeitig verfolgt werden können. Daher wird hier eine multimodale Modellierung benötigt. Die globale Lokalisierung ist entsprechend schwieriger als das Positionstracking.

Kidnapped Robot Diese Art der Lokalisierung ist eine Verallgemeinerung der globalen Lokalisierung, bei der die Positionsbestimmung zusätzlich dadurch erschwert wird, dass ein Roboter gekidnappt und zu einer anderen Position transportiert werden kann, ohne dass der Roboter davon etwas merkt. Auch in Szenarios, wo der Roboter nie manuell umgesetzt wird, ist ein solcher Lokalisierungsansatz hilfreich, da kein Algorithmus fehlerfrei ist und durch die Betrachtung als Kidnapped Robot von einer falschen Annahme wieder abgekommen werden kann.

Da im RoboCup Szenario die initiale Position der Roboter häufig nicht bekannt ist, wird ein globaler Lokalisierungsansatz benötigt. Da die Roboter zudem auch manuell umplatziert werden können (in den humanoiden Ligen z.B. durch die Robot-Handler

oder Schiedsrichter) oder durch Stürze ihre Orientierung verlieren, muss die Lokalisierung robust genug sein, um diese Probleme auch lösen zu können. Im Folgenden wird nun einer der am häufigsten verwendeten Ansätze zur Lokalisierung erläutert: die *Monte-Carlo-Lokalisierung* (MCL)[18]. Dieser Ansatz erlaubt die Positionsbestimmung des Roboters in einer bekannten Umgebung, wobei die anfängliche Startposition unbekannt ist. Auch das Kidnapped Robot Problem kann durch die MCL abgedeckt werden.

2.2 Monte-Carlo-Lokalisierung

Bei der Monte-Carlo-Lokalisierung wird ein *Partikelfilter* verwendet, dessen Partikel die möglichen Positionen (x, y, θ) des Roboters in seiner Umwelt darstellen. Initial sind die Partikel zufällig verteilt. In jedem Zyklus werden diejenigen Partikel behalten und vervielfältigt, deren Positionen am ehesten mit der tatsächlichen Position des Roboters übereinstimmen. Andere Partikel sterben nach und nach aus, sodass die Partikelanzahl konstant bleibt. Das Vervielfältigen und Aussterben lassen von Partikeln wird als *Resampling* bezeichnet. Die Entscheidung, ob ein Partikel die Roboterposition gut oder schlecht repräsentiert, wird über eine *Bewertungsfunktion* getroffen, welche die Sensordaten des Roboters verwendet. Wenn die gemessenen Sensordaten gut mit dem Modell an der aktuellen Partikelposition korrelieren, wird der *Belief*, d.h. die Gewissheit sich an einer bestimmten Position zu befinden, in dem Partikel hoch eingestuft, andernfalls wird er niedrig gesetzt.

Wenn das *Bewegungsmodell* des Roboters bekannt ist, können zusätzlich alle Partikel vor der Bewertung entsprechend der letzten Bewegung verschoben werden.

Eine Visualisierung der Funktionsweise des Partikelfilters ist in Abbildung 2.1 zu finden. x beschreibt dabei den aktuellen Zustand (hier die Position) des Roboters, $bel(x)$ ist demzufolge der Belief in die Position. $p(z|x)$ gibt die Wahrscheinlichkeit an, dass die Sensordaten z bei Zustand x gemessen werden können, spiegelt also die Korrelation von Modell zu Sensordaten wieder.
In Teil a) ist der Initialzustand dargestellt, der zunächst entsprechend der Übereinstimmung mit den Sensordaten bewertet (b) und entsprechend resampelt wird. Damit endet der erste Zyklus. In Bild c) wird die Partikelmenge durch das Bewegungsmodell verschoben; es ist hier zu erkennen, dass die Partikeldichte durch das Resampling um die vorher gut bewerteten Partikel zugenommen hat. Dann wird für die neue Position in Teil d) erneut der Belief berechnet und entsprechend resampelt.

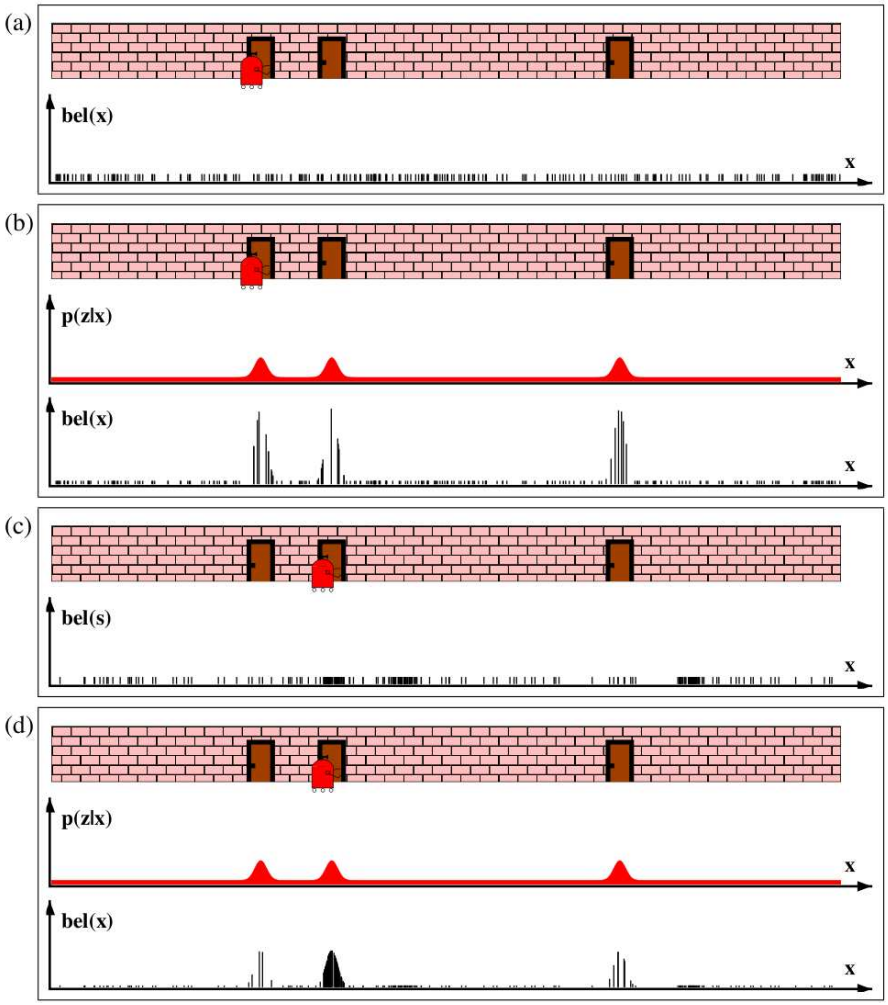


Abbildung 2.1: Monte-Carlo-Lokalisierung (Quelle: [18, S. 251])

2.2.1 Augmented MCL Algorithmus

Bis jetzt ist noch nicht ersichtlich, wie das Kidnapped Robot Problem genau durch die MCL gelöst werden kann. In der bisher beschriebenen Form ist dies auch nicht möglich, der sogenannte *Augmented MCL Algorithmus*[18] erlaubt aber durch zufälliges Einstreuen neuer Partikel die Lösung. Im Folgenden soll nun diese Variante (Pseudocode in Algorithmus 1 dargestellt) erläutert werden.

Die Idee des Augmented-MCL ist, dass neben dem Resampling zufällig neue Partikel *injiziert* werden. Im Falle des Kidnapping sorgen diese zufällig verteilten Partikel dafür, dass die neue Position des Roboters auch abgedeckt wird.

Algorithmus 1 Augmented MCL Algorithmus

1: w_{slow}, w_{fast}
2: **for** $t = 0 \rightarrow \infty$ **do**
3: $\bar{\chi}_t = \chi_t = \emptyset$
4: $w_{avg} \leftarrow 0$
5: **for** $m = 1 \rightarrow M$ **do**
6: $x_t^{[m]} \leftarrow$ Update durch Motion-Model$(u_t, x_{t-1}^{[m]})$
7: $w_t^{[m]} \leftarrow$ Gewichtung durch Sensor-Model$(z_t, x_t^{[m]}, m)$
8: $\bar{\chi}_t \leftarrow \bar{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
9: $w_{avg} \leftarrow w_{avg} + \frac{1}{M} w_t^{[m]}$
10: **end for**
11: $w_{slow} \leftarrow w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
12: $w_{fast} \leftarrow w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
13: **for** $m = 1 \rightarrow M$ **do**
14: **if** Mit Wahrscheinlichkeit $\max(0.0, 1.0 - \frac{w_{fast}}{w_{slow}})$ **then**
15: füge zufälligen neuen Partikel zu χ_t hinzu
16: **else**
17: Ziehe $i \in \{1, \ldots, M\}$ mit Wahrscheinlichkeit $\propto w_t^{[i]}$
18: Füge $x_t^{[i]}$ zu χ_t hinzu
19: **end if**
20: **end for**
21: **end for**

Es stellt sich nun die Frage, wie viele solcher zufälligen Partikel gestreut werden sollen. Es erscheint sinnvoll die Anzahl an neuen Partikeln adaptiv zu bestimmen. Dazu wird die durchschnittliche Wahrscheinlichkeit w_{avg}, dass die Sensormessungen mit dem Modell übereinstimmen, über einen kurzen und einen längeren Zeitraum (w_{fast} bzw. w_{slow}) betrachtet. Aus dem Verhältnis von diesen beiden Wahrscheinlichkeiten zueinander wird dann die Anzahl der neu zu streuenden Partikel bestimmt. Ist die kurzfristige Wahrscheinlichkeit größer oder gleich der langfristigen Wahrscheinlichkeit, so werden keine neuen Partikel eingefügt. Ist sie jedoch geringer, so werden neue Partikel eingefügt. Solch ein Fall tritt nur dann auf, wenn plötzlich die Sensormessungen nicht mehr mit dem Modell übereinstimmen, was vor allem beim Kidnapped Robot der Fall ist.

Statt einfach zufällige Partikel zu streuen, können Positionshypothesen direkt aus den Sensordaten generiert werden. Dies bietet sich vor allem dann an, wenn es feste Landmarks gibt, wie sie auch im RoboCup Szenario vorkommen (z.B. feste Position der Tore und der Feldlinien).

2.2.2 Resampling

Die Zeilen 17-18 des Algorithmus beschreiben den Resampling-Schritt. Die hier aufgezeigte Form wird auch als *Importance Resampling* bezeichnet, bei dem die Partikel unabhängig voneinander gesampelt werden.
Ein Problem dieser Methode kann durch folgendes Extrembeispiel verdeutlicht werden: Ein Roboter sei unbeweglich und bekomme keine Sensordaten. Initial sei die Partikelmenge zufällig verteilt. Da es keine Bewegungen und keine Sensordaten gibt, sollte zu jedem beliebigen Zeitpunkt die Partikelmenge wie zu Beginn erhalten bleiben. Doch durch den Resampling-Schritt geschieht dies nicht, denn es werden zufällig einige Partikel nicht mit in den nächsten Zyklus übernommen und andere tauchen hingegen mehrfach auf. Mit der Zeit wird dann nur ein Partikel (bzw. viele Kopien dieses Partikels) überleben, so als wäre der Roboter vollkommen sicher über seine Position.

Der *Low Variance Sampler* umgeht dieses Problem, indem die Partikel nicht unabhängig voneinander gesampelt werden, sondern abhängig voneinander. Dazu wird eine Zufallszahl r im Intervall von $[0; \frac{1}{M}]$ gewählt, wobei M die Anzahl der Partikel ist. Zu r wird dann sukzessive $\frac{1}{M}$ hinzugefügt. Diese Werte dienen als „Zeiger“ in die Partikelmenge, die durch ihre Gewichte repräsentiert wird, und alle „getroffenen“ Partikel werden gesampelt. Eine Veranschaulichung von diesem Prozess ist in Abbildung 2.2 zu sehen.

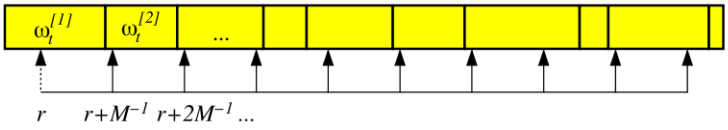


Abbildung 2.2: Low Variance Sampler (Quelle: [18, S. 111])

Es ist offensichtlich, dass diese Sampling-Variante nicht an dem genannten Problem scheitert, denn ohne Input von Sensordaten oder Bewegungsmodell bekommen alle Partikel den gleichen Belief zugewiesen. In Folge dessen wird jeder Partikel dann exakt einmal getroffen.

3 Verwandte Arbeiten

In den humanoiden Ligen des RoboCup kommen unterschiedlichste Ansätze zur Lokalisierung der Roboter auf dem Feld zum Einsatz.

3.1 Monte-Carlo-Lokalisierung

Einer der am weitesten verbreiteten Ansätze ist die bereits im vorangehenden Kapitel beschriebene Monte-Carlo-Lokalisierung.

Viele Teams in der Standard-Plattform-Liga als auch in der humanoiden Liga verwenden diesen Ansatz. So basierte die Lokalisierung der FUmanoids in 2009 und 2010 ebenfalls auf einem Partikelfilter[3, 8], andere Top Teams der Ligen setzen auch auf MCL, wie die Darmstadt Dribblers, CIT Brains oder auch Team NimbRo[1, 4, 6]. In der Standard-Plattform-Liga spielt u.a. B-Human mit einer Augmented MCL-basierten Lokalisierung[16].

Auf Grund der weiten Verbreitung dieser Lokalisierungsvariante wurden viele Anpassungen und Erweiterungen vorgestellt. So wurden verschiedene Vorschläge erläutert, wie mit mehrdeutigen Landmarks umgegangen werden kann, wie z.B. T- und L-Feldlinien-Features[11, 17]. Des weiteren existieren Kombinationen von Partikelfilter und Kalman-Filter, dabei wird das Ergebnis des Partikelfilters zusätzlich durch einen Kalman-Filter nachgeglättet, um noch genauere Positionsbestimmungen zu erreichen[16].

Da bei der MCL-basierten Lokalisierung viele Parameter über die Qualität der Lokalisierung entscheiden, angefangen von der Anzahl der Partikel bis hin zur Gewichtung einzelner Sensormessungen, ist das Finetuning dieser Parameter besonders wichtig. Dazu haben Burchardt et al.[2] einen Optimierungsansatz vorgestellt, der dem „Particle Swarm Optimization-Algorithmus“ folgt.

3.2 Kalman-Filter Lokalisierung

Eine weitere Möglichkeit mit der Lokalisierungsproblematik umzugehen ist die Verwendung von Kalman-Filter Implementierungen. Der Kalman-Filter ist ein auf dem Bayes-Ansatz beruhender Filter, der in seiner ursprünglichen Form nur geeignet ist lineare Gauß-Systeme zu beschreiben, d.h. ein System, in dem das Rauschen durch eine Normalverteilung beschreibbar ist und alle Zustandsübergänge und Messungen sich linear verhalten[18].

Da die Bewegungen der Roboter im Allgemeinen nicht linear sind, werden entsprechende Erweiterungen des Kalman-Filters benötigt, um nichtlineare Systeme zu beschreiben. Der *Extended Kalman-Filter* (EKF) oder der *Unscented Kalman-Filter* (UKF) lösen dieses Problem, wobei der UKF gleich gute oder bessere Ergebnisse als der EKF erzielen kann, dabei allerdings etwas rechenintensiver ist[18].

Diese Varianten erlauben nur eine unimodale Repräsentation, was, wie bereits erläutert, nicht ausreichend im RoboCup-Szenario ist. Eine UKF Anpassung zur Lokalisierung im RoboCup, die multimodal funktioniert, wurde in [7] beschrieben. Eine auf der EKF-Lokalisierung basierende Variation, ebenfalls zur Verfolgung mehrerer Hypothesen, wurde in [12] erläutert.

Allgemein gilt, dass Kalman-Filter basierte Lokalisierungen weniger aufwendig zu berechnen sind als Partikelfilter basierte Lokalisierungen. Dies liegt vorwiegend daran, dass mit steigender Dimension des Zustandsraums die Rechenzeit bei Partikelfiltern exponentiell wachsen kann (wesentlich mehr Partikel benötigt), wohingegen bei Kalman-Filtern die Dimension des Zustandsraums nur quadratisch in die Berechnung einfließt. Da im RoboCup der Zustandsraum meistens nur dreidimensional ist, bestehend aus Ort (x, y) und Orientierung θ, hält sich der Nachteil von Partikelfiltern in Grenzen und es überwiegt häufig die leichtere Implementierbarkeit. Zudem kann durch geeignete Methoden die Dimension des Zustandsraums gesenkt werden, indem beispielsweise versucht wird die Orientierung des Roboters θ direkt zu berechnen[13].

3.3 Constraintbasierte Lokalisierung

Eine nicht auf dem Bayes-Ansatz beruhende Lokalisierung ist die sogenannte *Constraint-basierte Lokalisierung*. Dabei werden bestimmte Bedingungen definiert – die *Constraints* – die an die Positionen der Welt gestellt werden, basierend auf den Informationen, die von

der Welt bekannt sind (z.B. die Anordnungen auf dem Spielfeld). Ein solcher Constraint wäre z.B. die Festlegung des Abstands zu einer Landmark oder der Winkel, unter dem eine Landmark gesehen werden muss, wenn der Roboter an einem bestimmten Punkt steht. All diese Constraints bilden zusammen ein *Constraint-Netz*. Bei der Lokalisierung wird nun eine Position in der Welt gesucht, die möglichst alle Constraints erfüllt. Mittels Constraint-Propagierung können gültige Positionen gefunden werden[10, 5].

4 Selbstlokalisierung der FUmanoids

Das Lokalisierungsmodul der FUmanoids basiert auf einem vereinfachten *Augmented MCL–Algorithmus* (vgl. 2.2.1). Die verwendeten Partikel sind dreidimensional und repräsentieren Ort (x, y) und Orientierung θ des Roboters auf dem bekannten Spielfeld.

Der Partikelfilter selbst verwendet allerdings die Orientierung der Partikel nicht, da diese aus den Daten des Visionsystems über die Feldlinien bestimmt werden kann. Wie diese Orientierungsberechnung erfolgt, wird in Abschnitt 4.1 erklärt. Dadurch kann die Anzahl der Partikel drastisch gesenkt werden.

Beim Resampling wird der erklärte Low Variance Sampler verwendet. Anstatt aber immer nur Kopien der guten Partikel zu erzeugen, werden die Partikel entsprechend der Normalverteilung um die guten Partikel gestreut. Die Standardabweichung der Streuung variiert dabei abhängig davon, ob der Roboter steht oder läuft.

Als Roboterposition wird nicht nur der am besten bewertete Partikel verwendet, sondern die Gesamtheit der Partikel betrachtet, um daraus einen guten Positionskandidaten zu erhalten. Zur Bestimmung der endgültigen Roboterposition wird dieser Positionskandidat anschließend mit Hilfe der zuletzt angenommenen Roboterposition nachgeglättet (vgl. 4.4).

4.1 Bestimmung der Roboterorientierung

Da alle Feldlinien (mit Ausnahme des Mittelkreises) waagerecht oder senkrecht verlaufen, kann aus den Richtungen, in denen die Feldlinien gesehen werden, die Orientierung des Roboters im Intervall von $[0°; 90°]$ direkt bestimmt werden. Eine 360° Auflösung ist ohne Vorwissen nicht möglich.

Unter Verwendung der vorangehenden Orientierung des Roboters haben Ribeiro et al.[13] eine Lösung dazu vorgestellt. Ihre Idee ist die Punktwolke der gesehenen Feldlinien ausgehend von der bisherigen Orientierung stückweise zu rotieren und in jedem Rotations-

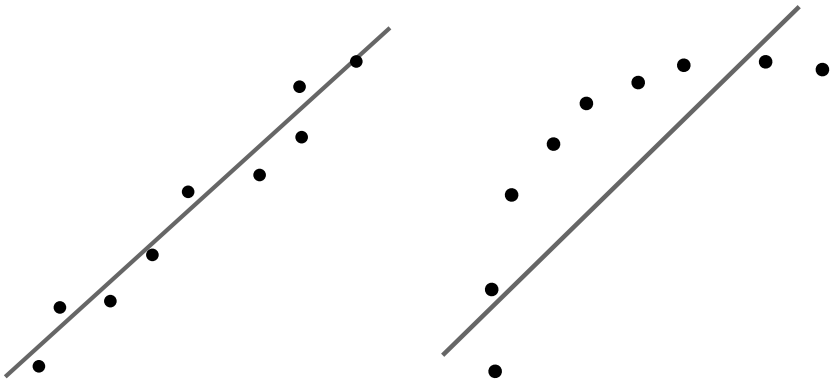
schritt Histogramme über die Verteilung der Punkte in horizontaler und vertikaler Richtung zu erstellen. Die Roboterorientierung ergibt sich dann als der Winkelwert, unter dem die Summe der Maximalwerte beider Histogramme am Höchsten ist.

Im Folgenden wird nun ein Ansatz vorgestellt, der nur ein Histogramm benötigt. Folgende Schritte sind dazu notwendig:

1. Trennung von gerade verlaufenden Feldlinien und solchen Feldliniensegmenten, die zum Mittelkreis gehören
2. Erstellung eines Orientierungshistogramms aller geraden Feldlinien

4.1.1 Trennung von geraden Feldlinien und Kreis-Feldlinien

Die vom Visionsystem gelieferten Daten über die Feldlinien liegen als geordnete Punktfolgen vor, sodass eine Punktfolge den Verlauf einer gesehenen Linie beschreibt. Eine gerade Linie hat die Eigenschaft, dass alle Punkte der Folge sehr nah bei (oder im Optimalfall direkt auf) der Ausgleichsgeraden liegen, die durch die Punktfolge verläuft. Bei einer gekrümmten Kreislinie ist dies nicht der Fall, hier kann nur schwer eine gute Ausgleichsgerade gefunden werden, bzw. der Fehler zwischen Punktfolge und der Ausgleichsgeraden ist sehr groß (siehe Abbildung 4.1).



(a) Gerade Feldlinie (b) Kreis-Feldlinie

Abbildung 4.1: Das Finden einer Ausgleichsgeraden (grau dargestellt) gelingt im Fall von einer geraden Feldlinie, aber scheitert bei Liniensegmenten des Kreises.

Diese Eigenschaft kann sich nun zur Trennung von Kreissegmenten und geraden Feldlinien zunutze gemacht werden, indem mit dem RANSAC-Algorithmus[1] (siehe Algorithmus 2)

[1] **Ran**dom **Sa**mple **C**onsensus

versucht wird eine Ausgleichsgerade zu finden, die einen möglichst kleinen Fehler zur Punktmenge der Linie hat.

Algorithmus 2 RANSAC-Algorithmus

```
1: err = ∞
2: model = (∅, ∅)
3: for i = 1 → maxIterations do
4:    (p1, p2) ← zufällige Punkte der Punktfolge als temporäres Modell
5:    tempErr ← Abstand der Punktfolge zu temporärem Modell (p1, p2)
6:    if tempErr < err then
7:       model ← (p1, p2)
8:       err ← tempErr
9:    end if
10: end for
```

Eine erste Version würde dann so funktionieren, dass ein Schwellenwert für die erlaubte Abweichung von der Ausgleichsgeraden festgelegt wird. Ist diese Abweichung größer als der Schwellenwert, dann kann es sich nicht um eine gerade Feldlinie handeln. Daher kommt diese Punktfolge dann in die nähere Auswahl der Kreisliniensegmente.

Dieser Ansatz allein hat sich allerdings als nicht ausreichend herausgestellt, da kein Schwellenwert für den Fehler gefunden werden konnte, der in allen Fällen zur Unterscheidung genügt. Es kam vor, dass eigentlich gerade Linien durch Ausreißer einen größeren Fehlerwert erhielten und somit als Kreiskante einsortiert wurden oder umgekehrt. Ein Grund für dieses Verhalten ist die Fischaugenlinse, die bei den FUmanoids zum Einsatz kommt. Durch die Starke Verzerrung gerade an den Bildrändern sorgt sie für große Fehler in der Entzerrung und bei der Projektion in Weltkoordinaten.

Daher werden die Punktfolgen auf ein weiteres Kriterium hin analysiert: Die Kreislinien eine Krümmung besitzen. Durch Berechnung des Winkels, der zwischen den Strecken zwischen dem ersten und mittleren Punkt und dem mittleren und letzten Punkt der Linie entsteht, kann die Krümmung approximiert werden. Ein weiterer Schwellenwert, der den maximalen Winkel festlegt, unter dem eine Linie noch als gerade gelten soll, löst das oben genannte Problem weitgehend und die Kreislinien können von geraden Feldlinien unterschieden werden.

Aus den Kreislinien kann zusätzlich noch der Mittelpunkt des Kreises bestimmt werden.

4.1.2 Orientierungshistogramm

Sind die geraden Feldlinien bestimmt, kann nun ein Histogramm der Richtungen erstellt werden. Das Histogramm besteht aus 90 Bins, entsprechend der Winkel von 0° bis 90°.

Mittels atan2(y, x) wird für alle Punkte jeder geraden Linie der Winkel berechnet und ins Intervall von 0° bis 90° abgebildet. Die Abbildung erfolgt durch die Modulo-Operation mit 90. Das Ergebnis wird dann dem Histogramm hinzugefügt.

Ist das Histogramm gefüllt, so wird die Orientierung bestimmt. Es wird dazu die Hauptrichtung der Feldlinien gesucht. Die Idee ist dabei folgende: Wenn der Roboter sich um x Grad dreht, haben sich die Feldlinien von ihm aus gesehen um $-x$ Grad gedreht. Wenn also die Hauptrichtung der Feldlinien z.B. 10° beträgt, kann daraus geschlussfolgert werden, dass der Roboter sich um $-10°$ gedreht hat. Die Hauptrichtung bestimmt sich folgendermaßen: Es wird zunächst der Winkel i_{max}, der am häufigsten auftrat, ausgewählt. Um Rundungsfehlern und Messungenauigkeiten vorzubeugen, wird anschließend die Nachbarschaft um diesen Winkel betrachtet. Dazu wird das gewichtete arithmetische Mittel der Nachbarschaft berechnet. i_{avg} entspricht dann dem gesuchten Winkel, w_i ist der Wert an Index i des Histogramms.

$$i_{avg} = \frac{\sum_{i=i_{max}-m}^{i_{max}+m} i w_i}{\sum_{i=i_{max}-m}^{i_{max}+m} w_i}$$

Allerdings sind die negierte Gradzahl addiert mit Vielfachen von 90° im Bereich von 0° bis 360° zulässige Orientierungen des Roboters.

$$\theta = a \cdot 90° - i_{avg} \quad \text{mit } a \in (0, 4]$$

Ein Beispiel zur Bestimmung der Orientierung ist in Abbildung 4.2 dargestellt.

Ist die Orientierung nun auf die vier Möglichkeiten eingeschränkt, können die Partikel entsprechend aktualisiert werden. Da der Roboter sich zwischen zwei Lokalisierungszyklen nicht stark drehen kann (außer bei einem Sturz), wird als neue Orientierung der Partikel nur der Wert genommen, der am nächsten zur vorangehenden Orientierung liegt. Folglich ist also die Dimension des Partikelfilters von drei auf zwei gesenkt worden und nur noch die absolute Position bleibt unbekannt.

Abbildung 4.2: Gerade Feldlinien relativ zum Roboter gesehen sind in Weiß gezeichnet. In Schwarz ist das Orientierungshistogramm dargestellt, der rote Block gibt die gemittelte Orientierung in der Nachbarschaft von $\pm 10°$ an. Hier ergibt sich z.B. ein Wert von $15°$, d.h. der Roboter ist um $-15°$ gedreht (oder $90 - 15 = 75°$, $180 - 15 = 165°$ usw.). Die gelben Feldlinien spiegeln die um $75°$ rotierten Feldlinien wieder, die nun nur senkrecht und waagerecht verlaufen.

4.2 Generierung von Positionshypothesen

Wie bereits beschrieben, werden beim Augmented MCL-Ansatz zufällige bzw. aus den Sensordaten generierte Partikel eingestreut. Die Anzahl der neuen Partikel ist dabei abhängig von dem kurzfristigen bzw. langfristigen Belief aller Partikel. Bei den FUmanoids kommt eine vereinfachte Version zum Tragen, die nur auf einem Schwellenwert für den durchschnittlichen Belief der Partikel beruht. Zusätzlich ist die Anzahl der neuen Partikel nicht beschränkt. Diese werden nicht zufällig bestimmt, sondern beruhen auf berechneten Positionshypothesen der Sensordaten.

Aus dem Visionsystem werden die relativ zum Roboter gesehenen Positionen der Tore, Säulen und Feldlinien-Features gewonnen. Die gesehenen Tore und Säulen sind eindeutig mit einer Position auf dem Feld assoziiert, sodass jeweils vier mögliche Positionen eingestreut werden können. Es gibt vier Möglichkeiten, da die Orientierung bis auf vier

Varianten festgelegt ist (vgl. Abschnitt 4.1). Zudem kann sich beim Einfügen neuer Hypothesen nicht auf die vorangehende Orientierung verlassen werden, da diese falsch berechnet worden sein und somit selbst zum schwachen Belief geführt haben könnte, der die Generierung neuer Positionen notwendig macht.

Die Feldlinien-Features treten nicht nur einmalig auf dem Feld auf; ein T-Stück ist so z.B. sechs mal auf dem Spielfeld zu finden. Daher müssen für jede dieser möglichen Positionen vier neue Hypothesen generiert werden.

Alle so generierten Hypothesen werden dann direkt der Partikelmenge hinzugefügt. Um die Partikelanzahl dennoch konstant zu halten, werden am Ende des Zyklus die überschüssigen Partikel entfernt. Dabei werden zuerst die Partikel aussortiert, deren Belief am niedrigsten ist.

4.3 Partikelbewertung durch Sensordaten

Die Bewertung der Partikel bildet den Kernpunkt der Lokalisierung. In diesem Schritt werden die Partikel entsprechend der Sensordaten auf Plausibilität überprüft und mit einem Belief versehen.

Zur Bewertung werden folgende Sensordaten des Visionsystem verwendet:

- Gerade Feldlinien und Feldliniensegmente des Kreises (e)
- Aus den Kreisliniensegmenten berechneter Mittelpunkt des Kreises (c)
- T- und X-Feldlinien-Features (flf)
- Landmarks wie Tore und Säulen (l)

Für jede genannte Kategorie x wird ein spezieller Belief $bel(x, p)$ für Partikel p berechnet. Diese Werte werden dann durch α_x unterschiedlich gewichtet, wobei $\sum_x a_x = 1$, und zu dem Gesamtbelief $Bel(p)$ verrechnet. Ein weiteres Kriterium ist noch die Entfernung des Partikels zur vorangehenden Position des Roboters (lp), die natürlich besonders klein sein soll, da sich der Roboter nicht schnell bewegt.

$$Bel(p) = \alpha_{lp} bel(lp, p) + \alpha_e bel(e, p) + \alpha_{flf} bel(flf, p) + \alpha_l bel(l, p) + \alpha_c bel(c, p)$$

Sollten nicht aus allen Kategorien Daten zur Verfügung stehen, so werden die entsprechenden Summanden weggelassen. Bei neu eingestreuten Positionshypothesen ist zusätzlich

$\alpha_{lp} bel(lp, p)$ wegzulassen, da man ansonsten Positionshypothesen von Anfang an schlecht bewertet und so im Falle des Kidnapped Robot Probleme bekommt.

Die speziellen Beliefs berechnen sich folgendermaßen:

$$bel(x, p) = \exp\left(-\frac{1}{n}\sum_{i=1}^{n}\frac{\left(\frac{d_i}{e+\alpha_d d_i}\right)^2}{2}\right)$$

d_i ist der Abstand von dem gesehenen Objekt aus Sicht des Partikels zum nächstmöglichen Pendant auf dem Spielfeldmodell. e gibt die Fehlertoleranz in der Distanzmessung an. Zusätzlich wird durch α_d die Fehlertoleranz für weit entfernte Objekte vergrößert, um Messungenauigkeiten vorzubeugen. n gibt dabei die Anzahl der vorhandenen Sensordaten an.

Bei Feldlinien werden Kreisliniensegmente nur mit Kreislinien des Modells verglichen, horizontale Linien nur mit horizontalen und vertikale nur mit vertikalen. Die Bestimmung des Abstands erfolgt hier über eine vorberechnete Lookup-Tabelle.

Bei den Landmarks wird nicht der Abstand in Zentimetern, sondern der Winkelabstand verwendet, da z.B. die Tore auch noch aus weiter Entfernung gesehen werden und der Fehler der Distanzmessung dann sehr groß wird. Dies liegt vor allem an der verwendeten Fischaugenlinse. Der Winkel ist hingegen nicht so fehlerbehaftet und bleibt auch bei größerer Entfernung zuverlässig.

4.4 Bestimmung der Roboterposition

Ist für alle Partikel der Belief bestimmt, kann zum letzten Schritt übergegangen werden – der Extraktion der Roboterposition aus der Partikelmenge. Dies geschieht in zwei Schritten: Zunächst wird aus der Partikelmenge ein neuer *Positionskandidat* berechnet. Anschließend wird dieser Positionskandidat mit der vorangehenden Position verrechnet, um große Sprünge zu vermeiden und den Positionsverlauf zu glätten.

Zur Bestimmung des Positionskandidaten werden zwei verschiedene Möglichkeiten vorgestellt. Eine Variante ist den am besten bewerteten Partikel als Positionskandidat auszuwählen, die andere Variante analysiert die Verteilung der Partikel unter Zuhilfenahme eines Gitters und wird als *Binning* bezeichnet.

4.4.1 Bester Partikel

Die einfachste Methode einen Positionskandidaten zu bestimmen besteht darin den am besten bewerteten Partikel der Partikelmenge auszuwählen. Dieser Partikel ist sehr effizient zu bestimmen und die Methode hat keinen Overhead. Ein Nachteil ist allerdings, dass nur die aktuellen Sensordaten berücksichtigt werden, da nur diese Einfluss auf den Belief der Partikel haben. Durch eine falsche Messung geht demnach sofort die Lokalisierung kaputt. Zudem kann es in Fällen, in denen der Partikelfilter multimodal fungiert, also zwei oder mehr Hypothesen gleichzeitig verfolgt werden, vorkommen, dass der Positionskandidat zwischen den verschiedenen Hypothesen hin- und herspringt.

4.4.2 Binning

Das Binning ist eine weit verbreitete Methode zur Bestimmung eines Positionskandidaten[9, 15]. Beim Binning wird der Partikelraum mit einem Gitter von $n \times m$ Bins überzogen. Es wird dann der Bin bestimmt, welcher am meisten Partikel enthält. Von den Partikeln innerhalb dieses Bins wird dann der Mittelwert bestimmt und als Positionskandidat verwendet.

Dadurch, dass die Verteilung der Partikel und nicht die Bewertung betrachtet wird, haben falsche Sensordaten keinen so hohen Einfluss wie bei der Verwendung des besten Partikels, denn in der Verteilung spiegeln sich die vorangehenden Sensormessungen wieder: Nur dort, wo der Belief in die Messungen hoch war, werden mehr Partikel gestreut. Zudem sind Oszillationen zwischen zwei oder mehr Anhäufungen unwahrscheinlicher.

Es kann jedoch passieren, dass zwischen zwei Bins hin und her gesprungen wird, wenn sich die Anhäufung von Partikeln über mehrere Bins erstreckt. In dem Fall ist es sinnvoll nicht den Bin mit den meisten Partikeln zu betrachten, sondern eine 2×2-Nachbarschaft von Bins. Die Größe der Bins muss zudem sorgsam gewählt werden, denn bei zu kleinen Größen tritt das Oszillationsproblem wieder auf und bei zu großen können mehrere Anhäufungen in einem Bin landen, sodass die multimodale Behandlung schwierig wird.

In Abbildung 4.3 ist das Binning visualisiert.

4.4.3 Glättung des Positionskandidaten

Ist nun der Positionskandidat mittels einer der vorgestellten Methoden bestimmt, so kann die endgültige Position berechnet werden. Es wird der Positionskandidat p_{cand} mit der

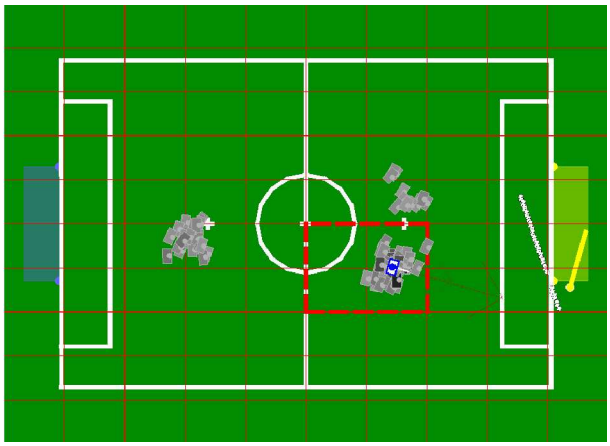


Abbildung 4.3: Die größte Anhäufung innerhalb eines 2×2 großen Fensters wird im rot eingerahmten Bereich gefunden. Der blau gezeichnete Partikel entspricht dem gemittelten Wert aller in diesem Bereich befindlichen Partikel, dies ist der Positionskandidat. (Quelle: [9])

vorangehenden Position p_{old} in Beziehung gesetzt. Die Entfernung der neuen Position von der alten wird nach oben hin begrenzt (z.B. auf $10cm$), abhängig von der Höchstgeschwindigkeit des Roboters, und gleichzeitig mittels des Beliefs von dem Positionskandidaten skaliert:

$$d = \begin{pmatrix} x_{cand} - x_{old} \\ y_{cand} - y_{old} \end{pmatrix}$$

$$d_s = \begin{cases} d & |d| \leq 10cm \\ \frac{10cm \cdot d}{|d|} & \text{sonst} \end{cases}$$

$$p_{new} = p_{old} + Bel(p_{cond}) \cdot d_s$$

Die Berücksichtigung des Beliefs sorgt dafür, dass der Roboter bei plötzlich auftretenden schlechten Bewertungen (z.B. durch fehlerhafte Sensordaten) auf seiner bisherigen Position verharrt und sich nur träge der neuen Position angenähert. Dies funktioniert, da sich der Roboter nicht allzu schnell bewegt und zusätzlich sich kurzfristig auf das Bewegungsmodell verlassen kann. Im Falle eines Kidnapped Robot reagiert die Glättung auch recht träge, sodass es ein paar Sekunden dauern kann, bis der Roboter sich vollkommen neu lokalisiert hat.

5 Ergebnisse und Ausblick

5.1 Auswertung

5.1.1 Auswertung im Simulator

Mit Hilfe der vorhandenen Simulatorsoftware kann die Lokalisierung unkompliziert getestet werden. Der Simulator ersetzt dabei die elektronischen Komponenten wie Sensoren und Motoren durch Software-Emulationen. Die erzeugten Sensordaten (z.B. das Kamerabild) werden dabei transparent an das FUmanoid-Programm übermittelt, damit das Programm mit diesen Daten rechnen kann. Der Auswertung der Sensoren folgt eine entsprechende Aktion, wie z.B. das Abspielen einer speziellen Bewegung oder das Laufen zu einem bestimmten Punkt. Diese Aktion wird dem Simulator bekannt gegeben und dort emuliert.

Da der Simulator die „Wahrheit“ kennt, also weiß, wo sich der Roboter tatsächlich befindet, können diese *Ground Truth Positionen* zum Vergleich mit den von der Lokalisierung bestimmten Positionen verglichen werden.

Zur Auswertung wurde ein vorgegebenes Muster mehrfach auf dem Spielfeld abgefahren. Es wurden jeweils fünf Testläufe mit unterschiedlichen Einstellungen gefahren. Zuerst wurde die Positionsbestimmung über die Benutzung des besten Partikels mit anschließender Glättung durchgeführt, anschließend ohne Glättung. Dann wurde die Positionsbestimmung über Binning mit und ohne anschließender Positionsglättung ausgewertet. Zum Vergleich wurde pro Testlauf die Standardabweichung σ von berechneter Position und Ground Truth Position ermittelt. Die Ergebnisse sind in Tabelle 5.1 aufgeführt. Eine graphische Darstellung der jeweils besten Läufe ist in Abbildung 5.1 zu finden.

In Bezug auf den Simulator kann eindeutig gefolgert werden, dass die Variante mit ausschließlicher Benutzung des besten Partikels am sinnvollsten erscheint. Dies muss aber dahingehend relativiert werden, dass die Bewegungsgeschwindigkeit im Simulator höher als in Realität ist (ca. $45cm/s$ bis $55cm/s$), sodass die zu niedrige Bewegungsgrenze im

Optionen	$\min \sigma$	$\max \sigma$	$\varnothing \sigma$
Bester Partikel und Glättung	$28,96cm$	$48,05cm$	$27,25cm$
Bester Partikel ohne Glättung	$13,13cm$	$15,12cm$	$14,16cm$
Binning und Glättung	$101,05cm$	$160,80cm$	$133,33cm$
Binning ohne Glättung	$55,25cm$	$64,30cm$	$57,82cm$

Tabelle 5.1: Auswertung mit Hilfe des Simulators. Standardabweichung σ wurde aus jeweils fünf Testläufen berechnet.

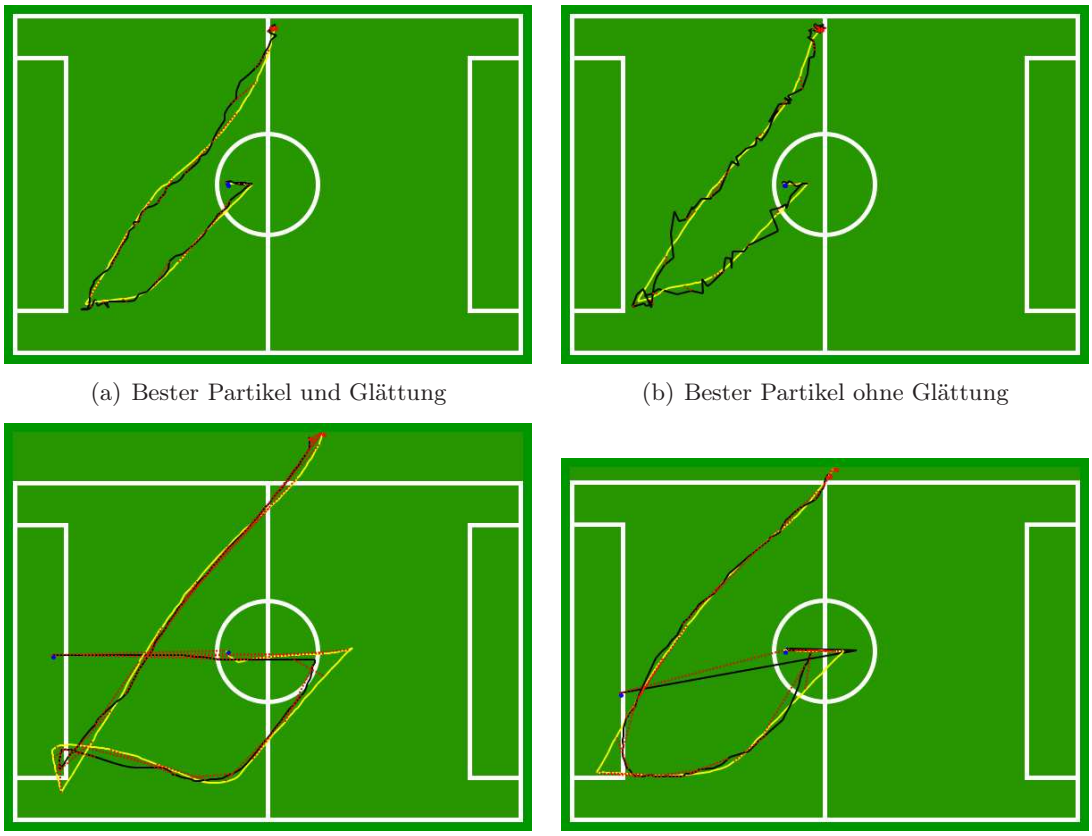
Glättungsschritt zu einer gravierenden Verzögerung der Positionsänderung führt. Ohne diese Trägheit ist das geglättete Signal näher an der Ground Truth Position als ohne Glättung.

Bei der Verwendung des Binning ist zu beobachten, dass die anfänglichen Positionen sehr schlecht sind, was zu den hohen Standardabweichungen führt. Dies liegt daran, dass die gleichverteilte initiale Partikelverteilung zu keiner Clusterbildung in den ersten Durchläufen führt. Ist aber erstmal eine gute Position gefunden, so sind die Ergebnisse recht genau. Da das Binning durch die Berechnung der Durchschnittsposition innerhalb des meistbesetzten Bins bereits eine Glättung durchführt, ist hier die zusätzliche Positionsglättung nicht notwendig. Die schlechten Ergebnisse in diesem Fall sind auch wieder auf die zu niedrigen Bewegungsgrenzen zurückzuführen.

5.1.2 Auswertung auf dem Roboter

Zur Analyse der Lokalisierung auf den Robotern wurde die Microsoft Kinect verwendet. Die Kinect ist ein Sensor, der neben einem Kamerabild auch ein Tiefenbild liefert, welches über die Abtastung der Umwelt via Infrarot erstellt wird. Der Sensor wurde dabei ursprünglich dazu entwickelt Videospiele nicht mit Hilfe der altbekannten Gamepads zu steuern, sondern über die Bewegungen des Spielers.

Durch Auswertung des Tiefenbilds kann sehr einfach ein Roboter auf dem Spielfeld getrackt werden. Dazu muss lediglich die Spielfeldebene im Tiefenbild definiert werden. Die so gewonnenen Ground-Truth-Daten können dann, wie auch schon bei der Auswertung mit Hilfe des Simulators, mit den Positionen der Lokalisierung verglichen werden. Es ist allerdings zu beachten, dass das Rauschen der Tiefeninformationen abhängig von der Entfernung ist. Je weiter ein Objekt entfernt ist, umso verrauschter sind die Entfernungsdaten. Daher eignet sich die Kinect nur für Messungen im Bereich von ca. $0,8m$ bis $3,5m$.



(a) Bester Partikel und Glättung

(b) Bester Partikel ohne Glättung

(c) Binning und Glättung

(d) Binning ohne Glättung

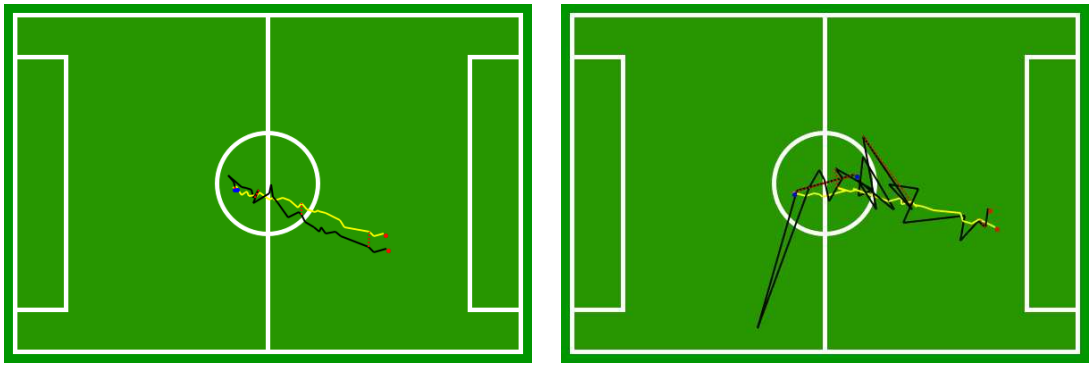
Abbildung 5.1: Visualisierung der vom Simulator gelieferten Positionen (gelb) und der durch den Roboter berechneten Positionen (schwarz). Die roten gestrichelten Linien zeigen die Zusammengehörigkeit von Simulatorposition zu Roboterposition. Roter bzw. blauer Punkt visualisieren Start- bzw. Endpunkt.

Die Auswertung erfolgte mit dem Roboter ähnlich wie mit dem Simulator. Es wurde wiederholt eine vorgegebenen Position angelaufen, dabei die verschiedenen Kombinationsmöglichkeiten analysiert. Die Ergebnisse der Tests sind in Tabelle 5.2 aufgeführt. Eine graphische Darstellung einiger Ergebnisse ist in Abbildung 5.2 zu sehen.

Die Tests zeigen deutlich, dass in Realität die Fehler in den Sensordaten und somit auch in der Lokalisierung wesentlich größer sind als im Simulator. Dies ist an den großen Sprüngen bei der Verwendung des reinen besten Partikels als Position (vgl. Abbildung 5.2(b)) zu erkennen. Daher bietet es sich auf dem Roboter an, die Glättung der Positionen zu aktivieren. Daraus resultieren wesentlich glatter verlaufende Positionspfade (vgl.

Optionen	$\min \sigma$	$\max \sigma$	$\varnothing \sigma$
Bester Partikel und Glättung	$14,42cm$	$28,10cm$	$19,42cm$
Bester Partikel ohne Glättung	$33,87cm$	$61,46cm$	$47,47cm$

Tabelle 5.2: Auswertung auf den Robotern mit Hilfe der Microsoft Kinect. Standardabweichung σ wurde aus jeweils drei Testläufen berechnet.



(a) Bester Partikel und Glättung

(b) Bester Partikel ohne Glättung

Abbildung 5.2: Visualisierung der durch die Kinect gelieferten Positionen (gelb) und der durch den Roboter berechneten Positionen (schwarz). Die roten gestrichelten Linien zeigen die Zusammengehörigkeit von Ground-Truth-Position zu Roboterposition. Roter bzw. blauer Punkt visualisieren Start- bzw. Endpunkt.

Abbildung 5.2(a)). Auch die gemessene Standardabweichung konnte hierdurch reduziert werden, da Falschmessungen weniger Auswirkungen auf die Position haben.

5.2 Fazit und Ausblick

Die vorliegende Arbeit beschreibt und diskutiert das Lokalisierungsmodul der FUmanoids. Es wird ein Augmented Monte-Carlo-Lokalisierungsansatz implementiert, bei dem die Dimension der Partikel auf den Ort (x, y) beschränkt wird, da es möglich ist die Orientierung des Roboters direkt zu bestimmen. In der Implementierung werden der Partikelmenge neue Partikel hinzugefügt, wenn der Durchschnittsbelief unter einen gegebenen Schwellenwert fällt. Diese neuen Partikel sind Positionshypothesen, die aus den Sensordaten generiert werden. Zur Bestimmung der endgültigen Position des Roboters wird zunächst ein Positionskandidat aus der Partikelmenge bestimmt; dazu wird das Binning und die Verwendung des besten Partikels umgesetzt. Anschließend kann dieser

Positionskandidat noch unter Zuhilfenahme der vorangehenden Position geglättet werden.

Die experimentelle Auswertung im Simulator und auf den Robotern mit entsprechenden Ground-Truth-Daten hat gezeigt, dass die Variante unter Verwendung des besten Partikels als Positionskandidat mit anschließender Glättung die besten Ergebnisse liefert. Dieser Ansatz ist auch recht präzise mit ca. 20 Zentimetern Standardabweichung. Die beschriebene Lokalisierung kam mit den genannten Einstellungen im RoboCup 2011 in Istanbul zum Einsatz, bei dem das Team den 4. Platz errang.

Zur weiteren Verbesserung der Ergebnisse bietet sich in Zukunft an die verwendeten Parameter wie Schwellenwerte oder Gewichtungsfaktoren automatisch lernen zu lassen. Mit der Kinect und dem Simulator existiert dazu auch schon eine passende Infrastruktur, um Feedback für den Lernprozess zu erhalten. In Bezug auf die Rechenzeit könnte es interessant sein Kalman-Filter-basierte Lokalisierungsansätze zu testen, da diese zumeist schneller berechenbar sind.

Hohe Messfehler liegen häufig an der starken Verzerrung der verwendeten Fischaugenlinse. Dies fällt besonders im Bereich der Distanzmessung auf. Durch Austausch der Linse könnten die Sensordaten zuverlässiger werden, was einen direkten Einfluss auf die Qualität der Lokalisierung hätte.

Literaturverzeichnis

[1] Behnke, Sven ; Missura, Marcell: *NimbRo TeenSize 2011 Team Description*. 2011

[2] Burchardt, Armin ; Laue, Tim ; Röfer, Thomas: Optimizing Particle Filter Parameters for Self-localization. Version: 2011. http://dx.doi.org/10.1007/978-3-642-20217-9_13. In: Solar, Javier Ruiz-del (Hrsg.) ; Chown, Eric (Hrsg.) ; Plöger, Paul (Hrsg.): *RoboCup 2010: Robot Soccer World Cup XIV* Bd. 6556. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–20216–2, 145-156

[3] Fischer, Bennet ; Heinrich, Steffen ; Hohl, Gretta ; Lange, Felix ; Langner, Tobias ; Mielke, Sebastian ; Moballegh, Hamid R. ; Otte, Stefan ; Rojas, Raúl ; Schmude, Naja von ; Seifert, Daniel ; Steig, Daniel ; Weissgerber, Thomas: *FUmanoid Team Description Paper 2010*. 2010

[4] Friedmann, M. ; Kuhn, J. ; Kohlbrecher, S. ; Petersen, K. ; Scholz, D. ; Thomas, D. ; Wojtusch, J. ; Stryk, O. von: *Darmstadt Dribblers - Team Description for Humanoid KidSize League of RoboCup 2011*. 2011

[5] Göhring, Daniel ; Mellmann, Heinrich ; Burkhard, Hans-Dieter: Constraint Based World Modeling in Mobile Robotics. In: *Proc. IEEE International Conference on Robotics and Automation ICRA 2009*, 2009, S. 2538–2543

[6] Hayashibara, Yasuo ; Minakata, Hideaki ; Irie, Kiyoshi ; Ichizawa, Katsuhiro ; Machi, Kousuke ; Takamatsu, Kazushiro ; Nohira, Kosuke ; Tsuchihashi, Issei ; Yamada, Yuka ; Akitani, Toshiyuki ; Miki, Shigechika ; Nishizaki, Yoshitaka ; Kanemasu, Kenji ; Sakamoto, Hajime: *CIT Brains (Kid Size League)*. 2011

[7] Jochmann, Gregor ; Kerner, Sören ; Tasse, Stefan ; Urbann, Oliver: Efficient Multi-Hypotheses Unscented Kalman Filtering for Robust Localization. In: Röfer, Thomas (Hrsg.) ; Mayer, Norbert M. (Hrsg.) ; Savage, Jesus (Hrsg.) ; Saranli, Uluç (Hrsg.): *RoboCup 2011: Robot Soccer World Cup XV*. Springer Berlin / Heidelberg, 2012 (Lecture Notes in Computer Science)

[8] Langner, Tobias: *Selbstlokalisierung für humanoide Fußballroboter mittels Mono- und Stereovision*, Freie Universität Berlin, Fachbereich Mathematik und Informatik, Diplomarbeit, September 2009

[9] Laue, Tim ; Röfer, Thomas: Pose Extraction from Sample Sets in Robot Self-Localization - A Comparison and a Novel Approach. In: Petrović, Ivan (Hrsg.) ; Lilienthal, Achim J. (Hrsg.): *Proceedings of the 4th European Conference on Mobile Robots - ECMR'09*. Mlini/Dubrovnik, Croatia, 2009, S. 283–288

[10] Mellmann, Heinrich: *Ein anderes Modell der Welt. Alternative Methoden zur Lokalisierung mobiler Roboter.*, Humboldt-Universität zu Berlin, Institut für Informatik, Diplomarbeit, April 2010

[11] Özkucur, N. ; Akin, H.: Localization with Non-unique Landmark Observations. Version: 2011. http://dx.doi.org/10.1007/978-3-642-20217-9_7. In: Solar, Javier Ruiz-del (Hrsg.) ; Chown, Eric (Hrsg.) ; Plöger, Paul (Hrsg.): *RoboCup 2010: Robot Soccer World Cup XIV* Bd. 6556. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–20216–2, 72-81

[12] Quinlan, Michael ; Middleton, Richard: Multiple Model Kalman Filters: A Localization Technique for RoboCup Soccer. Version: 2010. http://dx.doi.org/10.1007/978-3-642-11876-0_24. In: Baltes, Jacky (Hrsg.) ; Lagoudakis, Michail (Hrsg.) ; Naruse, Tadashi (Hrsg.) ; Ghidary, Saeed (Hrsg.): *RoboCup 2009: Robot Soccer World Cup XIII* Bd. 5949. Springer Berlin / Heidelberg, 2010. – ISBN 978–3–642–11875–3, 276-287

[13] Ribeiro, Fernando ; Lopes, Gil ; Pereira, Bruno ; Silva, João ; Ribeiro, Paulo ; Costa, João ; Silva, Sérgio ; Rodrigues, João ; Trigueiros, Paulo: Robot Orientation with Histograms on MSL. In: Röfer, Thomas (Hrsg.) ; Mayer, Norbert M. (Hrsg.) ; Savage, Jesus (Hrsg.) ; Saranli, Uluç (Hrsg.): *RoboCup 2011: Robot Soccer World Cup XV*. Springer Berlin / Heidelberg, 2012 (Lecture Notes in Computer Science)

[14] RoboCup Soccer Humanoid League Technical Committee: *RoboCup Soccer Humanoid League Rules and Setup 2011*. Mai 2011

[15] Röfer, Thomas ; Jüngel, Matthias: Fast and Robust Edge-Based Localization in the Sony Four-Legged Robot League. Version: 2004. http://dx.doi.org/10.1007/978-3-540-25940-4_23. In: Polani, Daniel (Hrsg.) ; Browning, Brett (Hrsg.) ; Bonarini, Andrea (Hrsg.) ; Yoshida, Kazuo

(Hrsg.): *RoboCup 2003: Robot Soccer World Cup VII* Bd. 3020. Springer Berlin / Heidelberg, 2004. – ISBN 978–3–540–22443–3, 262-273

[16] Röfer, Thomas ; Laue, Tim ; Müller, Judith ; Burchardt, Armin ; Damrose, Erik ; Fabisch, Alexander ; Feldpausch, Fynn ; Gillmann, Katharina ; Graf, Colin ; Haas, Thijs J. ; Härtl, Alexander ; Honsel, Daniel ; Kastner, Philipp ; Kastner, Tobias ; Markowsky, Benjamin ; Mester, Michael ; Peter, Jonas ; Riemann, Ole Jan L. ; Ring, Martin ; Sauerland, Wiebke ; Schreck, André ; Sieverdingbeck, Ingo ; Wenk, Felix ; Worch, Jan-Hendrik: *B-Human Team Report and Code Release 2010.* 2010. – Only available online: `http://www.b-human.de/file_download/33/bhuman10_coderelease.pdf`

[17] Schulz, Hannes ; Liu, Weichao ; Stückler, Jörg ; Behnke, Sven: Utilizing the Structure of Field Lines for Efficient Soccer Robot Localization. Version: 2011. `http://dx.doi.org/10.1007/978-3-642-20217-9_34`. In: Solar, Javier Ruiz-del (Hrsg.) ; Chown, Eric (Hrsg.) ; Plöger, Paul (Hrsg.): *RoboCup 2010: Robot Soccer World Cup XIV* Bd. 6556. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–20216–2, 397-408

[18] Thrun, Sebastian ; Burgard, Wolfram ; Fox, Dieter: *Probabilistic Robotics.* MIT Press, 2005