
**„Kamerabasierte Selbstlokalisierung
humanoider Roboter
in der RoboCup Umgebung“**

Diplomarbeit

vorgelegt von

Roman Guilboud
389 21 79

Freie Universität Berlin
Institut für Informatik



Hauptgutachter: Prof. Dr. Raúl Rojas

2 Juni 2008

Erklärung

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

Berlin, den 1. Juni 2008

Roman Guilbourd

Zusammenfassung:

Eine zuverlässige Selbstlokalisierung stellt eine der wichtigsten Voraussetzungen für die autonome Mobilität dar. Die Kenntnis der eigenen Pose gehört zu den wichtigsten Zustandsparametern eines mobilen Robotersystems, denn eine halbwegs anspruchsvolle Bewegungsplanung ist auf die Informationen über die aktuelle Position und Orientierung angewiesen. Allgemein gesehen handelt es sich bei der Selbstlokalisierung um ein Schätzproblem. In dieser Diplomarbeit werden zwei klassische stochastische Verfahren (Kalman Filter, Partikelfilter) zur Zustandsschätzung von dynamischen Systemen beschrieben und im Hinblick auf die spezifischen Anforderungen der humanoiden Liga des RoboCup-Wettbewerbs analysiert. Anschließend wird die Implementierung einer auf der Partikelfilterung basierenden Selbstlokalisierungsmethode präsentiert. Ein besonderer Augenmerk gilt dabei der Effizienz der Implementierung, wodurch das hohe Datenaufkommen einer Videokamera mit den knappen Hardware-Ressourcen eines mikrocontroller-gesteuerten Roboters vereinbart werden sollen.

Inhaltsverzeichnis

Kapitel 1: Motivation

KI-Herausforderung RoboCup

1.1 Geschichte	8
1.2 Ziele, Regeln und Vorgehensweise beim RoboCup	9
1.3 Die Humanoiden-Liga	13

Kapitel 2: Problemstellung

Team Fumanoid

2.1 Einführung	17
2.2 Hardware	18
2.3 Kamera-Modul	19
2.4 Software	21

Kapitel 3: Lösungsmöglichkeiten

Standardmethoden der Selbstlokalisierung

3.1 Einführung	25
3.2 Kalman Filter	26
3.2.1 Begriffsklärung	27
3.2.2 Herleitung der Berechnungsvorschrift	28
3.2.3 Zusammenfassung	31
3.3 Markow-Lokalisierung	33
3.3.1. Bayes Filter	33
3.3.2 Bewegungsmodell	36
3.3.3 Sensormodell	39
3.3.4 Zusammenfassung	41
3.4 Monte-Carlo-Lokalisierung (MCL).....	44
3.4.1 Ablauf	45
3.4.2 Initialisierungsphase	45
3.4.3 Prädiktion-Phase	45

3.4.4 Importance Sampling	47
3.4.5 Update-Phase	48
3.4.6 Degenerationsproblem	49
3.4.7. SIR-MCL: Zusammenfassung	50

Kapitel 4: Problemlösung

Implementierung einer Monte-Carlo-Selbstlokalisierungsmethode

4.1 Einführung	52
4.2 Programmstruktur	53
4.2.1 Particle-Stuktur	54
4.2.2 Globale Selbstlokalisierung	55
4.2.3 Lokale Selbstlokalisierung	57
4.3 Implementierung des Sensorsmodells	59
4.3.1 Überblick	59
4.3.2 Punkt 1: Feldmodell	61
4.3.3 Punkt 2: Differenzbestimmung	63
4.3.4 Punkt 3: Messfehlermodell	64
4.3.5 Zusammenfassung der Schätzung	65
4.3.6 Eigenschaften der Implementierung	66
4.3.7 Optimierung der Implementierung	68
4.4 Implementierung des Bewegungsmodells.....	70
4.5 Implementierung der Sampling-Methode.....	72

Kapitel 5: Tests

5.1 Sensormodell-Fehlervarianz	77
5.2 Bewegungsmodell-Parameter, Bewegungsmodell-Fehlervarianz	79
5.3 Sensormodell	80
5.4 Sampling-Funktion	81
5.5 Systemtests	81
5.5.1 Globale Selbstlokalisierung	81
5.5.2 Lokale Selbstlokalisierung	83
5.6 Kidnapped Robot	85

Kapitel 6: Ergebnis und Ausblick

6.1 Ergebnis	87
6.2 Ausblick	88
Literaturverzeichnis.....	92

Kapitel 1

KI-Herausforderung RoboCup

1.1 Geschichte

Die Idee, Schach spielende Automaten zu Zwecken der Bewertung des Entwicklungsfortschritts von Computertechnik und KI zu verwenden, wurde in der Mitte des 20. Jahrhunderts weit verbreitet. Man hoffte unter anderem, in einem direkten Vergleich sehen zu können, wie gut sich menschliche Intelligenz automatisieren lässt, ob Maschinen eine Herausforderung für das menschliche Denkvermögen darstellen können oder sogar in der Lage sind den Menschen zu übertreffen. Allerdings stellte man bald fest, dass angesichts der stark ausgeprägten deterministischen Natur des Schachs, das Spiel kein geeignetes Ähnlichkeitsmaß zwischen der „echten“ und simulierten menschlichen Intelligenz zur Verfügung stellte. Je näher man sich dem Ziel näherte, die besten menschlichen Schachspieler besiegen zu können, desto offensichtlicher wurden prinzipielle Unterschiede in der Herangehensweise von Mensch und Maschine an das Spiel. Während Computer von ihrer überlegenen Rechenleistung profitieren und riesige Zustandsräume durchsuchen, verwenden menschliche Schachspieler vor allen Dingen Heuristiken, mit denen sie den Suchraum so verkleinern, dass kein gewaltiger Rechenaufwand entsteht. Ermöglicht wird die Brut Force-Strategie der Rechner nicht zuletzt durch die diskrete Natur des Schachspiels und genau diese seine Eigenschaft ist das Problematische am Schachspiel als Maß der Ähnlichkeit zwischen der menschlichen und maschinellen Intelligenz.

Parallel zur technischen Entwicklung, beschäftigte man sich mit der generellen Frage, was Intelligenz eigentlich ist, und obwohl eine allgemeingültige Definition des Begriffs noch aussteht, waren sich die meisten Forscher einig, dass folgende Fähigkeiten:

- Objekterkennung (visuelle Intelligenz)
- Handeln, Motorik (manipulative Intelligenz)
- Reaktives Verhalten, Logisches Schließen (rationale Intelligenz)
- Sprechen, Sprache verstehen (Sprachliche Intelligenz)

sowie Fähigkeit zu lernen und zu planen wesentliche Merkmale von Intelligenz sind. Nun suchte man eine neue Herausforderung für die Forscher, die alle diese Fähigkeiten, unter realistischeren Bedingungen als im Falle des Schachspiels, auf die Probe stellt, etwas, das die einzelnen Errungenschaften auf dem Gebiet der

schwachen KI zu einer abgeschwächten Form der starken KI vereinen kann oder zumindest so erscheinen würde. Die Antwort wurde schließlich in dem *Roboterfußball* gefunden. Momentan existieren zwei unabhängige bedeutende internationale Wettkämpfe im Roboterfußball: *RoboCup* und Federation of International Robot-Soccer Association (FIRA). Das Konzept der fußballspielenden Roboter RoboCup wurde zum ersten Mal im Jahr 1993 vorgestellt. Zwei Jahre später gab es eine offizielle Ankündigung der ersten Weltmeisterschaftsspiele, die zwei Jahre später, im Jahr 1997, in Nagoya, Japan ausgetragen wurden. Seitdem finden die RoboCup-Weltmeisterschaften jährlich in verschiedenen Gastgeberländern der Welt statt. RoboCup bietet ein weites, fachübergreifendes Forschungsfeld in den Bereichen KI, Robotik, Elektrotechnik und anderen verwandten Disziplinen. Von besonderer Bedeutung ist dabei die Möglichkeit für die Teilnehmer ihre Erfahrungen auszutauschen und von wichtigen Forschungsergebnissen zu berichten.

Das viel zitierte (und belächelte) erklärte Maximalziel von RoboCup ist es, die Entwicklung von Künstlichen Intelligenz und Robotertechnik bis zum Jahr 2050 so weit voranzutreiben, dass ein Roboterteam den amtierenden menschlichen Weltmeister besiegen kann. Das 50-jährige Zeitintervall entspricht ungefähr der Zeitspanne von den ersten Überlegungen zum Konzept der schachspielenden Automaten bis zum Jahr 1996, in dem der damalige Schachweltmeister von einem Schachcomputer geschlagen wurde.

1.2 Ziele, Regeln und Vorgehensweise beim RoboCup

Dass es sich bei diesem neuen Forschungsfeld wieder um ein Spiel handelt, ist sicherlich kein Zufall. Spiele haben meistens klar definierten Ziele und Regeln, was ihre Implementierung in den Maschinen vereinfacht. Die Dynamik der Umgebung ist eingeschränkt, wodurch ein kompaktes Weltmodell und eine überschaubare Menge der zu berücksichtigenden (Zufalls-)Ereignissen möglich werden. Ferner bieten viele Spiele gleich mehrere Kriterien, anhand deren sich einerseits Fortschritt in der Entwicklung einzelner Fertigkeiten messen lässt, andererseits das Können der Spieler objektiv verglichen werden kann. Das wiederum ermöglicht eine natürliche Auslese der verschiedenen Lösungsansätze sowie eine quantitative Bewertung des Resultats. Immerhin wird es auch in der Tierwelt häufig durch Spielen gelernt.

Fußball erfüllt alle genannten Anforderungen und ist sogar besonders geeignet, da weltweit verbreitet. Es ist ein Mannschaftssport und so spielen hier sowohl individuelle als auch Mannschaftsqualitäten eine Rolle. Es wird in einer

realitätsnahen Umgebung gespielt, die Roboterspieler werden mit den physikalischen Gesetzen konfrontieren, müssen sich auf die Veränderungen der Umgebung einstellen und Standardaufgaben aus dem Alltagsleben der Menschen lösen. Dazu gehören in erster Linie:

- Sensorik (Objekterkennung, Hinderniserkennung)
- Motorik (Fortbewegung, Ball schießen)
- Verhalten (eine Rolle ausfüllen, Strategie verfolgen)
- Teamarbeit (Rollenteilung, Kooperation)
- Planung (Pfadplanung, Hindernisvermeidung)
- Lernen
- *Selbstlokalisierung*

Vergleicht man diese Liste mit der Liste der wichtigsten menschlichen kognitiven Leistungen, so stellt man fest, dass Roboterfußball sehr wohl ein Art stark vereinfachtes und dennoch geeignetes Modell der realen Welt ist, wobei die wichtigsten Herausforderungen für die Lebewesen in diesem Modell eine Repräsentation finden. Ein besonderes Augenmerk gilt dabei der Wiederverwendbarkeit der Technologien in der Industrie.

Der RoboCup umfasst zurzeit 5 Fußballligen mit jeweils unterschiedlichen Forschungsschwerpunkten und Regeln, sowie 2 RoboCupRescue-Ligen der Rettungsroboter und 1 RoboCup@Home Liga der Hausroboter mit Schwerpunkt Mensch-Maschine-Interaktion.

In der *Simulationsliga* der Fußballfamilie von RoboCup spielen 2 autonome Mannschaften mit jeweils 11 Spielern auf einem speziellen Server virtuell gegeneinander. Die physikalischen Gesetze, 2D/3D Umgebung, sowie Interaktion zwischen den Spielern werden simuliert. Der Schwerpunkt der Liga liegt auf der Entwicklung von Methoden der langfristigen Planung, Verfolgung einer Spielstrategie, sowie Lernverfahren. Die Simulationsliga spezialisiert sich somit auf den gleichen Aspekten des intelligenten Verhaltens, wie das Schachspiel, und während man sich in den anderen Ligen zum Teil noch mit Hardware-technischen Fragen beschäftigt, wird hier bereits an der Entwicklung des strategischen Verhaltens gearbeitet.

Four-legged robot league(Standard Platform League) ist die Liga der Fußball spielenden SONY AIBO Roboterhunde. Alle Mannschaften dieser Liga verfügen über das im Wesentlichen gleiche Hardware, deren Entwicklung damit in den Hintergrund tritt. Die wichtigsten zu bewältigenden Aufgaben hier sind Objekterkennung, Selbstlokalisierung, Planen, Strategie und Mannschaftsspiel. Im Gegensatz zur Simulationsliga sind die Roboter dieser Liga allen physikalischen

Einflüssen der Realwelt ausgesetzt, müssen mit dem Nicht-Determinismus zurechtkommen und können teilweise oder komplett ausfallen.

In der *small-size robot league* treten kleine (bis zu 18 cm Durchmesser) Fahrroboter mit fünf Spielern pro Mannschaft gegeneinander an. Die Steuerung der Roboter erfolgt zentralisiert per Funk von einem Server, der Bilder von einer über dem Spielfeld montierten Überkopfkamera bekommt, diese verarbeitet und das passende Verhalten für alle Spieler berechnet. Diese sind mit einem Kontroller ausgestattet, der die Motorik-Aufgaben übernimmt und die Server-Befehle ausführt. Somit handelt es sich dabei um ein Hybrid aus zentralisierten und verteilten Ansätzen, was auch der Forschungsschwerpunkt der Liga ist. Außerdem wird die Hardware der Roboter von den Teilnehmern selbst entwickelt. Fast alle Teams verwenden mittlerweile ein omnidirektionales Fahrsystem, das es den Robotern erlaubt gleichzeitig mit bis zu 4 m/s beliebige Kurven fahren und ihre Orientierung ändern, was für eine spektakuläre Spieldynamik sorgt.

Mid-size robot league ist die Liga der großen Fahrroboter (bis zu 50 cm Durchmesser) die vollkommen autonom agieren. Die Roboter spielen auf einem Feld, das größtmäßig einem echten „menschenauglichen“ Spielfeld am ähnlichsten ist. Die Spezialisierung der Liga gleicht der von der *Four-legged robot* Liga: Die Roboter müssen sich eigenständig auf dem Feld orientieren, Tore, Ball, Gegner und Mitspieler erkennen und ein zielgerichtetes reaktives Verhalten vorweisen können. Entwicklung und Implementierung einer zuverlässigen Hardware ist eine große Herausforderung der Liga, denn die bis zu 80 kg schweren Roboter erreichen Spitzengeschwindigkeiten von 20 km/h. Die Geschwindigkeit und Wendigkeit der Spieler entscheiden nicht zuletzt über das Erfolg der Mannschaft.



Abb.1.2.1: *Mid-size-league-Spieler mit einer omnidirektionalen Kamera*

Die *Humanoiden Liga* ist der Neuling des RoboCups. Das Team „FUMANOID“ nimmt an der Liga teil (s. den nächsten Abschnitt „Die Humanoiden Liga“).

Im Falle der letzten drei Ligen: *small-size robot league*, *Mid-size robot league* und *die humanoid league* spielt Robotik noch immer eine herausragende Rolle, ganz im Sinne der Überlegung, dass Intelligenz einen Körper und Sinne benötigt. Die ersten beiden Herausforderungen für die RoboCup-Teams betreffen Sensorik und Motorik. Während die Anforderungen an die Motorik sich grundsätzlich von Liga zu Liga unterscheiden, geht es bei der Sensorik in erster Linie um die

Erkennung von 5 Objekttypen: Ball, Tore, Gegner/Mitspieler, Feldlinien, Markierungssäulen. Im Moment spielt noch Farbkodierung der relevanten Objekte eine wichtige Rolle, es wird in geschlossenen Räumen gespielt um eine gewisse Beleuchtungshomogenität sicherzustellen, die Felddimensionen sind vergleichsweise gering. Die Spielregeln werden jedoch von Jahr zu Jahr neu festgelegt und die Spielumgebung wird kontinuierlich verändert: von den künstlichen Hilfestellungen hin zu realitätsnaher formbasierter Objekterkennung und natürlichen Lichtverhältnissen. Dabei ist das Problem der Farberkennung noch alles andere als gelöst und ist eine der häufigsten Ursachen für Fehler bei der Objekterkennung. Auch die Spielfelder werden ständig größer, was dazu führt, dass die Roboter immer längere Strecken zurücklegen müssen, was höhere Anforderungen an die Hardware stellt. Die Anzahl der Spieler auf dem Feld wird immer größer, was verstärkte Kooperation zwischen den Spielern nahe legt und komplexere Spielstrategien verlangt. Nicht selten werden auch Innovationen der Teilnehmer bei den Änderungen der Regeln berücksichtigt. Neuerungen, die nur auf den Sieg abzielen, ohne dass die einzelnen Fertigkeiten weiter entwickelt werden oder dass der Spielverlauf variiert wird, werden untersagt. Dagegen werden manche „nützlichen“ Neuerungen festgehalten und geregelt, wie das mit dem Hochschuss in der small-size league geschah.

Robuste Selbstlokalisierung ist momentan eine der größten Herausforderungen im Roboterfußball, was auf die geringe Anzahl der in Betracht gezogenen Orientierungsobjekte (Tore, Säulen, Markierungslinien) zurückzuführen ist. Die Roboter der small-size, sowie mid-size Ligen sind weniger von der Problematik betroffen, weil sie, dank der Überkopfkameras bzw. der in der mid-size Liga häufig eingesetzten omnidirektionalen Kameras, eine globale Rundumsicht auf das Feld haben. Dadurch stehen den Robotern in der Regel mehr Orientierungsobjekte zur Verfügung. Befindet sich der Roboter auf dem Feld innerhalb der Feldbegrenzungslinien, so sieht man garantiert eine bestimmte minimale Anzahl an relevanten Objekten, was bei den Robotern mit einer nach vorne gerichteten Kamera nicht der Fall ist. Die letzteren können sich nicht darauf verlassen, dass sie immer ausreichend Informationen in einem Bild zur Verfügung haben, um sich lokalisieren zu können und müssen sich aktiv um die nötigen Daten bemühen. Besonders schwierig gestaltet sich an der Stelle Verwendung der Markierungslinien, die nur in einem bestimmten Kontext als Orientierungshilfen eingesetzt werden können. Die Eckposten, die speziell für die Zwecke der Selbstlokalisierung auf dem Feld der Humanoiden, der vierbeinigen Roboter, sowie in der mid-size-Liga aufgestellt werden, werden stufenweise vom Feld genommen, sodass man schließlich in der Lage sein soll, sich nur anhand der weißen Markierungslinien und den Toren zu orientieren.

Anders als bei den schachspielenden Automaten brauchen Fußballroboter (noch) keine besonderen Fertigkeiten der langfristigen und gründlichen Planung, was, angesichts der nicht-deterministischen Umgebung, auch sehr schwierig wäre. Vielmehr liegt der Schwerpunkt derzeit noch auf der Koordination der Sensorik und Motorik. Die Fortschritte auf diesem Gebiet werden in den begleitenden Wettbewerben den s. g. „Technical Challenges“ kontrolliert.

1.3 Die Humanoiden-Liga

Die Humanoiden Liga des RoboCups ist die momentan jüngste Liga des Wettbewerbs, die zum ersten Mal im Jahr 2002 vorgestellt wurde. Die Liga wird in die Unterkategorien der Roboter in der Kindergröße(kid-size league, bis zu 60 cm hoch) und der Teenager-Größe(teen-size, bis 160 cm) unterteilt. Im Mittelpunkt der Forschung steht Entwicklung autonomer humanoider Systeme, wobei auf menschenähnliche Ausstattung und Proportionen der Roboter ein besonderer Wert gelegt wird. Sowohl die Sensorik als auch der Körperbau und die Funktionalität der einzelnen Körperteile müssen denen des menschlichen Körpers entsprechen und die dafür zuständigen

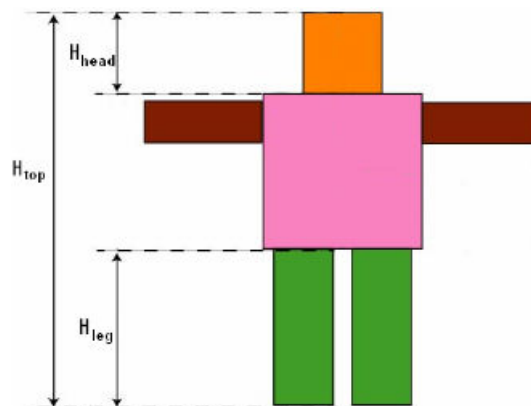


Abb. 1.3.1: Menschenähnlicher Körperbau der Roboter

Regeln werden von Jahr zu Jahr restriktiver. Die Roboter dürfen nur mit passiven Sensoren ausgestattet werden, die Kameras dürfen nur am Kopf des Roboters platziert sein. Seit dem letzten Jahr ist auch der maximale Blickwinkel der Roboter dem Sichtfeld der Menschen angeglichen (180°), was unter anderem den Einsatz von omnidirektionalen Kameras verbietet und weitreichende Konsequenzen für die Selbstlokalisierung und Verhalten hat. Die Roboter müssen vollkommen autonom agieren können, obwohl die drahtlose Kommunikation zwischen den Spielern erlaubt ist, wenn auch auf 1 Mbaud begrenzt. Neben den restriktiven Regelungen bezüglich der Roboterausstattung, werden die Leistungsanforderungen an die Hardware und Software ebenfalls stufenweise erhöht. So wurde in diesem Jahr (2008) die Feldlänge der kid-size-Subliga von 4.5 auf 6 m vergrößert. Die Anzahl der Spieler wird von 2 auf 3 erhöht, was den Spielverlauf variieren soll.

Die Roboter müssen sich selbstständig lokalisieren, Hindernissen ausweichen, Spielstrategien verfolgen und untereinander kooperieren können. Allerdings stellt vor allem in der teen-size Liga ein ganz normales Laufen noch ein Problem dar, was selbstverständlich die Entwicklung der „höheren“ Fähigkeiten verhindert. Die kleineren Roboter der kid-size Liga können zwar schon einigermaßen stabil laufen, scheitern jedoch häufig bei dem Versuch Kollisionen mit den anderen Robotern zu vermeiden, was auf die mangelnde Bewegungspräzision und Geschicklichkeit zurückzuführen ist. Schießverhalten ist eine weitere Motorik-Herausforderung der Liga. Diese an sich einfache Aufgabe wird in der typischen Spielsituation, als die beiden Opponenten es zum Ball geschafft haben, wobei sie sich dadurch gegenseitig beim Schießen behindern, zum Problem. Dribbeln, Passen oder gar den Gegner austricksen ist noch Zukunftsmusik in der Liga.

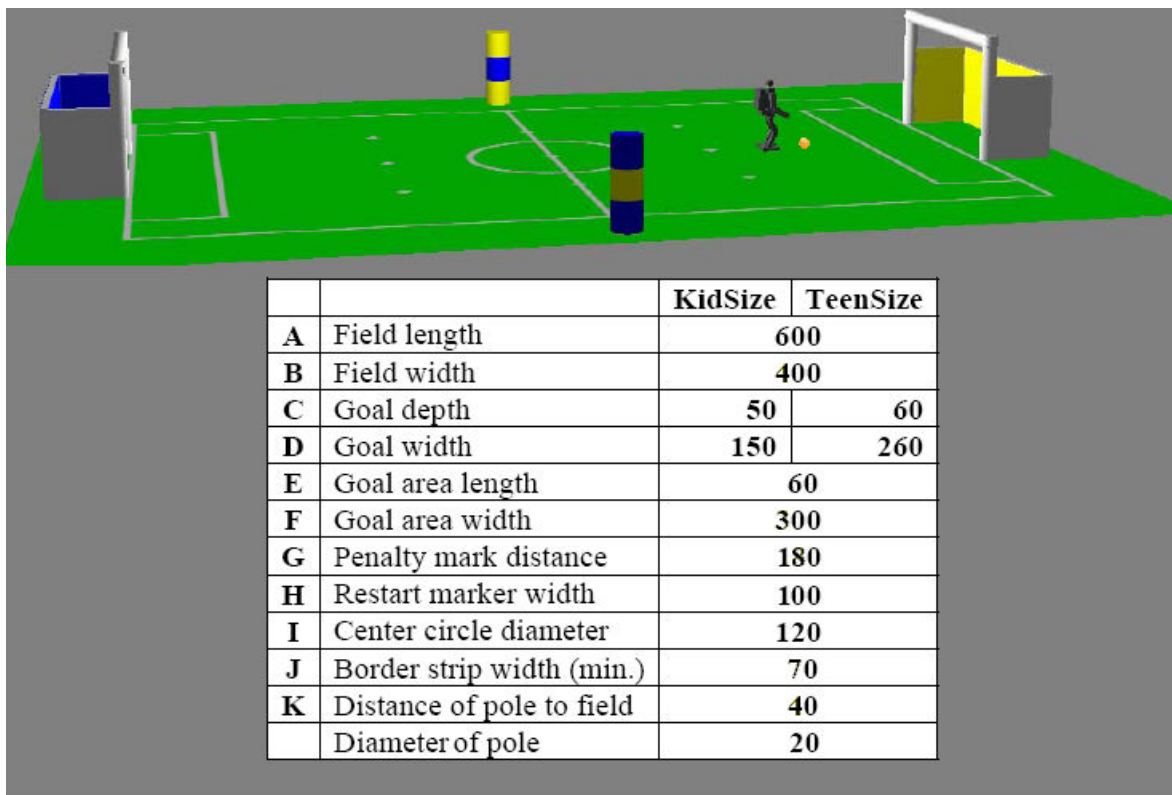


Abb. 1.3.2: Farbkodierung und Abmessungen der relevanten Objekte

Wie aus der oberen Beschreibung ersichtlich wird, sind die elementarsten motorischen Fähigkeiten derzeit noch der Forschungsschwerpunkt in der Humanoiden Liga. Das ist zum einen auf das junge Alter der Liga, zum anderen auf die Komplexität, die der aufrechte Gang mit sich bringt, zurückzuführen. Immerhin kann man bei den weiterführenden Fähigkeiten auf die Erfahrungen, die mittlerweile in den anderen Ligen gemacht wurden, zurückgreifen. Vor allem die AIBO-Roboter der *Standard Platform League* haben angesichts der vergleichbaren sensorischen Ausstattung ähnliche Problemstellungen wie die

Humanoiden, unter anderem auch in der Frage der Selbstlokalisierung. Gemeint ist in erster Linie die Ausrichtung des Videosensors/Sensoren, der ganz nach dem menschlichen Vorbild nach vorne gerichtet ist und einen Öffnungswinkel $\ll 360^\circ$ hat. Das eingeschränkte Sichtfeld macht ein spezielles Suchverhalten erforderlich, bei dem der Roboter die ausgeführte Tätigkeit gelegentlich unterbricht, um nötige Orientierungsinformationen zu sammeln, sich dafür möglicherweise umdrehen/von der Stelle bewegen muss, dann wieder in die Ausgangsposition zurückkehrt und dort weitermacht, wo er aufgehört hat. Das wiederum erfordert Verhaltensschachtelung und ein Weltmodell, in dem das Wissen über den im Moment nicht sichtbaren Teil des Feldes gespeichert wird. Selbstlokalisierung ist eine der Teilaufgaben, wo man mit der Problematik konfrontiert wird.

Trotz der erwähnten Ähnlichkeit bei der sensorischen Roboterausstattung in den beiden Ligen, bringt die abweichende motorische Gestaltung der Humanoiden eine Reihe von zusätzlichen Problemen mit sich. Der Kopf und damit die Kameras der humanoiden Roboter sind während des Laufens heftigen Schwankungen und Erschütterungen ausgesetzt, was eine präzise Messung verhindert. Es ist sehr schwierig diese ungewollten Bewegungen zu vermeiden oder rechnerisch zu kompensieren. Infolgedessen sind robustere Algorithmen gefragt, die mit erheblichen Messfehlern umgehen können. Die ständig variierende Körperneigung macht eine genaue Distanzmessung anhand der Y-Koordinate eines Gegenstands im Bild nahezu unmöglich, selbst wenn der Roboter stillsteht. Auch diese Problematik erschwert die Orientierung im Raum zusätzlich.

Neben der Kameramessung ist Odometrie das zweite klassische Werkzeug der Selbstlokalisierung. Unglücklicherweise ist Odometriemessung bei den Humanoiden besonders problematisch. Motorwertmessungen lassen kaum Rückschlüsse auf die tatsächlich ausgeführte Handlung zu, adaptive Laufmethoden, die bei den Humanoiden häufig zu den Zwecken der Gleichgewichtserhaltung eingesetzt werden, behindern eine zuverlässige Schätzung der zurückgelegten Distanz aufgrund der experimentell ermittelten Werte. Auch die tatsächliche Laufrichtung des Roboters lässt sich nur schwer ermitteln, schon kleinste Hardwarefehler und Bodenunebenheiten wirken sich spürbar auf das Laufverhalten aus. Angesichts der besprochenen Messproblematik gilt das Hauptaugenmerk bei der Selbstlokalisierung der Humanoiden der Robustheit der Verfahren gegenüber gravierenden Messfehlern, nicht der Genauigkeit der Schätzung.

Am Rande sei noch die Problematik der Gewichtsverteilung bei den Humanoiden erwähnt, die Einschränkungen bzgl. der Rechenleistung der Roboter auferlegt.

Leistungsfähige, gut abgeschirmte und mit Lüftern versehene Prozessoren sind schwer und verbrauchen viel Energie, was wiederum dazu führt, dass viele Akkumulatoren mit sich herum geschleppt werden müssen. Das belastet vor allem Knie- und Fußmotoren enorm, die ihrerseits viel Energie verbrauchen und sich außerdem schnell erhitzen. Dadurch verändert sich das Laufverhalten im Laufe des Spiels und das hat dann Einfluss auf andere Fähigkeiten des Roboters oder führt sogar zu Ausfällen. Des Weiteren sinkt die Widerstandsfähigkeit des Systems den Stößen gegenüber mit der steigenden Komplexität, wobei kaum ein Spiel ohne Stürze auskommt, worunter vor allem die oben platzierten Videokameras leiden. Alles in allem bringt die Humanoiden Liga neue Herausforderungen mit sich und stellt härtere Anforderungen an die Robustheit der Hard- und Software. Die Fähigkeit sich aufrecht und menschenähnlich zu bewegen scheint unabdingbar für die Simulation der menschlichen Intelligenz und obwohl die Erforschung der Humanoiden-Technologie im Rahmen des RoboCups noch ganz am Anfang steht, sind sich viele RoboCup-Teilnehmer einig, dass die Liga dazu verdammt ist, die mid-size-Liga als „Königdisziplin des RoboCups“ abzulösen.

Kapitel 2

Team Fumanoid

2.1 Einführung

Das Projekt Fumanoid wurde im Jahr 2007 am Fachbereich für Mathematik und Informatik der Freien Universität Berlin in der Arbeitsgruppe Künstliche Intelligenz gegründet. Das Ziel des Projekts war Entwicklung eines Teams humanoider Roboter für die Teilnahme am RoboCup-Wettbewerb, insbesondere der Weltmeisterschaft 2007 in Atlanta. Nach knapp 3 Monaten intensiver Entwicklungszeit wurde eine Fußballrobotermannschaft auf der Basis des „Edutainment“ Roboterbausatzes „Bioloid“ der Firma ROBOTIS Inc. geschaffen. Die Mannschaft bestand aus lediglich 3 Robotern: ein Spieler, ein Torwart und ein Ersatzspieler. Das Verhalten des Spielers war denkbar einfach und verfolgte die *den Ball finden – zum Ball gehen – um den Ball drehen, das Tor suchen – Schießen* – Grundstrategie. Nicht desto trotz war der erste Wettkampf gleich ein Erfolg für das Fumanoid – das Team landete unter 22 Teams der kid-size Unterliga auf dem 3. Platz.



In dem Regelsatz der humanoiden Liga des RoboCups wurden in diesem Jahr einige Änderungen vorgenommen, auf die bereits eingegangen wurde. Die wichtigste Änderung ist die Erhöhung der Anzahl der Spieler pro Mannschaft von 2 auf 3, was weitreichende Folgen für den Spielverlauf hat. Die neuen Umstände erfordern von den Teams eine etwas ausgefallenerere Spielstrategie, die Kooperation und Kommunikation unter den Spielern vorsieht. Steigende Konkurrenz um den Ballbesitz und Intensivierung des Ringens um die freie Schusslinie sind die Folgen der Neuerung. Passen war bis jetzt kein großes Thema in der Humanoiden Liga, die Pfadfindung ebenfalls. Nun werden diese Fähigkeiten ganz entscheidend für das Spielergebnis sein. Nicht zuletzt hängen alle genannten Kompetenzen von der Fähigkeit sich zu lokalisieren ab. Bis jetzt kamen die Roboter des Teams aufgrund der unkomplizierten Spielstrategie nämlich ohne Selbstlokalisierung aus. Implementierung einer Lokalisierungsmethode, Erweiterung des Verhaltensmusters, sowie Modifikationen der Hardware und die damit verbundenen Änderung der Bewegungssteuerung bedeuten, dass das Programm von Grund auf umgeschrieben werden musste.

2.2 Hardware

Hardware hat insofern einen großen Einfluss auf die Selbstlokalisierungsmethode, als die damit verbundenen Ressourcenbeschränkungen über die Genauigkeit, den Zeitverbrauch und sogar grundsätzliche Machbarkeit entscheiden – es war nämlich bis zum Schluss nicht klar, ob eine halbwegs robuste und präzise Selbstlokalisierung mit den uns zur Verfügung stehenden Mittel überhaupt möglich war. Der Roboter ist mit der bausatzeigenen CM-5 Steuerungseinheit auf der Basis von *dem mit 16 MHz getakteten ATMEL ATMEGA128 8 Bit RISC Hauptprozessor* ausgestattet. Der Vorteil von der einfachen Ausrüstung ist ihre Leichtigkeit, eine Eigenschaft, die in der Humanoiden Liga von besonderer Bedeutung ist. Ein bedeutender Nachteil dagegen ist der fehlende FPU, was vor allem für die bei der Selbstlokalisierung notwendige Triangulation weitreichende Folgen hat. Das Hauptproblem ist jedoch der 4 Kilobyte große RAM, wovon maximal 500 Bytes(!) für die Zwecke der Selbstlokalisierung verwendet werden können. Dabei verbrauchen gerade Echtzeitanwendungen besonders viel Speicher. Dieser Umstand ist eine ganz wesentliche Rahmenbedingung für das gesamte Projekt und hat die Wahl der Lokalisierungsmethode, sowie ihren Entwurf und ihre Implementierung *maßgebend* beeinflusst.

Das Programm wird unter Verwendung vom Bootloader über die Seriellschnittstelle in den 128 Kilobyte großen Flash-Speicher geladen. Im Moment besteht noch kein Problem mit der Programmgröße, allerdings wird der Flash auch für andere Zwecke, wie bspw. Nachschlagtabellen, genutzt und es ist zu erwarten, dass es auch hier zu Engpässen kommen kann. Die Übertragungsgeschwindigkeit beträgt lediglich 57600 bps und die Kommunikation mit der Steuerungseinheit erfolgt sehr langsam. Die Folge ist eine recht lange Programmladezeit, was das Testen des Programms erschwert. Die größte Herausforderung allerdings ist die Kommunikation mit der Kamera, die 160x120 Pixel große Bilder bis zu 10 Mal pro Sekunde liefert. Ihre Übertragung wäre sicherlich ein Problem, weswegen ein spezielles, mit einem eigenen Mikrokontroller ausgestattetes Kamera-Modul, welches in dem Abschnitt 2.3 genauer beschrieben wird, im Rahmen des Projekts entwickelt wurde. Die restlichen zu bewältigende Aufgaben: Verhaltensbestimmung, drahtlose Kommunikation, Steuerung der Motoren und die allgemeine Prozesskoordination erledigt der kleine Mikrokontroller der Steuerungseinheit.

Die Akteure des Systems sind zwölf Dynamixel RX-28 Servomotoren in den Beinen des Roboters und weitere sechs AX-12 Hand- und Halsmotoren.



Abb. 2.2.1: AX-12 und RX-28 Servomotoren

Die Kopf- und Handmotoren sind ein Überbleibsel des alten Systems und um einiges schwächer als die neuen RX-28 (Drehmoment 16.5 gegen 28.3 kg/cm). Der Einsatz der neuen Aktoren erlaubte uns die Laufgeschwindigkeit wesentlich auf bis zu 40 cm/s zu erhöhen. Dafür wurden die Beine der Roboter verlängert und eine adaptive Laufmethode entwickelt, die nur anhand der Motorwerte (Position, Winkelgeschwindigkeit, Belastung) der Knie- und Fußmotoren gleichgewichtserhaltende Korrekturen der Körperhaltung berechnet und die Schrittlänge anpasst. Darüber hinaus können die Roboter nun rückwärts und seitwärts mit der gleichen Methode laufen. In der humanoiden Liga ist die Geschwindigkeit nach wie vor der entscheidende Faktor und wir versprechen uns große Vorteile von den Verbesserungen.

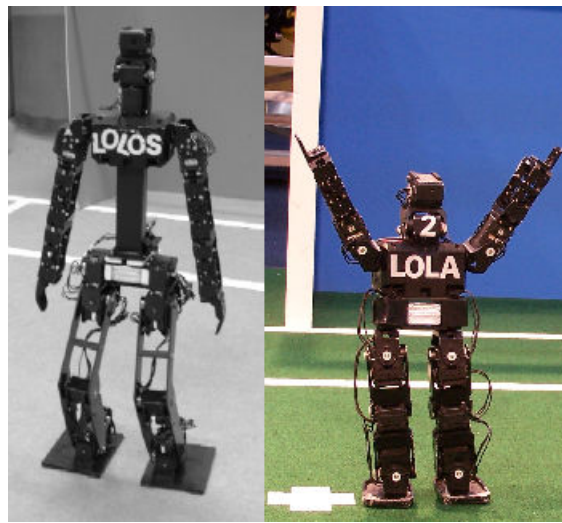


Abb. 2.2.2: Der neue (links) und der alte Spieler im Vergleich

2.3 Kamera-Modul

Das Erfolgsgeheimnis von dem FUmanoid-Team ist das Visionsystem, dessen Kernstück, das Kamera-Modul, es einem erlaubt mit den Ressourceneinschränkungen von CM-5 zurechtzukommen. Das Modul wurde im Rahmen des Projekts entwickelt und ist in der Lage auf dem integrierten Mikrokontroller unter Zuhilfenahme einer LookUp-Tabelle das Bild zu segmentieren. Dabei entfällt die Notwendigkeit das ganze Bild zur Steuerungseinheit zu übertragen, es werden

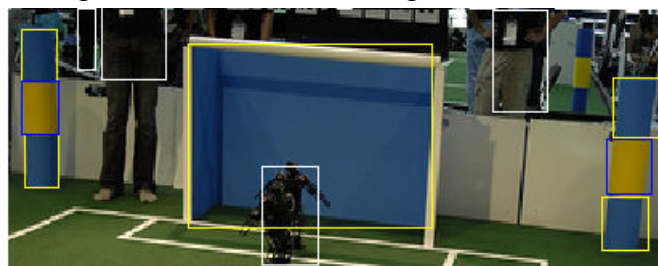


Abb. 2.3.1: Berechnungsergebnis des Kameramoduls

lediglich 4 Schranken(Bounding-Box-Form: minX, maxX, minY, maxY) von maximal 15 Regionen gesendet. Die Segmentierung erfolgt mit einem online-Verfahren, wobei der integrierte Speicher nur für eine einzige Bildzeile ausreicht. Für jede gefundene zusammenhängende Linie einer Zeile kontrolliert man, ob die direkt darüber liegende Zeile eventuell zu einer Region der gleichen Klasse gehört und, falls das zutrifft, mit dieser verschmolzen wird, indem die Grenzen der Region entsprechend aktualisiert werden. Die größte Schwierigkeit besteht darin, eine angemessene Rauschfilterungsmethode zu bestimmen. Geht man dabei zu restriktiv vor, zerhackt man unter Umständen eine Region in viele kleine Teile. Eine zu liberale Methode führt dagegen häufig zur Verschmelzung von auseinander liegenden Objekten. Zu kleine Regionen werden verworfen.

Bei dem Sensor handelt es sich um einen CMOS HV7131GP Kamera IC, der über zahlreiche Einstellungsmöglichkeiten verfügt, einschließlich der unterschiedlichen Ausgabeformate und der Möglichkeit „window of interest“ zu definieren. Der horizontale Öffnungswinkel der Kamera beträgt ca. 45°, der vertikale 30°. Die Farbempfindlichkeit der Kamera ist sehr gut, der Energieverbrauch ist minimal.

Zur Segmentierung des Bildes benötigt der Modul eine LookUp-Tabelle, die

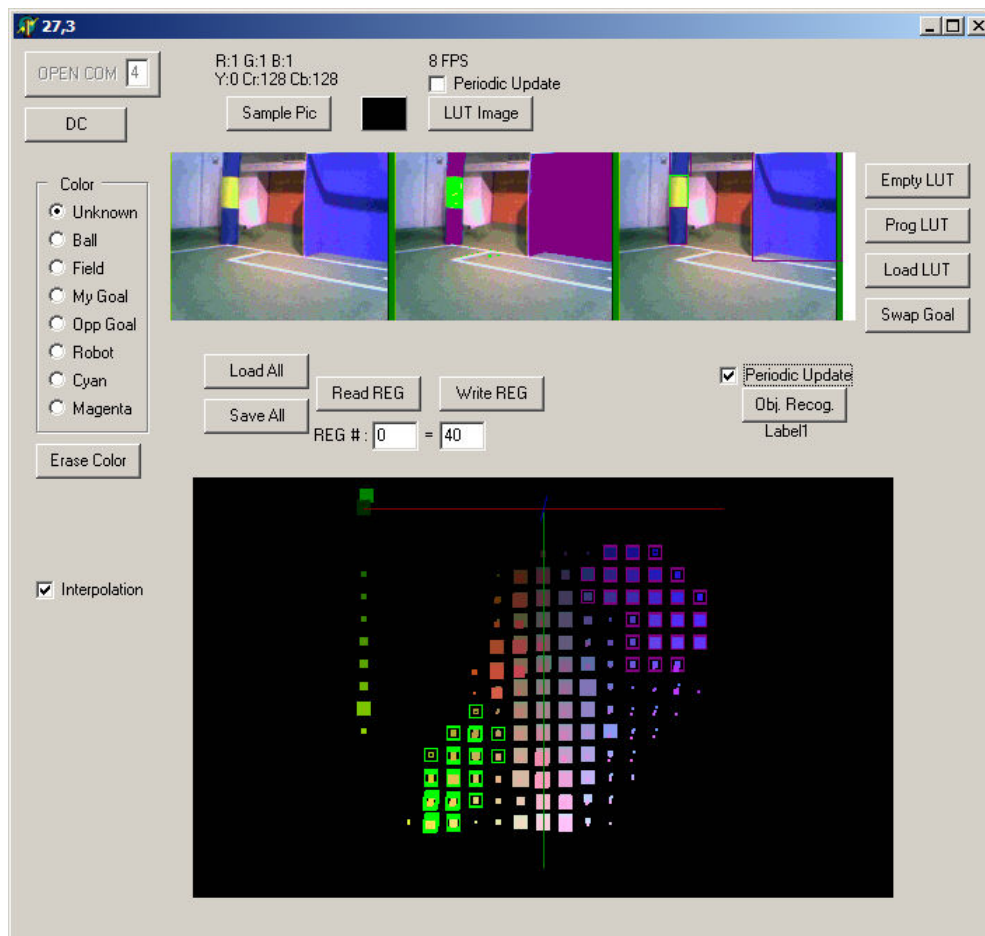


Abb. 2.3.2: Das Farbkalibrierungsprogramm

ebenfalls in dem Flash-Speicher des Moduls gehalten wird. Die Farbkalibrierung erfolgt manuell, da Farberkennung (zurzeit noch) von entscheidender Bedeutung für die Objekterkennung ist und automatische Methoden nicht die benötigten Sorgfalt und Zuverlässigkeit aufweisen. Die Kalibrierung erfolgt mittels eines speziellen Programms, das die Markierung der relevanten Farben (Orange für Ball, Gelb, Blau für Tore/Säulen und Schwarz für Hindernisse) sowohl durch das Anklicken der entsprechenden Pixel im Bildraum, als auch durch die Auswahl von ganzen Punktwolken im Farbraum erlaubt. Künftig sollen auch die weißen Markierungslinien erkannt werden können.

Das Modul verwendet das Kommunikationsprotokoll von Bioloid und verhält sich gegenüber der CM-5 wie ein Motor. Eine Funktion des Visionsystems schickt regelmäßig eine Anfrage an die Kamera und empfängt das Ergebnis, als Statusbericht eines Motors „getarnt“. Dabei kann es zu einem Buskonflikt zwischen den Motoren, CM-5 und der Kamera kommen. Um dem vorzubeugen, wurde auf dem Hauptmikrocontroller ein elementares Betriebssystem implementiert, das die kritische Ressource Bus verwaltet und im nächsten Abschnitt genauer beschrieben wird.

2.4 Software

Selbstlokalisierung ist eine Fähigkeit höherer Ordnung, die das Ergebnis des darunter liegenden Visionsystems nutzt. Darüber hinaus greift man dabei gelegentlich auf die Methoden des Bewegungsmoduls zu, um beispielsweise den Kopf zu drehen. Selbstlokalisierung ist außerdem als Verhalten implementiert und auch die Kommunikationsschnittstelle des Roboters wird genutzt. Somit hat Selbstlokalisierung mit allen Softwaremodulen zu tun, weshalb es wichtig ist, die Grundstruktur des Programms sowie der einzelnen Bausteine zu verstehen.

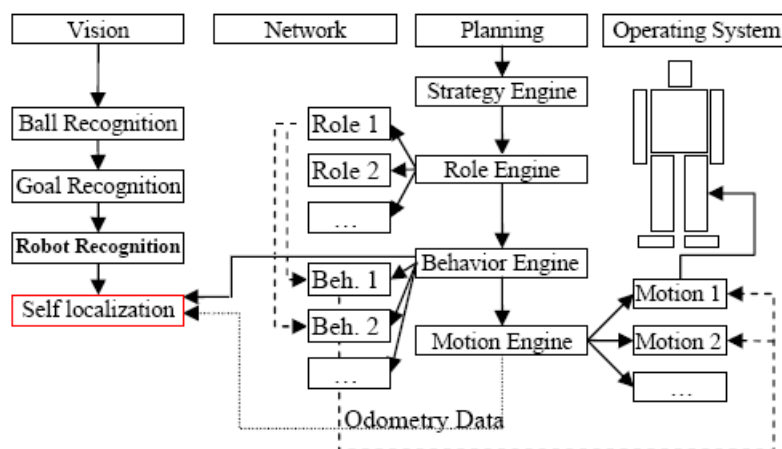


Abb. 2.3.2: Das Farbkalibrierungsprogramm

Im Mittelpunkt steht das Betriebssystem, dessen einzige Aufgabe es ist, prioritätsfähige Threads zu erzeugen und zu verwalten. Weil jeder Thread für genau eine Art von Peripheriegeräten verantwortlich sein soll (Vision - Kamera, Motion Engine –Motoren, Network – Netzwerkkarte), werden damit auch die Ressourcen des Systems verwaltet. Neben der Thread-Erzeugung und Thread-Vernichtung werden Methoden zur Thread-Zurückstellung (sleep) sowie Interprozesskommunikation zur Verfügung gestellt und der Mechanismus der kritischen Sektionen implementiert, mit dessen Hilfe vor allen Dingen die beiden einzigen kritischen Ressourcen, der Prozessor(unteilbare Operationen) und Bus, zur exklusiven Nutzung reserviert werden können. Die Unterbrechungsroutine wird alle 200 μ s ausgeführt. Die Threads nutzen ihre eigenen fest definierten Speicherbereiche und verhalten sich somit ähnlich wie Prozesse eines echten Betriebssystems.

Momentan besteht das Programm aus 6 Threads:

- Der sog. *Goastthread*, der als Benutzerschnittstelle agiert und hat vor allem in der Debugging-Phase eine Bedeutung. Während des Spiels sind lediglich Schiedsrichtersignale wie „Stop“ oder „Kick Off“ erlaubt.
- *VisionEngine-Thread* – ist für die Kommunikation mit der Kamera, Objekterkennung, sowie *Selbstlokalisierung* verantwortlich. Der Thread hat die höchste Priorität, weil der Kamerapuffer lediglich für einen Datensatz ausreicht und die Kamera deswegen nicht lange warten kann. Als erstes wird geklärt, ob das Bild bereits zum Abholen bereit ist und, falls nicht, wird abgewartet, ohne die kritische Sektion zu verlassen, bis es soweit ist. Dieser Synchronisierungsschritt ist hauptsächlich in der Anfangsphase notwendig, danach muss man auf das Bild unter normalen Umständen nicht mehr warten. Anschließend werden die Regionen abgeholt und gleich das nächste Bild veranlasst, sodass die Bildverarbeitung und Bildaufnahme parallel erfolgen. Die Objekterkennung besteht darin, die empfangenen Bildsegmente nach Farben zu sortieren und anhand von weiteren Eigenschaften die Objekte, insbesondere Tore und Säulen, zu identifizieren bzw. nicht-identifizierbare Regionen zu verwerfen. Es wird nicht zwischen den beiden Säulen unterschieden, um an der Stelle Zeit zu sparen. Darüber hinaus ist das in vielen Fällen gar nicht möglich, vor allem nicht, wenn man nur zwei von drei Segmenten der Säule sieht, weswegen unser Selbstlokalisierungs-verfahren auf die Unterscheidung nicht angewiesen ist. Es werden außerdem der Torwart (Gegner vor seinem Tor) und gegnerische Spieler im Ballbesitz (hinter dem Ball) identifiziert. Als letztes wird der Status der Selbstlokalisierung

kontrolliert und ggf. die entsprechende Lokalisierungsmethode aufgerufen und im gleichen Thread ausgeführt, und zwar nur, falls man ausreichend viele Informationen im Bild zur Verfügung hat. Mehr dazu ist im Kapitel 4 „Implementierung einer Monte-Carlo-Selbstlokalisierungsmethode“ nachzulesen.

- *StrategyEngine-Thread* – ist die höchste Planungsebene unseres Steuerungssystems. Hier werden Entscheidungen getroffen, die über längere Zeit, unter Umständen sogar das ganze Spiel über, gültig sind. Die wichtigste Aufgabe ist die Rollenzuteilung an die Spieler, abhängig von der aktuellen Spielsituation, wie beispielsweise Position auf dem Feld, Abstand zum Ball oder zum Gegner.
- *RoleEngine-Thread*. Hier werden alle rollenspezifischen Entscheidungen getroffen, sowie auf die Einhaltung der Regeln geachtet (z.B. Einhalten der vorgeschriebenen Zeitlimite). Im Moment sind 3 Rollen vorgesehen: Torwart, Angreifer und Verteidiger.
- *BehaviorEngine-Thread* steuert die einzelnen Verhaltensmuster des Roboters an, die von dem *RoleEngine* angeordnet werden. Dazu gehören beispielsweise alle Verhaltensmuster aus der letzten Version des Systems, wie Ball suchen, zum Ball gehen, um den Ball drehen. Globale Selbstlokalisierung ist auch ein Verhalten: wenn man seine Position infolge eines Fehlers verloren hat, es aber unbedingt wissen will jedoch im Bild weniger als 2 Objekte sieht, dann muss der Roboter sich auf die Suche nach mehr Information begeben, wobei ein einfaches Umschauen nicht unbedingt weiter hilft. Dann muss der Roboter mehrere Bewegungen ausführen, die im Kontext zueinander stehen und somit ein Verhalten sind.
- Schließlich ist *MotionEngine-Thread* für die Ausführung der einzelnen Bewegungen verantwortlich. Die meisten Bewegungen des Roboters, wie Ballschießen und Aufstehen sind semi-statisch, während das Laufen eine komplizierte adaptive Methode mit einer Vielzahl an Parametern ist. Die semi-statischen Motions werden in Tabellen in dem Flash-Speicher des Roboters gespeichert. Beim Laden der Bewegung werden die Besonderheiten der Konstruktion verschiedener Roboter, die einen großen Einfluss auf das Laufverhalten haben, berücksichtigt und die Motorzielwerte in der Tabelle mit roboterspezifischen Korrekturkonstanten addiert. Ein Keyframe-Interpolator glättet den Bewegungsablauf des Roboters ab. Im Falle der dynamischen Bewegungen werden die Motoren von den jeweiligen Methoden direkt kontrolliert.
- *Netzwerk-Modul* wird vor allem für die drahtlose Kommunikation zwischen den Mitspielern eingesetzt. Das erlaubt den Robotern ihre

Strategie (insbesondere Rollenverteilung) zu koordinieren und kann zum Informationsaustausch eingesetzt werden. Das findet vor allem bei der Ballsuche aber auch bei der Selbstlokalisierungsmethode Verwendung.

Das beschriebene Steuerungsprogramm bringt die Hardware des Roboters an ihre Leistungsgrenzen. Die Module müssen mit dem Speicherplatz von wenigen hundert Bytes auskommen. Die Prozessorzeit ist ebenfalls knapp und wenn ein Modul, wie Selbstlokalisierung mehr davon in Anspruch nimmt, so bekommen die anderen zwangsläufig weniger, was zu Bewegungsfehlern und Kameraaussetzern führen kann. Also galt es für uns entweder die beiden Ressourcen gleichzeitig in den Griff zu bekommen oder ohne Selbstlokalisierung auskommen zu müssen. Die einzige Anforderung, wo man eventuell Abstriche machen konnte, war die Genauigkeit des Lokalisierungsergebnisses. Zusätzlich musste das Sensorik-Handikap der Humanoiden (s. Abschnitt 1.3) bei der Wahl des geeigneten Verfahrens beachtet werden. All das erforderte ein sorgfältiges Studium der Standardmethoden der Selbstlokalisierung mit Beachtung der Modifikationsmöglichkeiten der Verfahren im Hinblick auf die genannten Einschränkungen.

Kapitel 3

Standardmethoden der Selbstlokalisierung

3.1. Einführung

Mit Lokalisierung bezeichnet man in der Robotik die Fähigkeit eines mobilen Roboters, seine Position im Raum in Bezug auf ein festes Koordinatensystem festzustellen. Meistens sind die kartesischen Koordinaten, sowie die Ausrichtung des Roboters vom Interesse. Die Fähigkeit zur Selbstlokalisierung ist eine der wichtigsten Voraussetzungen für ein intelligentes Verhalten, denn ohne jegliches Wissen über die Position des Roboters sind Weltmodellierung und komplexere Handlungsplanungen nicht möglich. Eine absolute Mobilität ist somit ganz ohne Selbstlokalisierung undenkbar.

Zur Strukturierung der Problematik ist Unterscheidung zwischen der lokalen und der globalen Selbstlokalisierung hilfreich. Im Falle der lokalen Selbstlokalisierung ist die Anfangsposition des Roboters bekannt und es geht darum die eventuellen Positionsänderungen zu verfolgen. Das Problem ist in der Fachliteratur unter dem Namen *Position Tracking* bekannt. Die globale Selbstlokalisierung, auch *Wake-up-Robot Problem* genannt, ist eine weitaus schwierigere Problemstellung, wobei die Initialposition von vorne herein unbekannt ist, während unter Umständen mehrere Positionshypothesen in Frage kommen. Noch komplizierter gestaltet sich das „*Kidnapped Robot*“ Problem, wo das lokale Selbstlokalisierungsproblem um die Möglichkeit einer „Teleportation“ des Roboters, erweitert wird - d.h. die Position des Roboters wird sprunghaft verändert, so dass der Roboter nicht in der Lage ist seine Position zu tracken. Diese Vorgehensweise bietet sich als ein ultimativer Test für eine Implementierung der lokalen Lokalisierungsmethode an, denn die Teleportation ist in dem Fall eine Art Simulation eines unvorhersehbaren Ereignisses, das eine extrem große Fehleinschätzung der Position zur Folge hatte. Praktische Lösungsansätze für die Selbstlokalisierung der Roboter sind häufig eine Kombination der beiden Problemstellungen, und zwar aus dem Grund, dass, während die Methoden der globalen Selbstlokalisierung fehlerresistenter sind und besser mit unerwarteten Ereignissen umgehen können, sind die Methoden der lokalen Selbstlokalisierung wesentlich effizienter.

Die große Mehrheit aller Lokalisierungsmethoden, in erster Linie die Methoden für die lokale Selbstlokalisierung, basieren auf der Verschmelzung der Daten unterschiedlicher Sensoren, typischerweise Odometrie mit Ultraschallsensoren,

Laserscanner oder wie in unserem Fall einer Videokamera, wobei kamerabasierte Lokalisierungsmethoden, wegen ihrer Komplexität und Fehleranfälligkeit, am wenigsten verbreitet sind. Im RoboCup Fall stellt das allerdings aufgrund der Gestaltung der Umgebung, etwa Einschränkung der Dynamik und computerfreundliche Einrichtung der künstlichen Landmarken, ein durchaus aussichtsreiches Unterfangen dar. Die Problemstellung der globalen Selbstlokalisierung: die Betrachtung mehrerer Positionshypothesen und Bestimmung der Wahrscheinlichsten aufgrund der fehlerbehafteten Sensordaten, legt den Einsatz von probabilistischen Algorithmen als Grundlage der Berechnung nahe, aber auch von robusten deterministischen Verfahren. Ein Vertreter der nicht-probabilistischen Lokalisierungsmethoden wäre z.B. der Scan-Matching Verfahren, bei dem man zunächst die Position anhand der Odometriedaten vorhersagt und dann versucht die auf diese Weise gewonnenen Daten zu verifizieren bzw. zu korrigieren, indem man die Messungen simuliert, die man in der vorhergesagten Pose machen würde und diese mit den tatsächlich empfangenen Sensordaten vergleicht/integriert.

Bei der Suche nach einer passenden Lokalisierungsmethode wurden mehrere unterschiedliche Ansätze in Betracht gezogen. Im folgendem werden drei wichtigen typischen Verfahren: *Kalman Filter*, *Bayes'scher (auch Markow-) Filter* und *Particle Filter (sequentielle Monte-Carlo-Lokalisierung)* vorgestellt und diskutiert.

3.2. Kalman Filter

Kalman Filter ist ein weit verbreiteter rekursiver Filter, mit dessen Hilfe sich der Zustand eines dynamischen Systems aufgrund von unvollständigen und fehlerhaften Messungen optimal schätzen lässt. Seit der erstmaligen Verwendung im Jahre 1960 wurde der Filter vielfach weiterentwickelt und spezialisiert. Die Einsatzmöglichkeiten sind ebenfalls zahlreich: Zustandsprädiktor, Fehlerfilter, Smoother oder Verknüpfung der Messungen redundanter Daten von unterschiedlichen Sensoren bei linearen und (mit dem erweiterten Kalman) nicht-linearen Systemen. Dank letzterer bietet sich der Kalman Filter für die Selbstlokalisierung der mobilen Roboter an, wobei der Zustand des Roboters durch einen Vektor (x,y,ω) repräsentiert wird, wo x,y - die kartesischen Koordinaten des Roboters auf der Fahrebene und ω die Orientierung (Pose) des Roboters sind. Der Filter liefert eine Positionsschätzung in Form einer parametrisierten Gauß-Kurve. Wegen seiner iterativen Natur gilt der Filter als zeit- und speichereffizient und folglich besonders geeignet für Echtzeitanwendungen.

Der Filter besitzt eine Prädiktor-Korrektur Struktur, bei der in jedem Schritt der erwartete Zustand des Systems im nachfolgenden Schritt aufgrund der Daten aus den vorangegangenen Durchläufe vorhergesagt wird und der Fehler der Prädiktion aus dem letzten Schritt bei den künftigen Schätzungen des Zustandes mittels Anpassung entsprechender Parameter in die Berechnung miteinbezogen wird.

3.2.1 Begriffsklärung

Der Einsatzgebiet für den klassischen Kalman Filter sind zeitdiskrete Systeme mit einer linearen Dynamik. Der Begriff *Zustand* ist grundlegend in der Theorie des Filters und als Element eines Vektorraums auf dem Körper der reellen Zahlen zu verstehen. Neben der Ermittlung des Zustandes ist die Berechnung der Fehlerkovarianzmatrix ein wichtiger Bestandteil des Algorithmus. Als Grundlage für die Prozessmodellierung dient das Modell der Markow-Ketten. Die Dynamik des Systems wird durch lineare Operatoren und der Fehler durch eine normalverteilte Zufallsvariable modelliert. Um einen zeitdiskreten Simulationsschritt der Systemdynamik durchzuführen, wird ein linearer Operator auf den aktuellen Zustand des Systems angewendet und das Ergebnis der Operation wird anschließend mit der normalverteilten Zufallsvariable addiert, wobei ein (weißes) Rauschen im System simuliert wird. Sei x_k der Zustand des Systems zum Zeitpunkt k , F_k , B_k – die zeitabhängigen linearen Operatoren: entsprechend Zustandsübergangs- bzw. Regeleinriffmatrix genannt, und $w_k \sim N(0, Q_k)$ – eine multivariate normalverteilte Zufallsvariable mit der Varianz Q_k , dann wird der Folgezustand des Systems entsprechend der Formel:

$$x_{k+1} = F_{k+1} x_k (+ B_{k+1} u_k) + w_k \quad (\text{F. 3.2.1})$$

aus dem aktuellen Zustand bestimmt. Der Term $B_{k+1} u_k$ simuliert den Einfluss der Steuerungshandlungen auf die Dynamik, falls diese bekannt sind. Manchmal ist auch eine Darstellung des Prozesses in Form von HMM (Hidden Markov Model) von Vorteil (s. Abb. 3.2.3), wobei die echten Zustände des Systems von verborgenen Zuständen des Modells repräsentiert werden.

Neben der Prozessgleichung (F.3.2.1) ist die Messungsgleichung der 2. wichtige Formalisierungsschritt. Gegeben sei y_k - Beobachtung (Messung) zum Zeitpunkt k und C_k die Messungsmatrix – ein Operator, der den echten Zustand x_k auf die zu erwartende, fehlerfreie Messung \hat{y}_k abbildet. Ferner sei die $v_k \sim N(0, R_k)$, eine ähnlich w_k , multivariate, normalverteilte Variable mit der Varianz R_k , stellvertretend für den Messfehler. Dann lässt sich die Abhängigkeit zwischen x_k – dem echten Zustand des Systems und y_k mit:

$$y_k = C_k x_k + v_k \quad (\text{F. 3.2.2})$$

bestimmen. In dem HMM entspricht die Matrix C der Emissionsmatrix. Man beachte, dass die Zufallsvariablen $\{v_0, \dots, v_k, w_0, \dots, w_k\}$ als stochastisch unabhängig behandelt werden. Obwohl im Falle echter Systeme diese Annahme i.A. nicht haltbar ist, weil bspw. die Messfehler häufig auf konkrete Mängel der Messgeräte zurückzuführen sind und dadurch nicht stochastisch unabhängig und normalverteilt sind, liefert der Kalman Filter in den meisten Fällen (wenn die Nicht-Gaußheit nicht allzu groß wird) durchaus zufrieden stellende Ergebnisse.

3.2.2 Herleitung der Berechnungsvorschrift

Der Kalman Filter macht im Prinzip nichts anderes als den mittleren quadratischen Fehler der Zustandsschätzung zu minimieren. Zur Verdeutlichung¹ nehmen wir an, wir würden eine Beobachtungsreihe:

$$y_k = x_k + v_k,$$

wo x – ein unbekannter Signal und v Rauschen, machen und anschließend versuchen aus $y = \{y_k\}$ x_k abzuschätzen. Die Schätzung \hat{x}_k weicht i.A. von dem wahren Wert x_k ab:

$$\tilde{x}_k = x_k - \hat{x}_k \quad (1)$$

Es gilt nun den Wert $J_k = E[\tilde{x}_k^2]$, wo E – Erwartungswert, zu minimieren, unter der Bedingung, dass der Schätzer \hat{x}_k linear sein soll. Das Minimum kann man auf unterschiedliche Art und Weise bestimmen, zum Beispiel, indem man die Orthogonalprojektion von x_k in den von $\{y_1, \dots, y_k\}$ aufgespannten Raum berechnet:

$$E(\tilde{x}_k y_i^T) = 0, \quad i=1, \dots, k-1 \quad (2)$$

Setzt man den Schätzer gleich:

$$\hat{x}_k = G'_k \hat{x}_k^- + G_k y_k \quad (3)$$

die Linearkombination von a-proiri Schätzung (F.3.2.1) und Messung y (F.3.2.2), können wir (1), (2) und (3) zu

¹ Quelle: Kalman filter for vision tracking. Erik Cuevas, Daniel Zaldivar and Raul Rojas [3]

$$E[(x_k - G'_k \hat{x}_k^- - G_k C_k x_k - G_k v_k) y_i^T] = 0 \quad (4)$$

kombinieren. Vorausgesetzt die statistische Unabhängigkeit von Modellrauschen und Messrauschen, haben wir $E(v_k y_i^T) = 0$, wodurch (4) zu

$$E[(I - G_k C_k - G'_k) x_k y_i^T + G'_k (x_k - \hat{x}_k^-) y_i^T] = 0 \quad (5)$$

umgeformt werden kann. Wegen der Orthogonalität ist $E[(x_k - \hat{x}_k^-) y_i^T] = 0$ und wir haben:

$$(I - G_k C_k - G'_k) E[x_k y_i^T] = 0, \text{ für } i = 1, \dots, k-1 \quad (6)$$

woraus folgt $(I - G_k C_k - G'_k) = 0$ und $G'_k = I - G_k C_k$ (7). Wenn man nun (7) in (3) einsetzt, bekommt man:

$$\hat{x}_k = \hat{x}_k^- + G_k (y_k - C_k \hat{x}_k^-) \quad (\text{F. 3.2.3}),$$

was der allgemeinen Form für a posteriori Zustandsschätzung bei allen Anwendungen von dem klassischen Kalman Filter entspricht. Der G –Term, Kalman Gain genannt, kann als Gewichtung der Zweiten der beiden zu fusionierenden Größen gegenüber der Ersten betrachtet werden. Wird der Kalman Filter beispielsweise als Prädiktor eingesetzt, so ist der Kalman Gain die Gewichtung der Messung gegenüber der Prädiktion.

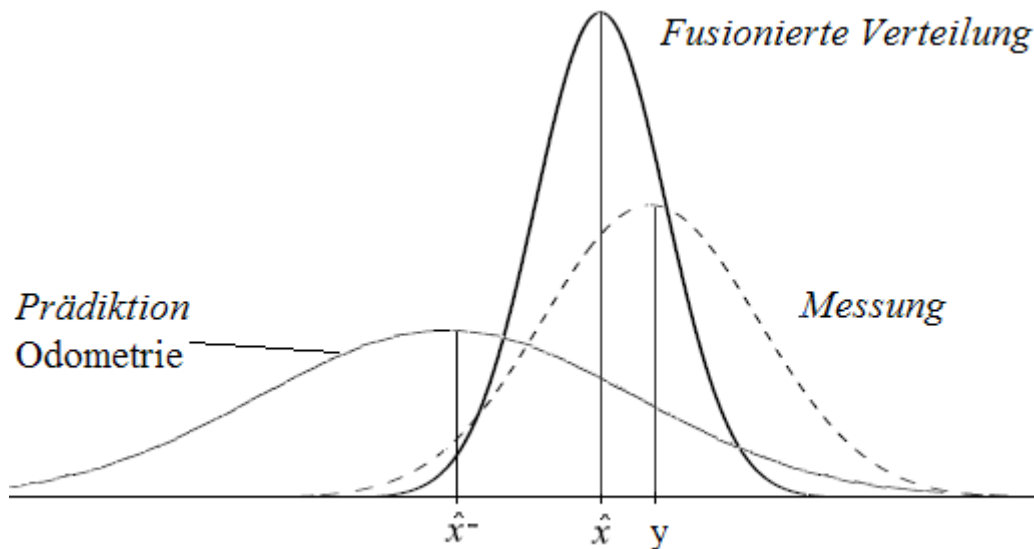


Abb. 3.2.1: Kalman Filter, Kurvenfusionierung

Neben der reinen Zustandsschätzung lässt sich mithilfe des Kalman Filters auch die Gewissheit (Sicherheit) der Schätzung, ausgedrückt durch die

Fehlerkovarianzmatrix P bestimmen, wobei die Berechnungsprozedur der Fehlerkovarianz synchron mit der Zustandschätzung prädiziert/aktualisiert wird. Die Aktualisierungsvorschrift kann man wie folgt herleiten:

$$P_k = \text{cov}(x_k - \hat{x}_k) \stackrel{(F. 3.2.3)}{=} \text{cov}(x_k - \hat{x}_k^- + G_k(y_k - C_k \hat{x}_k^-))$$

Setzt man (3.2.2) für y_k in die Formel ein:

$$P_k = \text{cov}(x_k - \hat{x}_k^- + G_k(C_k x_k + v_k - C_k \hat{x}_k^-)) = \text{cov}((I - G_k C_k)(x_k - \hat{x}_k^-) - G_k v_k) \quad (8)$$

Wegen $\text{cov}(Kx) = K \text{cov}(x) K^T$ und der Unabhängigkeit des v von den restlichen Termen:

$$P_k = (I - G_k C_k) \text{cov}(x_k - \hat{x}_k^-) (I - G_k C_k)^T + G_k \text{cov}(v_k) G_k^T \quad (9)$$

Nennt man $\text{cov}(x_k - \hat{x}_k^-) = P_k^-$ und weil laut (3.2.2) $\text{cov}(v_k) = R_k$:

$$P_k = (I - G_k C_k) P_k^- (I - G_k C_k)^T + G_k R_k G_k^T \quad (10)$$

Diese Kovarianzmatrix-Aktualisierungsvorschrift, lässt sich noch weiter vereinfachen, vorausgesetzt, dass der Kalman Gain G_k optimal ist, nämlich der, mit dem minimalen erwarteten Quadratfehler.

Schließlich können wir die allgemeine Form des optimalen Kalman Gains G_k bestimmen. Der Quadratfehler $E[\tilde{x}_k^2]$ ist nämlich auch dann minimal, wenn die Spur der a posteriori Schätzung der Kovarianzmatrix P_k minimal ist². Auf diese Weise findet man:

$$G_k = P_k^- C_k^T (C_k P_k^- C_k^T + R)^{-1} \quad (F. 3.2.4) \leftrightarrow$$

$$G_k (C_k G_k C_k^T + R_k) G_k^T = P_k^- C_k^T G_k^T \quad (F. 3.2.4a)$$

Wodurch sich (10) zu:

$$\begin{aligned} P_k &= (I - G_k C_k) P_k^- (I - G_k C_k)^T + G_k R_k G_k^T = P_k^- - G_k C_k P_k^- - P_k^- C_k^T G_k^T + \\ &+ G_k (C_k G_k C_k^T + R_k) G_k^T \stackrel{(3.2.4a)}{=} P_k^- - G_k C_k P_k^- - P_k^- C_k^T G_k^T + P_k^- C_k^T G_k^T = \\ &= P_k^- - G_k C_k P_k^- = (I - G_k C_k) P_k^- \quad (F. 3.2.5) \end{aligned}$$

vereinfachen lässt, wodurch die Berechnungskomplexität der a posteriori Schätzung reduziert wird.

² Robot Localization and Kalman Filters On finding your position in a noisy world by Rudy Negenborn

3.2.3 Zusammenfassung

Zusammengefasst:

- *Initialisierung*: geeignete x_0^3 , P_0 . Je mehr über den Initialzustand x_0 bekannt ist, desto kleiner können die Varianzen $P_0(i,i)$ gewählt werden, desto sicherer sind die künftigen Schätzungen. Allerdings sollte man darauf achten, dass die Initialparameter sorgfältig gewählt werden, sonst ist die Erwartungstreue der Schätzer nicht mehr gesichert:

$$E(z_k - x_k) = E(z_k - x_k^-) = 0, \text{ wo } z_k - \text{echter Zustand}$$

$$E(\hat{y}_k - C_k x_k^-) = 0, \text{ wo } \hat{y}_k - \text{fehlerfreie Messung}$$

- *Prädiktion (a priori-Schätzung)*:

$$x_k^- = F x_{k-1} + B u_{k-1} + w \quad (\text{F. 3.2.1})$$

$$P_k^- = F P_{k-1} F^T + Q, \text{ wegen } \text{cov}(Fx) = F\text{cov}(x)F^T \text{ und } \text{cov}(w) = Q$$

- *Update (a posteriori-Schätzung)*:

$$G_k = P_k^- C^T (C P_k^- C^T + R)^{-1} \quad (\text{F. 3.2.4})$$

$$x_k = x_k^- + G_k (y_k - C x_k^-) \quad (\text{F. 3.2.3})$$

$$P_k = (I - G_k C) P_k^- \quad (\text{F. 3.2.5})$$

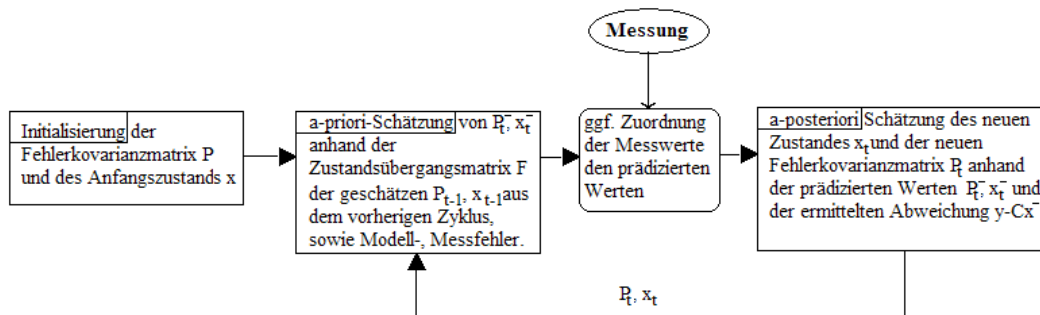


Abb. 3.2.2: Kalman Filter, Flussdiagramm

Am Ende eines jeden Schrittes hat man somit eine Positionsschätzung und, zusätzlich, die Gewissheit der Schätzung in Form von Fehlerkovarianzmatrix P , wobei die Schätzung nachgewiesenerweise optimal ist, d.h. hinsichtlich des Quadratfehlers liegt uns die bestmögliche (unter allen linearen Schätzern) Schätzung vor. Wie bereits erwähnt, ist der Filter außerdem, dank seiner rekursiven Natur, sehr speicher- und platzeffizient und gut konditioniert, was ihn zu einem der meistverwendeten Filter überhaupt gemacht hat. In der Robotik wird er, wegen der nicht-linearen Prozessdynamik wie Beschleunigung, häufig in

³ In der Zusammenfassung wird der a priori Schätzer \hat{x}^- und a posteriori-Schätzer \hat{x} einfach nur x^- bzw. x genannt, weil die Unterscheidung Schätzer - echter Zustand nicht mehr benötigt wird.

Beide zählen zu probabilistischen Methoden und haben die Markow-Ketten als Grundlage der Prozessmodellierung. Wie bereits erwähnt, kann man den Kalman Filter selbst als den optimalen Zustandsschätzer für die verborgenen, echten HMM-Zustände betrachten. Sie besitzen eine ähnliche Prädiktor-Korrektur Struktur, bei der in zwei Schritten Odometriedaten mit den Messdaten verknüpft werden. Schließlich basieren die beiden Verfahren auf der Bayes-Filterung, deren Beschreibung im nächsten Kapitel zu finden ist.

3.3. Markow-Lokalisierung

Der größte Unterschied zwischen der auf der Kalman Filterung basierenden Selbstlokalisierungsverfahren und *Markow-Lokalisierung* ist wohl der, dass im Falle des Kalman-Filters die Hypothesen parametrisch mithilfe einer oder mehreren (MHKF) Normalverteilungen dargestellt werden, während bei der Markow-Lokalisierung die Schätzung mittels stückweise linearen Funktionen erfolgt. Dieser Ansatz hat den Vorteil, dass man damit flexibler bei der Approximation von der Verteilungsfunktion der Positionsschätzung ist, obwohl eine stückweise lineare Funktion weniger aussagekräftig und weniger genau als eine Gauß-Kurve ist, wenn es darum geht Prozesse der realen Welt stochastisch zu beschreiben.

3.3.1 Bayes Filter

Bayes Filter dient als Grundlage für die meisten verbreiteten Lokalisierungsverfahren und ist eine Berechnungsvorschrift für die Schätzung der Positions-Verteilungsfunktion anhand der Messdaten. Der Filter hat seinen Namen von der Bayes-Formel für die Rechnung mit bedingten Wahrscheinlichkeiten, die es einem erlaubt die Wahrscheinlichkeit von Umkehrschlussfolgerungen (Abduktion) zu bestimmen. Es ist nämlich häufig so, dass es einfacher ist, die Wahrscheinlichkeit einer Beobachtung $p(y|x)$ in einem vorgegebenen Zustand x direkt zu bestimmen als umgekehrt ($p(x|y)$ – auch posterior Wahrscheinlichkeit genannt). Mithilfe von der *Formel von Bayes* kann man $p(x|y)$:

$$p(x|y) = p(x \wedge y)/p(y) = p(y|x) \cdot p(x)/p(y) \text{ oder in einer allgemeineren Form} \\ p(x|y_1, \dots, y_k) = p(y_k|y_1, \dots, y_{k-1}, x) \cdot p(x|y_1, \dots, y_{k-1}) / p(y_k|y_1, \dots, y_{k-1}) \quad (\text{F.v.Bayes})$$

bestimmen. Weil die Wahrscheinlichkeit $p(y)$ auch nicht immer gegeben ist, macht man sich häufig die Formel für die Berechnung der *totalen Wahrscheinlichkeit* zunutze:

$$p(y_k) = \int_i p(y_k | x_i) \cdot p(x_i) \quad (\text{F. 3.3.1}),$$

wo $\{x_i\}$ disjunkte Zerlegung vom Zustandsraum X .

Für das Lokalisierungsproblem kann man diese Formel wie folgt verwenden: Angenommen man möchte die Positions-Verteilungsfunktion $\{\text{Bel}(x_k^i)\}$ aufgrund der Messdatenreihe $(y_k, o_{k-1}, \dots, o_2, y_2, o_1, y_1)$ ⁵ schätzen. Bei den Messungen handelt es sich nach wie vor um die Odometriedaten $\{o_i\}$ und Messdaten anderer Sensoren (z.B. einer Videokamera) $\{y_i\}$. Ferner sei angenommen, dass es sich bei dem Prozessmodell um eine Markow Kette handelt. Dann kann man die Verteilungsfunktion mittels einer stückweise linearen Funktion über dem gesamten Positionsraum: $\text{Bel}(Z_k = x_i) : (x^i = (x, y, \omega)) \rightarrow [0,1]$ mithilfe der Formel von Bayes wie folgt bestimmen:

$$\begin{aligned} \text{Bel}(Z_k = x_i) &= p(x_k^i | y_k, o_{k-1}, \dots, o_2, y_2, o_1, y_1) \stackrel{(\text{F.v.Bayes})}{=} & (1) \\ &= p(y_k | x_k^i, o_{k-1}, \dots, o_2, y_2, o_1, y_1) \cdot p(x_k^i | o_{k-1}, \dots, o_2, y_2, o_1, y_1) / p(y_k | o_{k-1}, \dots, o_2, y_2, o_1, y_1) \end{aligned}$$

Der Term $p(x^i | o_{k-1}, \dots, o_2, y_2, o_1, y_1)$ ist als Positionsschätzung anhand der ersten $k-1$ Daten zu verstehen. Wie man bereits ahnen kann, arbeitet der Bayes Filter, ähnlich wie der Kalman Filter, *rekursiv*, mit dem Initialzustand als Rekursionsanker. Seiner iterativen Natur, sowie der *Markow'schen Eigenschaft* hat er seine Speicher- und Zeiteffizienz zu verdanken. Die Markow'sche Eigenschaft hat bekanntlich folgenden mathematischen Ausdruck:

$$p(Z_{t+1} = x_{j(t+1)} | Z_t = x_{j(t)}, Z_{t-1} = x_{j(t-1)}, \dots, Z_1 = x_{j(1)}, Z_0 = x_{j(0)}) = p(Z_{t+1} = x_{j(t+1)} | Z_t = x_{j(t)}) \quad (\text{F. 3.3.2})$$

Damit lässt sich die Formel (1) wesentlich vereinfachen:

$$\text{Bel}(Z_k = x_i) = p(y_k | x_k^i) \cdot p(x_k^i | o_{k-1}, \dots, y_1) / p(y_k | o_{k-1}, \dots, y_1) \quad (2)$$

Der Ausdruck $p(y_k | o_{k-1}, \dots, y_1)$ ist unabhängig von x^i und kann als Normierungskonstante angesehen werden.

Ersetzt man in (2) $1/p(y_k | o_{k-1}, \dots, y_1)$ durch α , dann ist:

⁵ Die Reihenfolge der Messdaten ist für die Markow-Selbstlokalisierungsmethode unerheblich. Mit der Reihenfolge $(y_k, o_{k-1}, \dots, o_2, y_2, o_1, y_1)$ wird der Normalfall, bei dem die Odometrie- und Kamerasensoren abwechselnd abgelesen werden, markiert.

$$\text{Bel}(Z_k = x_i) = \alpha p(y_k | x_k^i) \cdot p(x_k^i | o_{k-1}, \dots, y_1) \quad (3)$$

Nun gilt es den Term $p(x_k^i | o_{k-1}, \dots, y_1)$ zu bestimmen. Anhand der Formel für die Totale Wahrscheinlichkeit sieht man, dass

$$p(x_k^i | o_{k-1}, \dots, y_1) = \int p(x_k^i | x_{k-1}^{i'}, o_{k-1}, \dots, y_1) \cdot p(x_{k-1}^{i'} | o_{k-1}, \dots, y_1) dx_{k-1}^{i'} \quad (4),$$

wobei $x_{k-1}^{i'}$ Position zum Zeitpunkt k-1. Gegeben die Markow'sche Eigenschaft:

$$\begin{aligned} p(x_k^i | o_{k-1}, \dots, y_1) &= (4) = \int p(x_k^i | x_{k-1}^{i'}, o_{k-1}) \cdot p(x_{k-1}^{i'} | o_{k-1}, \dots, y_1) dx_{k-1}^{i'} =^{(1)} \\ &= \int p(x_k^i | x_{k-1}^{i'}, o_{k-1}) \cdot \text{Bel}(Z_{k-1} = x_{i'}) dx_{k-1}^{i'} \end{aligned} \quad (\text{F.3.3.3})$$

Setzt man (F.3.3.3) in die (3) ein, bekommt man die rekursive Berechnungsvorschrift für die Positionsschätzung:

$$\text{Bel}(Z_k = x_i) = \alpha p(y_k | x_k^i) \cdot \int p(x_k^i | x_{k-1}^{i'}, o_{k-1}) \cdot \text{Bel}(Z_{k-1} = x_{i'}) dx_{k-1}^{i'} \quad (\text{F. 3.3.4})$$

Beim näheren Betrachten erkennt man in der Rekursion die Prädiktor-Korrektur Struktur des Bayes Filters und stellt Ähnlichkeiten zum Algorithmus der Kalman Filterung fest. Formt man die Kalman-Filter-Berechnungsvorschrift entsprechend um, so wird die Ähnlichkeit der beiden Verfahren offensichtlich. Die a-priori Schätzung (Prädiktion) ist nämlich die Positionsschätzung des Zustandes x_k^- anhand der k-1 ersten Elemente der Messungsdatenreihe, vergleichbar mit (4). Für den Kalman Filter lautet der Ausdruck:

$$p(x_k^-) = p(x_k | y_1, \dots, y_{k-1}) =^6 \int N(Fx_{k-1}, Q) \cdot N(x_{k-1}, P_{k-1}) dx_{k-1}^*$$

Und die a-posteriori Schätzung des Kalman Filters mit Bayes sieht folgendermaßen aus:

$$\begin{aligned} p(x_k) &= p(x_k | y_1, \dots, y_k) = N(C x_k^-, R) \cdot p(x_k^-) / p(y_k | y_{k-1}) =^{(\text{Totale Wkt})} \\ &= N(C x_k^-, R) \cdot p(x_k^-) / p(y_k | y_{k-1}) =^* \\ &= \alpha N(C x_k^-, R) \cdot \int N(Fx_{k-1}, Q) \cdot N(x_{k-1}, P_{k-1}) dx_{k-1}, \end{aligned} \quad (\text{F. 3.3.5})$$

mit $\alpha = 1/p(y_k | y_{k-1}) = 1/\int N(C x_k^-, R) \cdot p(x_k^-) dx_k$ – Normierungskonstante.

Hält man nun (F. 3.3.5) und (F. 3.3.4) nebeneinander, dann identifiziert man die einzelnen Terme des Bayes Filters hinsichtlich ihrer Stellung bei der

⁶ F- Zustandsübergangsmatrix, Q- Kovarianzmatrix der Modell-Ungenauigkeit, P- Fehlerkovarianzmatrix der Schätzung, C- Messungsmatrix, R- Messfehlerkovarianzmatrix. Die Regelneingriffsmatrix B sei vernachlässigt.

Prozessmodellierung. Der Ausdruck $p(y_k|x_k^i)$ ist offensichtlich der „Messungsteil“ der Formel und gehört somit in die (Korrektur-) Aktualisierungsphase des Algorithmus. Dagegen ist $p(x_k^i|x_{k-1}^i, o_{k-1})$ die Wahrscheinlichkeit dafür, aus dem Zustand $Z_{k-1} = x_{k-1}^i$ zum Zeitpunkt $k-1$ in den Zustand $Z_k = x_k^i$ zu kommen, unter der Voraussetzung, dass man die Odometriedaten o_{k-1} empfangen hat. Man berechnet die Wahrscheinlichkeit für alle möglichen Vorgängerzustände x_{k-1}^i und bestimmt auf diese Weise die Gesamtwahrscheinlichkeit dafür, mit o_{k-1} zur Position x^i zu gelangen. Damit gehört der ganze rechte Teil der Formel zur a priori Schätzung der Position. Die Verschmelzung der Daten besteht bei der Markow-Methode aus der Multiplikation der a-priori Wahrscheinlichkeit, an die Position x^i zu kommen, mit der Wahrscheinlichkeit an der Position x^i y_k zu messen. Die a priori Positionsschätzung anhand der Odometriedaten bezeichnet man in dem Zusammenhang üblicherweise *Bewegungsmodell*, die Korrektur des Belief-Wertes unter Zuhilfenahme der Messdaten - *Sensormodell*.

3.3.2 Bewegungsmodell

Die Implementierung des Bewegungsmodells stellt hier ein großes Problem dar. Insbesondere im Falle der humanoiden Roboter hat man nur Informationen über die „Absichten“ des Roboters und kaum Kenntnis über den Verlauf der Befehlsausführung zur Verfügung. Obwohl in dem von uns implementierten Bewegungsmodell stellenweise auch Daten über die Motorwerte während der Bewegungsausführung (eigentlich für die Gleichgewichtserhaltung ermittelt) miteinbezogen werden, gestaltet sich die Bewegungsüberwachung humanoider Roboter wesentlich schwieriger als die fahrender Roboter. Die Folge ist kleinere Genauigkeit der a-priori Schätzung, die durch ein mächtigeres und fehlerresistenteres Sensormodell ausgeglichen werden muss.

Das Bewegungsmodell kann formal als Funktion $\text{Motion_Model}: (x_{k-1}, o_k, x_k) \rightarrow [0,1]$ beschrieben werden, wo x_{k-1}, x_k Elemente des (x,y,ω) -Raums und o_k Odometriemessung (wird vorwiegend durch die auszuführende Bewegungsanweisung festgelegt). Die Auswirkungen eines Bewegungsbefehls auf die einzelnen Komponenten des Zustandsvektors lassen sich meistens problemlos ermittelt, so sind sie im Wesentlichen Inversionen von den Bewegungskontrollfunktionen, die, gegeben eine Startposition x_{k-1} und eine Zielposition x_k , den erforderlichen Steuerungsbefehl o_k bestimmen. Allerdings ist kein reales mobiles System fehlerfrei und Bewegungen der Roboter sind fehlerhaft. Dieser Fehler wird meist mithilfe der Gauß-Verteilung mit $(0,0,0)$ als Erwartungswert modelliert, obwohl die Verwendung der Normalverteilung hier, im Gegensatz zur Kalman Filterung, nicht zwingend ist. Gerade die humanoiden

Systeme, wo selbst einfaches geradliniges Vorwärtslaufen ein Forschungsfeld ist, bedürfen manchmal eine ausgefallene Fehlermodellierung.

Die Fehlerverteilungen der 3 Komponenten des Zustandsvektors (x,y,w) ist für unterschiedliche Kontrollanweisungen verschieden. Beim geradlinigen Laufen kommt es bei den Humanoiden zur Fehleinschätzung der zurückgelegten Distanz in die Laufrichtung, zur seitlichen Abweichung von der Ideallinie infolge der ungleichen Schrittlänge der beiden Beine und zur unerwünschten Änderung der Körperorientierung aus verschiedensten Gründen wie z.B. Bodenunebenheiten. Gerade die letzteren sind besonders belastend, insbesondere wenn sie zu systematischen Fehlern gehören, d.h. wenn sie einseitige Abweichung von dem Erwartungswert bewirken. Fehleinschätzungen der Orientierung nehmen zwangsläufig Einfluss auf die seitliche Verschiebung und diese beeinflusst ihrerseits den Distanzfehler (s. Abb.3.3.1a). Weil es sich bei den 3 Fehlern um die s.g. Trendfehler handelt, steigen die Varianzen mit der zurückgelegten Strecke d , die ihrerseits von der Odometriemessung o_k abhängt:

$$\Sigma(o_k) = \begin{pmatrix} \sigma_{\text{Distanz}}^2(o_k) & \text{cov}(\Delta_D, \Delta_S, o_k) & \text{cov}(\Delta_D, \Delta_W, o_k) \\ \text{cov}(\Delta_D, \Delta_S, o_k) & \sigma_{\text{Seitl.}}^2(o_k) & \text{cov}(\Delta_S, \Delta_W, o_k) \\ \text{cov}(\Delta_D, \Delta_W, o_k) & \text{cov}(\Delta_S, \Delta_W, o_k) & \sigma_{\text{Winkel}}^2(o_k) \end{pmatrix} \quad (\text{F. 3.3.6}),$$

wo $\sigma_{\text{Distanz}}^2$, $\sigma_{\text{Seitl.}}^2$, σ_{Winkel}^2 – Distanzfehler-, Seitwärtsabweichungs-, Winkelfehlervarianz,
 $\text{cov}(\Delta_D, \Delta_W, o_k)$, $\text{cov}(\Delta_S, \Delta_W, o_k)$, $\text{cov}(\Delta_D, \Delta_S, o_k)$ – Fehlerkovarianzen.

Die Simulation der Bewegung kann schließlich entsprechend der Verteilung:

$$x_k \sim N(\mu(x_{k-1}, o_k), \Sigma(o_k)) \quad (\text{F. 3.3.7}),$$

erfolgen, wo $\mu(x_{k-1}, o_k)$ –die erwartete Endposition, $\Sigma(o_k)$ –Fehlerkovarianzmatrix und N –angenommene Normalverteilung des Fehlers.

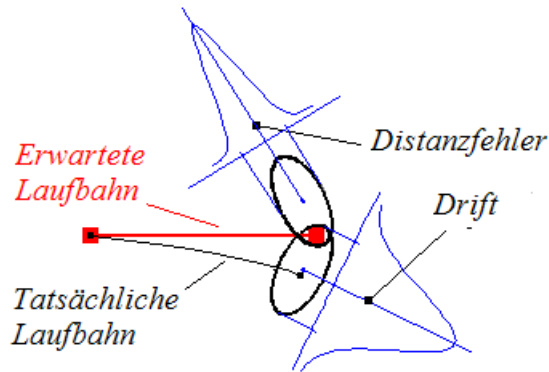


Abb. 3.3.1a: Bewegungsmodell, Distanzfehler-Driftfehler-Korrelation

Die Beschreibung der erwarteten Zielposition sowie der erwarteten Abweichung von der erwarteten Zielposition erfolgt mittels einer für jedes einzelne Verhalten (s.2.4) experimentell ermittelten Erwartungsvektor bzw. der entsprechenden Kovarianzmatrix. Eine analytische, realitätsnahe Beschreibung der Kovarianzen von den 3 erwähnten Fehlerarten erwies sich speziell für die Humanoiden als schwierig.

Nachdem die Kovarianzmatrix ermittelt wurde, lässt sich die Wahrscheinlichkeit $p(x_k^i | x_{k-1}^i, o_{k-1})$ mit der Verteilungsfunktion der Normalverteilung Φ berechnen.

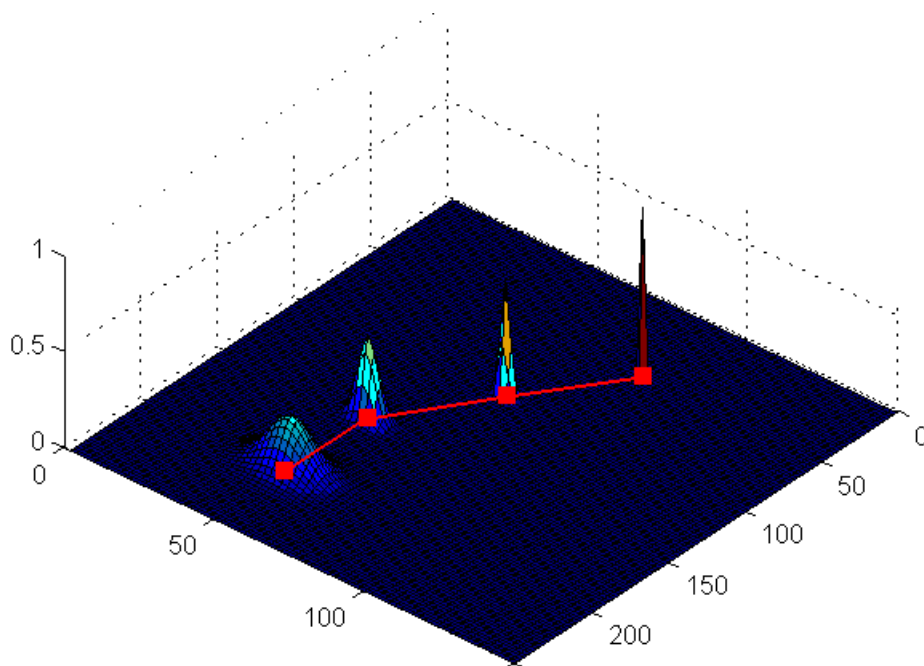


Abb. 3.3.1.b: Bewegungsmodell, Fehlerfortpflanzung

3.3.3 Sensormodell

Entwicklung einer Methode für die Bestimmung der Wahrscheinlichkeit $p(y_k | x_k^i)$, die Messdaten y_k zu empfangen unter der Bedingung das man sich im Zustand x^i befindet, ist die größte Herausforderung der Markow-Lokalisierung. Die Genauigkeit und Robustheit der Methode entscheiden über das Erfolg des Verfahrens, insbesondere wenn man an die oben beschriebenen Schwächen der Bewegungsmodellierung der humanoiden Systeme denkt. Ein mächtiges Sensormodell ist auch deshalb wichtig, weil die Wake-Up-/Kidnapped-Robot-Problemstellung Selbstlokalisierung in Situationen verlangen, in denen man nur Sensordaten zur Verfügung hat.

Üblicherweise werden selbstlokalisierende mobile Systeme mit Scannvorrichtungen ausgestattet, die eine vergleichsweise kleine Datenrate haben und trotzdem eine Rundumsicht liefern. Die Datenrate erlaubt es sogar unter Umständen mit rohen Sensordaten zu arbeiten⁷, was praktisch ausgeschlossen ist, wenn eine Videokamera als Sensor eingesetzt wird. Aber unabhängig davon, welches Sensormodell verwendet wird, besteht die Verarbeitungsprozedur in vielen Fällen aus folgenden Schritten:

1. Berechnung der für die gegebene Position x_k^i erwarteten fehlerfreien Messung \hat{y} . Diese wird üblicherweise anhand eines eigens dafür bestimmten Umgebungsmodells⁸ ermittelt.
2. Lösung des Teilproblems für die einzelnen relevanten Objekte der Umgebung unter Einbeziehung der Messungsfehlermodellierung.
3. Verschmelzung der ermittelten Wahrscheinlichkeiten zu einer gemeinsamen Verteilung.

Offensichtlich wird bei 2 die vorangegangene Objekterkennung/-zuordnung vorausgesetzt (s. 2.3). Es ist auch denkbar sich die Zuordnung von erkannten Objekten zu Modellobjekten zu ersparen und, stattdessen, das Ergebnis der Berechnung unter 2 zu filtern, indem man beispielsweise die bedingten Wahrscheinlichkeiten für alle oder mehrere heuristisch bestimmten Zuordnungen

⁷ Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids, Wolfram Burgard and Dieter Fox and Daniel Hennig and Timo Schmidt

⁸ Eine automatische Erstellung eines solchen Modells (Kartenerstellung) ist ebenfalls ein interessantes Problem der Selbstlokalisierung (z.B. Globale Selbstlokalisierung für mobile Service Roboter, Joachim Weber), ist allerdings derzeit kein Thema in der RoboCup Liga der Humanoiden.

berechnet und anschließend die maximale auswählt, allerdings lohnt sich das bei den eindeutig markierten Objekten der RoboCup-Umgebung nicht.

Der Messfehler wird sinngemäß größtenteils durch eine Normalverteilung modelliert, es müssen jedoch manchmal einige folgenreiche Ausnahmefälle beachtet werden, die die Fehlerverteilung fühlbar verändern (s. Abb.3.3.2). Durch eine gewisse Dynamik des Systems kommt es immer wieder zu Fehlererkennung von Objekten, die das zufällige Messrauschen hinsichtlich der Folgen für die Fehlerverteilung weit übertrifft. Dazu zählen in erster Linie Nicht-Erkennung von Objekten auf dem Bild und Entdeckung der „Phantomobjekte“, wobei letzteres in der Regel problematischer ist, während das erstere erfahrungsgemäß häufiger vorkommt. Die große Herausforderung hier ist eine geeignete Handhabung der Ausnahmefälle zu finden, denn eine zu restriktive Herangehensweise führt dazu, dass überall auf dem Feld die Wahrscheinlichkeit $p(y_k|x_k^i) = 0$ wird, womit die gesamte Positionsschätzung scheitert. Eine allzu sehr freizügige Fehlerbehandlung kann dagegen Fehllokalisierung verursachen. Ein ähnliches Problem hat man bei der Wahl einer geeigneten Varianz für die Modellierung des zufälligen Messfehlers, wobei die Standardabweichung der Messung in Abhängigkeit von den Lichtverhältnissen und Kamerabewegungsgeschwindigkeit variiert.

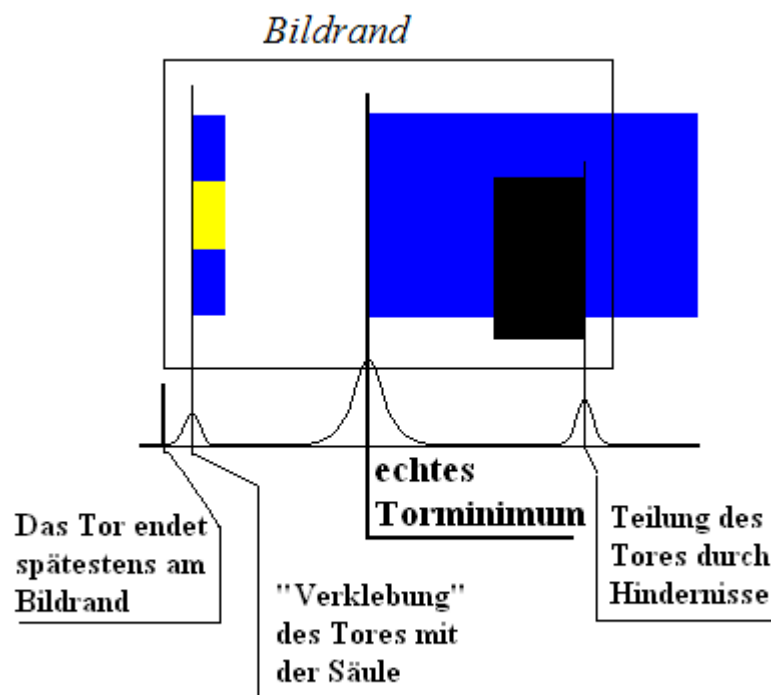


Abb. 3.3.2: Sensormodell, Fehlererkennung des Torminimums

Im letzten Schritt erfolgt Fusionierung der Wahrscheinlichkeitsschätzungen für die einzelnen Landmarken zur Gesamtwahrscheinlichkeit $p(y_k | x_k^i)$. Im einfachsten Fall werden die Messungen als unabhängige Ereignisse behandelt und mit

$$\prod_j p(y_k^j | x_k^i)$$

kombiniert, wo y_k^j – Messung der Landmarke j zum Zeitpunkt k . Die Unabhängigkeitsannahme stimmt in manchen Ausnahmefällen nicht, diese sollten allerdings bereits im Schritt 2 erkannt werden und vorsichtshalber eine Sonderbehandlung bekommen.

Die im 1. Kapitel beschriebenen neuen Regelungen für die Sensorik der Humanoiden, wie beispielsweise die Einschränkung des Sichtfeldes, verstärken die Komplexität des Sensormodells von humanoiden Robotern zusätzlich. Die Besonderheiten des Modells werden im Abschnitt 4.3 „Implementierung des Sensormodells“ beschrieben und diskutiert.

3.3.4 Zusammenfassung

Mit den oben beschriebenen Sensor- und Bewegungsmodellen haben wir nun alle für die Markow-Lokalisierung nötigen Teile zusammen. An der Stelle ist es sinnvoll von der bisher weitgehend allgemeingehaltenen theoretischen Betrachtungsweise des Bayes Filters hin zu praxisnahen Überlegungen überzugehen. Das bedeutet in erster Linie eine Diskretisierung des Zustandsraums X :

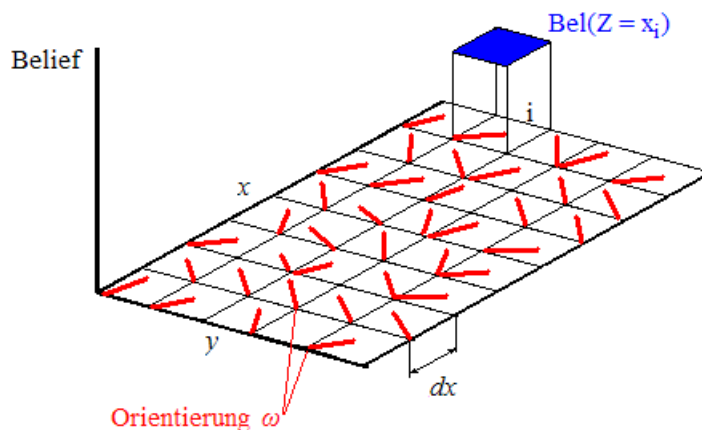


Abb. 3.3.3: Markow-Lokalisierung, Diskretisierung des Zustandsraums

mittels eines dreidimensionalen Gitters.⁹ Jeder Zelle $x^i = (x, y, \omega)$ wird zu jedem Zeitpunkt k ein *Belief*-Wert: $Bel(Z_k = x_i)$ entsprechend der Berechnungsvorschrift

⁹ Dieses war bereits bei der Einführung der Notation $Bel(Z_k = x_i) : x^i \rightarrow [0,1]$ und den theoretischen Betrachtungen des Bewegungsmodells angebracht, als es darum ging die

(F. 3.3.4) zugeordnet, wobei statt Integration die Summe $\sum p(x_k^i | x_{k-1}^{i'}, o_{k-1}) \cdot Bel(Z_{k-1} = x_{i'})$ berechnet wird, was der linearen Interpolation der Dichtefunktion entspricht. Die Genauigkeit der Methode kann man mittels Veränderung der Gittergröße dx regulieren, was sich allerdings quadratisch auf die Berechnungszeit auswirkt.

Pseudocode der Markow'schen Methode:

```

/*
Initialisierung: In der Phase des Programms wird das Wissen über die
Initialposition des Roboters als Verteilungsfunktion dargestellt. Falls die exakte
Position  $x^j$  bekannt ist, wird ein einziger Belief-Wert  $Bel(Z_0 = x_j) = 1$  und der Rest
= 0 gesetzt. Falls gar nichts über den Anfangszustand bekannt ist, ist es meist
sinnvoll Gleichverteilung anzunehmen und alle Beliefs =  $1/(\text{Anzahl der Zellen})$  zu
setzen.
*/

forall Position  $x^i$  from  $X$ 
     $Bel(Z_0 = x_i) = p(x^i)$ 
end for
k=1 // Zeitpunkt 1
/*
Hauptschleife: je nach Datenart(Odometrie oder Messung) wird
Positionsschätzung anhand der neuen Daten vorgenommen
*/
while(true)
    get(data)
    // Bewegungsmodell
    if data is odometry data  $o_k$ 
        forall Position  $x^i$  from  $X$ 
             $Bel(Z_k = x_i) \leftarrow 0$ 
            forall Position  $x_j$  from  $X$ 
                 $Bel(Z_k = x_i) \leftarrow Bel(Z_k = x_i) + Motion\_Model(x^i, x^j, o_k) * Bel(Z_{k-1} = x_j)$ 
            end for
        end for
    end if

    // Sensormodell

```

Wahrscheinlichkeit $p(x_k^i | x_{k-1}^{i'}, o_{k-1})$ (vom Punkt $x_{k-1}^{i'}$ zum Punkt x_k^i mit o_{k-1} zu gelangen) unter Verwendung der kontinuierlichen Verteilungsfunktion $\Phi((x_k^i - \mu) \sum (x_k^i - \mu)^T)$ abzuschätzen, wurde aber aus den Gründen der Wiederverwendbarkeit der Betrachtungen im nächsten Abschnitt über die Monte-Carlo Methode aufgeschoben.

```

else if data is sensor data  $y_k$ 
     $\alpha_k \leftarrow 0$ 
    forall Position  $x^i$  from  $X$ 
         $Bel(Z_k = x_i) \leftarrow Sensor\_Model(x^i, y_k) * Bel(Z_{k-1} = x_i)$ 
         $\alpha_k \leftarrow \alpha_k + Bel(Z_k = x_i)$ 
    end for

    // Normierung
    forall Position  $x^i$  from  $X$ 
         $Bel(Z_k = x_i) \leftarrow Bel(Z_k = x_i) / \alpha_k$ 
    end for
end if
 $k = k + 1$ 
end while

```

In der Methode fällt zunächst der quadratische Aufwand der Zustandsschätzung anhand der Odometriedaten auf, aber auch die a-posteriori Schätzung mit dem Sensormodell ist nicht zu vernachlässigen, denn die Routine $Sensor_Model(x^i, y_k)$ ist meist wesentlich rechenintensiver als $Motion_Model(x^i, x^j, o_k)$. In der Regel erfolgt die Auswertung beider Methoden gleich häufig, denn die Odometriemessung und Sensormessung werden meist abwechselnd durchgeführt, wobei in jedem Schritt das gesamte Gitter neu berechnet wird. Aus diesen (Effizienz-) Gründen stand eine Implementierung der Markow'schen Lokalisierungsmethode in ihrer klassischen Form in unserem Projekt nicht zu Diskussion, sondern diente vielmehr als Grundlage für die probabilistische Variante dieser Herangehensweise, *sequentielle Monte-Carlo-Methode (SMC-Methode)*, auch *Partikel Filter* genannt (den von uns gewählte Vertreter der Familie bezeichnet man üblicherweise als *sampling importance resampling (SIR)*). Die Verbesserungsidee beruht auf der Feststellung, dass für die Tracking-Problemstellung Aktualisierung aller Gitterzellen eines gitterbasierten Lokalisierungsverfahrens überflüssig ist, falls die aktuelle Position des Roboters weitgehend bekannt ist. Die Berechnung des neuen Belief-Wertes kann man nur auf einige relevanten, „wichtigen“ Positionshypothesen beschränken, was den Bayes Filter für die Echtzeitanwendungen natürlich attraktiver macht.

3.4. Monte Carlo Lokalisierung(MCL)

Die Grundidee der SIR-MCL ist, Dichte der Positionsschätzung mithilfe einer gewichteten Stichprobe zu repräsentieren. Ein Element der Stichprobenmenge wird Partikel(engl. Particle) genannt und besteht aus einem Position-Gewicht-Tupel (x^i, w^i) , wobei das Gewicht w^i Relevanz der Position x^i für die Stichprobe angibt und mit der Aufenthaltswahrscheinlichkeit des Roboters im Zustand x^i zusammenhängt. Wenn geeignet normiert, entspricht das Gewicht w^i dem Belief-Wert $\text{Bel}(Z = x_i)$ in der Markow'schen Lokalisierung genau, allerdings bleibt hier der Zustandsraum X kontinuierlich. Somit wird die kontinuierliche Dichtefunktion numerisch approximiert, so gilt für den Erwartungswert der Positionsschätzung:

$$\int x_k p(x_k | y_0, \dots, y_k) dx_k \approx \sum_{i=1}^N \bar{w}^i x_k^i \quad (\text{F.3.4.1})$$

falls N -Anzahl der Partikel und w^i normiert sind. Analog kann man Momente höherer Ordnung bestimmen, wodurch unsere Verteilungsfunktion beliebig genau approximiert werden kann, falls die Anzahl der Partikel $N \rightarrow \infty$ ¹⁰. Während bei der Markow-Lokalisierung die Dichte des Gitters (dx, dy, dw) das entscheidende Parameter für die Genauigkeit der Schätzung ist, hängt die Sorgfalt der MCL von

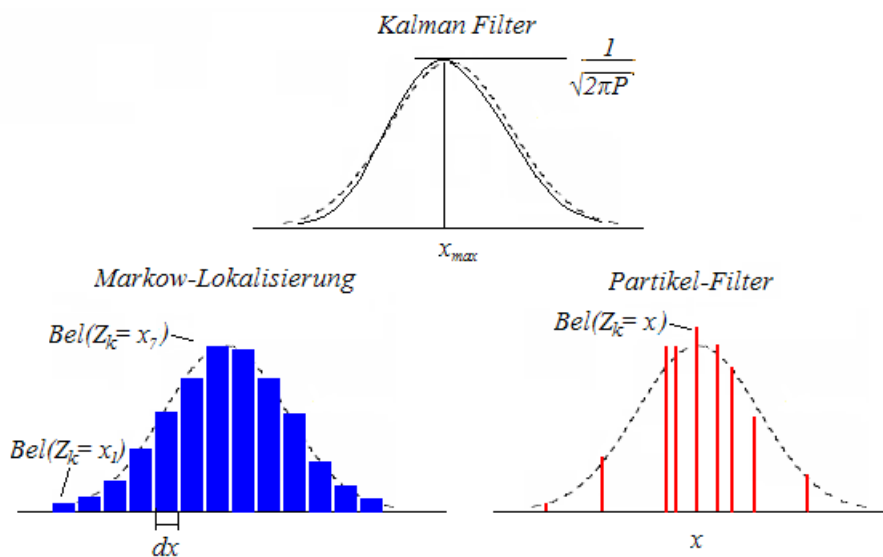


Abb. 3.4.1: Partikel-Filter: Dichtefunktions-Darstellungsvergleich

der Anzahl der Partikel ab. Allerdings steigt die Komplexität einer klassischen Implementierung der Markow-Lokalisierung mit der Verfeinerung des Gitters quadratisch, während die Komplexität der MCL, je nach Implementierung des Sampling-Verfahrens der Methode, sich auf $O(n)$ in der Anzahl der Partikel

¹⁰ Durch die Angabe aller nicht verschwindenden Momente ist eine Wahrscheinlichkeitsverteilung wohldefiniert.

reduzieren lässt (s. Abschnitt 4.5 „*Implementierung der Sampling-Methode*“). Außerdem wird die lokale „Genauigkeitskapazität“ der MCL nicht gleichmäßig auf dem ganzen Feld verteilt, sondern konzentriert in dem Bereich der größten Aufenthaltswahrscheinlichkeit. Schließlich lässt sich die MC-Methode so implementieren, dass die Anzahl der Partikel adaptiv, je nach der Gewissheit der Schätzung, gewählt wird. Zu den Nachteilen der MCL gegenüber der Markow-Lokalisierung gehört sicherlich seine ausgeprägte nicht-deterministische Natur und folglich kleinere Robustheit gegenüber unvorhersehbaren Ereignissen, wie die Bezeichnung „Monte-Carlo“ im Namen der Methode bereits andeutet. Demzufolge ist die MCL in ihrer klassischen Form schlechter für die Lösung des Kidnapped-Robot-Problems geeignet als ML, denn die Problemstellung dient ja gerade als Simulation größerer Verirrungen.

3.4.1 Ablauf

Der Filter verfügt über eine ähnliche Prädiktor-Korrektur-Struktur und basiert auf dem Bayes Filter, wie alle bisher betrachteten Verfahren.

3.4.2 Initialisierungsphase

In der *Initialisierungsphase* wird das Wissen über die Initialposition durch die Zusammensetzung der ursprünglichen Partikelmenge repräsentiert. Ähnlich wie im Falle der Markow'schen Lokalisierung, empfiehlt es sich mit der Gleichverteilung der Partikel, d.h. $\text{Bel}(Z_0 = x) = \{(x, 1/N)\}_N$, wo N -Anzahl der Partikel, anzufangen, falls keinerlei Wissen über die Anfangsposition zur Verfügung steht. Interessanterweise sollte Kenntnis der genauen Pose des Roboters nicht durch eine Multimenge, in der das entsprechende Element des Zustandsraums N -fach vorkommt, dargestellt werden. Vielmehr ist es darauf zu achten, dass bereits in der Initialisierungsphase, aber auch im späteren Verlauf der Berechnung, die Varianz der Partikel eine bestimmte Schwelle nicht unterschreitet, um die Degeneration des Algorithmus zu vermeiden, nämlich eine Situation in der alle Partikel bis auf ganz wenige ein Gewicht in der Nähe von 0 haben und damit praktisch keine Chance haben bei der nächsten Ziehung gewählt zu werden, was sich negativ auf die Genauigkeit der Schätzung auswirken würde. Aus diesem Grund werden Partikel der Initialmenge in der Nähe der bekannten genauen Position verstreut und entsprechend der Gaußverteilung gewichtet.

3.4.3 Prädiktion-Phase

Die Aufgabe der *Prädiktion-Phase* der MCL ist es, ähnlich wie im Falle der Markow-Lokalisierung, eine a-priori Positionsschätzung anhand der

Odometriedaten und des Bewegungsmodells zu geben. Allerdings wird dieses Problem hier ganz anders angegangen. Anstatt den Raum zu diskretisieren und alle Punkte dieses neuen Raums zu bewegen, werden nur die wahrscheinlichsten (und das ist der *springende* Punkt des Verfahrens) Hypothesen $\sim(w^i \sim p(x^i))$ ausgewählt und gemäß dem Bewegungsmodell (s. Abschnitt 3.3, F.3.3.7) aktualisiert. Genau an der Stelle entstehen die größten Unterschiede der beiden Methoden und auch die höhere Effizienz der MCL, die auf Kosten der Robustheit geht. Die Bewegungssimulation braucht nun zwar wesentlich weniger Operationen, da weniger Positionshypothesen betrachtet werden müssen, aber man kann sich nicht mehr darauf verlassen, dass sich auch richtige, d.h. aus einer bestimmten festen Umgebung der tatsächlichen Position stammende Partikel darunter befinden. Schlimmstenfalls muss man über mehrere Sampling-Generationen hinweg mit falschen Ergebnissen rechnen. Dank der nicht-deterministischen Verteilung der Punkte in Folge der Bewegung ($N(\mu, \Sigma)$), kann die Methode trotzdem gegen den richtigen Wert konvergieren, denn Partikel die in einem monotonen Bereich der Sensormodellfunktion liegen und sich näher zu der tatsächlichen Position befinden, werden von dem Sensormodell „schwerer“ gewichtet. Die Konvergenzgeschwindigkeit hängt von der Varianz der Fehlerverteilung, die andererseits auch die Gewissheit der Schätzung beeinflusst.

Die gesamte a-priori-Schätzung der MCL-Methode besteht aus folgenden Schritten:

- Eine Positionshypothese $x_{k-1} \sim \text{Bel}(Z_{k-1} = x)$ wird gemäß der Verteilung von Gewichten gezogen. (s. Unterabschnitt *Importance Sampling*)
- Die ausgewählte Positionshypothese wird entsprechend dem Bewegungsmodell aktualisiert $x_k \sim N(\mu(x_{k-1}, o_k), \Sigma(o_k))$ (s. Kapitel 3.3. Abschnitt *Bewegungsmodell*)
- Die Schritte werden wiederholt, bis man den gewünschten Stichprobenumfang erreicht hat.

Hier unterscheidet man zwischen adaptiven Methoden und Methoden, bei denen die Anzahl der Partikel immer gleich bleibt. Die ersteren nutzen ihr Wissen über die vermutete Komplexität der zu approximierenden Dichtefunktion und darüber wie gut die Approximation der früheren Schritte funktioniert hat. Beide diese Faktoren sind schwierig zu formalisieren, weswegen die entsprechenden Methoden nicht trivial sind¹¹. Wie auch immer sind adaptive Methoden nicht das Thema dieser Diplomarbeit, weil wir davon ausgehen, dass alleine schon aus den Speicherplatzgründen wir uns keine optimale Partikelmenge leisten können.

¹¹ Self Adaptive Particle Filter, Alvaro Soto

Die gemeinsame Verteilung der beiden Stichproben ist nach der Stichprobenziehung offensichtlich:

$$\{x_k, x_{k-1}\} \sim p(x_{k-1}) \cdot p(x_k | x_{k-1}, o_k) = \text{Bel}(Z_{k-1} = x) \cdot p(x_k | x_{k-1}, o_k)$$

und die Randverteilung von x_k :

$$x_k \sim \int \text{Bel}(Z_{k-1} = x) \cdot p(x_k | x_{k-1}, o_k) dx_{k-1} \quad (\text{F.3.4.2}),$$

was der F.3.3.3 (Kapitel 3.3) für die Bestimmung der a priori-Schätzung anhand der Odometriemessung beim Bayes-Filter entspricht. Für den diskreten Fall lässt sich (F.3.3.2) zu:

$$x_k^j \sim \sum_{i=1}^N w_{k-1}^i p(x_k^j | x_{k-1}^i, o_k) \quad (\text{F.3.4.3})$$

umformen.

3.4.4 Importance Sampling

Die Stichprobenziehungsroutine, *importance sampling* genannt, spielt eine große Rolle in dem MCL-Verfahren. Ihre Aufgabe besteht darin, Positionshypothesen aus der Partikelmenge entsprechend der Verteilung der Gewichte zufällig zu ziehen (ungeordnete Ziehung, mit Zurücklegen). Weil die Partikelmenge die Dichtefunktion der Positionsverteilung approximiert, werden die wahrscheinlichsten Positionshypothesen bevorzugt ausgewählt und weiter betrachtet. Dieses Prinzip ist eines der wichtigsten in der Theorie der evolutionären Algorithmen und unter dem Namen „survival of the fittest“-Prinzip bekannt. Es ist außerordentlich wichtig, dass die Auswahl „nicht-deterministisch“ erfolgt, damit auch Partikel mit kleinen Gewichten eine Chance haben, zu „überleben“. Obwohl man i.A. annimmt, dass eine Positionshypothese, deren Gewicht klein ist, weit von der tatsächlichen Position entfernt ist, kann sie im weiteren Verlauf der Berechnung fitte „Nachkommen“ produzieren, weil sie trotz der insgesamt falschen Schätzung richtige Komponenten enthalten könnte. Außerdem muss man mögliche Irrtümer in der Gewichtsverteilung aufgrund von immer präsenten Messfehlern einkalkulieren.

Es ist wichtig sich die Unterschiede zwischen dem Importance Sampling und der „herkömmlichen“ MC-Simulation vor Augen zu führen. Das Ziel dabei ist in den beiden Fällen den Erwartungswert einer Zufallsvariable X oder allgemeiner einer Funktion auf X abzuschätzen. Die klassischen Verfahren verwenden dafür den Standardschätzer:

$$\hat{E}_n(g(X)) = \int g(x) dP_n = 1/n \sum_{i=1}^n g(X_n)$$

wobei $X_n \sim P$ eine Stichprobe mit n Elementen ist. Nun sind manche Werte der Zufallsvariable für uns „wichtiger“ als andere (Positionen mit großen Aufenthaltswahrscheinlichkeiten), also möchte man, dass diese Werte häufiger in unserer Stichprobe vorkommen, ohne dass dadurch die Schätzung beeinträchtigt wird. Deswegen werden die Stichproben nicht gemäß der tatsächlichen Verteilungsfunktion p , sondern gemäß einer anderen Verteilungsfunktion (Stichprobenplan) q gezogen, wodurch „interessante“ Posen bevorzugt ausgewählt werden. Die Gewichtung der Stichprobenelemente sorgt dafür, dass der Schätzer trotz der „gefälschten“ Stichprobe erwartungstreu bleibt.

$$\begin{aligned} \hat{E}_n(g(X)) &= \int g(x) dP_n = \int g(x) p(x) dx = \int g(x) \frac{p(x)}{q(x)} q(x) dx \\ &= 1/n \sum_{i=1}^n g(x_i) \frac{p(x_i)}{q(x_i)} = \sum_{i=1}^n g(x_i) w(x_i), \end{aligned}$$

wo w_i –ein normierter Likelihood-Quotient ist, der in unserem Fall von dem (normierten) Gewicht der Partikel repräsentiert wird. Mathematisch gesehen hat dieser Schätzer eine kleinere Varianz als der Standardschätzer, vorausgesetzt die Sampling-Verteilung q wurde richtig gewählt, was nun die größte Schwierigkeit bei der Anwendung der Methode ist. Mehr dazu ist im Unterabschnitt 3.4.5 Update-Phase zu lesen.

3.4.5 Update-Phase

Am Ende der Prädiktion-Phase haben wir eine a-priori-Positionsschätzung (im SIR-Kontext auch *proposal distribution* genannt) in Form einer Partikelmenge, wobei alle Gewichte nach dem Resampling wieder auf N^{-1} gesetzt (oder als solche gedacht, da praktisch irrelevant) werden. Die alten Gewichte werden einfach verworfen. Analog zu F.3.4.1 können wir den Erwartungswert der Position mit:

$$\int x_k p(x_k | y_0, \dots, y_k) dx_k \approx? \sum_{i=1}^N N^{-1} x_k^i = N^{-1} \sum_{i=1}^N x_k^i$$

schätzen. Weil wir allerdings i.A. die echte Verteilung $p(x_k | y_0, \dots, y_k)$ nicht beobachten können und die Samples aus der Approximation der letzten Schritte generieren, ist dieser Schätzer nicht erwartungstreu. Wie im letzten Abschnitt bereits angesprochen, handelt es sich bei den Partikelgewichten w^i um Likelihood-Quotienten, die die Verzerrung korrigieren sollen. Es wäre naheliegend anzunehmen, dass das Sensormodell, das für einen gegebenen Zustand x_k^i und eine Messung y_k die Wahrscheinlichkeit $p(y_k | x_k^i)$ angibt, als *importance weight*

angesehen werden kann. Tatsächlich entspricht die Funktion $p(y_k | x_k^i)$ des Sensormodells der *Likelihood-Funktion* in der Theorie von *Importance sampling*¹². Sei $p(x) \sim \alpha p(y_k | x_k^i) \cdot \text{Bel}(Z_{k-1} = x) \cdot p(x_k | x_{k-1}, o_k)$ die zu bestimmende posteriori Dichtefunktion unserer Verteilung (vgl. Bayes Filter) und $q(x) \sim \text{Bel}(Z_{k-1} = x) \cdot p(x_k | x_{k-1}, o_k)$ ihre a-priori Schätzung (*proposal distribution*). Die Gewichtsfunktion ist dann gleich:

$$w(x_k^i) = p(x_k^i) / q(x_k^i) \sim \frac{\alpha p(y_k | x_k^i) \cdot \text{Bel}(Z_{k-1} = x^i) \cdot p(x_k^i | x_{k-1}^i, o_k)}{\text{Bel}(Z_{k-1} = x^i) \cdot p(x_k^i | x_{k-1}^i, o_k)} = \alpha p(y_k | x_k^i) \quad (\text{F. 3.4.4})$$

Demzufolge ist der Aktualisierungsschritt der MCL äquivalent zu dem der Markow-Lokalisierung und besteht aus der Berechnung der Wahrscheinlichkeit $p(y_k | x_k^i)$ Mithilfe des Sensormodells (s. Kapitel 3.3. Abschnitt *Sensormodell*) und anschließenden Normierung der Gewichte. Sei $w(x_k^i) = p(x_k^i) / q(x_k^i)$ der Likelihoodquotient. Dann gilt:

$$E(x_k) = \int x_k \cdot p(x_k) dx_k = \frac{\int x_k \cdot w(x_k) \cdot q(x_k) dx_k}{\int w(x_k) \cdot q(x_k) dx_k} \approx \frac{1/N \sum_{i=1}^N w_k^i x_k^i}{1/N \sum_{i=1}^N w_k^i} = \sum_{i=1}^N \bar{w}_k^i x_k^i \quad (\text{vgl. F.3.4.1})$$

wo \bar{w}_k^i -normiertes Gewicht des i-ten Partikels zum Zeitpunkt k. Somit lässt sich die Position mittels F.3.4.1 erwartungstreu schätzen.

3.4.6 Degenerationsproblem

Eines der größten Anwendungsprobleme des MCL ist das Degenerationsproblem. Dabei „konvergiert“ die Methode, d.h. die große Mehrheit der Partikel hat ein Gewicht in der Nähe von 0 und von einigen wenigen Partikeln dominiert. Im nächsten Schritt würden die meisten Partikel einer solchen Menge verworfen werden, was problematisch aus der „evolutionären“ Sicht der genetischen Algorithmen ist. Die beschriebene Situation lässt sich erkennen, indem man die „effektive“ Anzahl der Partikel gemäß der Formel:

$$N_{\text{eff}} = 1 / \sum_{i=1}^N (w_k^{(i)})^2 \quad (\text{F. 3.4.5})$$

berechnet¹³. Sobald N_{eff} einen bestimmten Schwellwert unterschreitet, liegt das Degenerationsproblem vor und die Gewichtsverteilung wird so modifiziert, dass die Gefahr der Degeneration nicht mehr besteht, indem man beispielsweise die Gleichverteilung erzwingt. Kurioserweise hat man an der Stelle umso mehr

¹² A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking, Arulampalam, S., Maskell, S., Gordon, N., Clapp, T.

¹³ Kitagawa, G., Monte Carlo filter and smoother for non-Gaussian nonlinear state space

Probleme, je präziser die Messung ist, wobei im hypothetischen Fall einer völlig rauschfreien Messung MCL zum Scheitern verurteilt ist.

3.4.7. SIR-MCL: Zusammenfassung

Pseudocode der MCL:

//Initialisierung:

//Gleichverteilung, falls Initialposition unbekannt, Normalverteilung, falls bekannt

forall Particles (x_0^i, w_0^i) from X_0

$$w_0^i = p(x^i)$$

end for

$k=1$

while(true) //Hauptschleife

for $i=1..N$

 //Prädiktion

$x_{k-1}^i = \text{draw_Sample}(X_{k-1});$

$x_k^i = \text{Motion_Modell}(x_{k-1}^i, o_k);$

 //Update

$w_k^i = \text{Sensor_Modell}(x_k^i, y_k);$

$X_k = X_k \cup \{x_k^i, w_k^i\}$

$\alpha_k \leftarrow \alpha_k + w_k^i$ //Normalisierungsterm

end forall

forall Particles (x_k^i, w_k^i) from X_k

$w_k^i = w_k^i * \alpha_k^{-1}$ //Normalisierung

end forall

$k=k+1$

end while

Output:

$$\sum_{i=0}^N w_k^i x_k^i$$

Mit MCL steht einem ein mächtiges Lokalisierungsverfahren zur Verfügung, das eine Reihe positiver Eigenschaften besitzt:

- Dichtefunktion der Positionsverteilung wird stichprobenbasiert mittels:

$$p(x \mid y_0, o_0, \dots, o_k, y_k) \approx \sum_{i=0}^N w_k^i \delta(x - x_k^i) \quad (\text{F.3.4.6})$$

geschätzt, wobei die Genauigkeit der Schätzung flexibel über die Anzahl der Partikel N gesteuert werden kann. Der Algorithmus konzentriert sich an „interessanten“ Stellen der Funktion, d.h. an Positionen mit der

größten Aufenthaltswahrscheinlichkeit. Dadurch erzielt man eine kleinere Varianz der Schätzung gegenüber Markow'schen Lokalisierung bei der gleichen. Diese Eigenschaft des Algorithmus war für uns aufgrund der Ressourceneinschränkungen von entscheidender Bedeutung.

- die Methode erlaubt Approximation von multimodalen Verteilungsfunktionen, was für die Problemstellung der globalen Selbstlokalisierung von Vorteil ist
- dank der parameterfreien Natur von MCL ist man außerordentlich flexibel bei den Modellierungsfragen wie denen der Systemdynamik, Messungsfehler oder Gütefunktion der Schätzung
- der Algorithmus ist intuitiv und einfach zu implementieren

Zu den *Nachteilen* des Algorithmus zählen in erster Linie:

- Stichprobenbasierte Repräsentation der Dichtefunktion ist relativ speicheraufwändig, wobei die Genauigkeit und die Stabilität der MCL mit der Anzahl der Partikel und somit dem Speicherverbrauch ansteigt.
- Defizite bei der Lösung des Kidnapped-Robot-Problems
- das Degenerationsproblem

Heutzutage gehört MCL zu den beliebtesten Zustandsschätzungsmethoden und es wurden zahlreiche Verbesserungsvorschläge zur Lösung der genannten Probleme gemacht. Zu erwähnen sind an dieser Stelle die *Mixture-MCL*¹⁴, *Auxiliary particle filter*¹⁵ oder *stratified resampling*-Methode. Allerdings sind diese Verfeinerungen mit einer gewissen Steigerung der Laufzeit verbunden, wodurch man wieder bei der Ressourcen-Problematik ist. Schließlich würde jeder zusätzliche Durchlauf der Partikelmenge unserem 16 MHz starken ATMEL Prozessor schwer ins Gewicht fallen, während wir ohnehin von Anfang an mit erheblichen Laufzeitproblemen rechneten. Letztendlich haben wir uns für das erste gegen die anerkannten Erweiterungen der MCL auf Kosten der Laufzeit entschieden und haben die Probleme mit eigenständig produzierten und somit im hohen Masse adaptierten und effizienten (Not-)Lösungen überwunden.

¹⁴ Robust Monte Carlo Localization for Mobile Robots, Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert

¹⁵ Filtering via Simulation: Auxiliary Particle Filters, Michael K. PITT and Nen SHEPHARD

Kapitel 4

Implementierung einer Monte-Carlo-Selbstlokalisierungsmethode

4.1 Einführung

Als erstes gilt es die Frage zu klären, welche der beiden betrachteten Lokalisierungsmethoden besser zu unseren Rahmenbedingungen passt: Kalman Filter(bzw. MHKF) oder die Monte-Carlo-Methode(SIR-MCL). Fassen wir die Vor- und Nachteile der beiden Methoden hinsichtlich der im Kapitel 2 besprochenen Anforderungen zusammen:

	Vorteile	Nachteile
Kalman Filter (s. 3.2.3)	<ul style="list-style-type: none">⇒ speicherschonend⇒ Tracking optimal¹⁶	<ul style="list-style-type: none">⇒ Globale Selbstlokalisierung problematisch⇒ Modellierungseinschränkungen⇒ Vergleichsweise zeitaufwändig⇒ Relativ schwierige Implementierung (MHKF-Fall)
MCL (s. 3.4)	<ul style="list-style-type: none">⇒ Globale Selbstlokalisierung und Tracking möglich⇒ Flexible Regelung der Effizienz-Genauigkeit-Relation durch Partikelanzahl⇒ Flexible Gestaltung des Modells	<ul style="list-style-type: none">⇒ Speicheraufwändige Repräsentation der Dichte

Tabelle 4.1: Vergleich MCL-MHKF

In den oben aufgeführten Punkten unterscheiden sich die Verfahren voneinander. Daneben gibt es einige weitere wichtige Problemstellen, die die beiden Methoden gemeinsam haben, darunter z.B. die mangelhafte Eignung zur Lösung des Kidnapped-robot-Problems und somit zur Berichtigung größerer Fehleinschätzungen. Dieser Makel sollte im Hinterkopf behalten werden, bis man

¹⁶ Erweiterter KF ist nicht optimal

sich der Problematik nach der Feststellung des Ausmaßes der Auswirkungen widmen kann.

Nach der Abwägung der beiden Alternativen entschieden wir uns für die SIR-Monte-Carlo-Methode, die vor allem durch ihre Flexibilität und Adaptierbarkeit überzeugte. Allerdings blieb die Frage, ob der uns zu Verfügung stehende Speicherplatz ausreichen bzw. erschwingliche Partikelzahl die geforderte Robustheit der Selbstlokalisierung gestatten würde zunächst einmal offen, da der Einfluss der Partikelzahl auf die Stabilität des Algorithmus sich nur schwer formalisieren lässt. Für die Selbstlokalisierung sind maximal 500 Bytes reserviert, was in etwa 1/8 des gesamten RAMs ausmacht. Um die Realisierbarkeit des Vorhabens festzustellen, muss man in dem Fall die Methode implementieren und ausführlichen Tests unterziehen. Aus vergleichbaren Arbeiten zu MCL in der RoboCup-Umgebung (z.B. in der Standard Platform League) wussten wir, dass etwa 100 Partikel die unterste Grenze für ein zuverlässiges Funktionieren der Methode sind und erst mit 200 ganz gute Ergebnisse erzielt wurden. Das würde allerdings bedeuten, dass man schätzungsweise $500/100=5$ bzw. 2-3 Bytes pro Partikel verwenden darf.

4.2 Programmstruktur

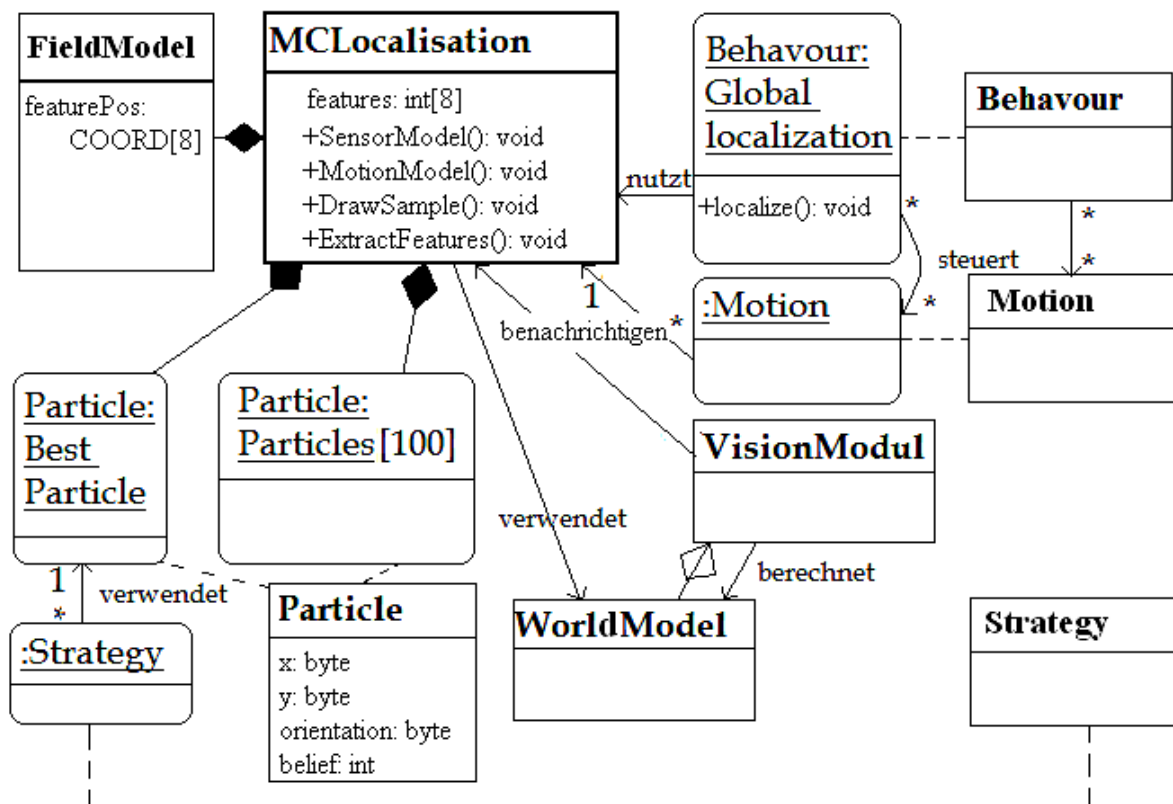


Abb. 4.2.1: Programmstruktur um die Selbstlokalisierung

Die Struktur der MCL-Implementierung lehnt sich im Großen und Ganzen an den im Abschnitt 3.4.8 präsentierten Pseudo-Code der Methode an. Ihre drei wichtigsten Bestandteile sind die Implementierungen des Sensor- und Bewegungsmodells, sowie die Sampling-Funktion. Ein Überblick zu den Implementierungen gibt es in den Abschnitten 4.3 4.4 und 4.5. Sie alle greifen auf das global definierte Array mit den Partikelstrukturen (Abschnitt 4.2.1) zu, das die Partikelmenge repräsentiert. Das Ergebnis der Schätzung samt der Gewissheit wird in einem speziellen Partikel namens *BestParticle* abgelegt. Etwas komplizierter ist die Einbindung des Moduls in den Gesamtkontext des Steuerungsprogramms. Wie in dem Kapitel 2 erläutert, zeichnet sich die Selbstlokalisierung durch eine enge Verflochtenheit mit praktisch allen anderen Programmmodulen aus. Zur Lösung der beiden wichtigsten Problemstellungen der Selbstlokalisierung (die globale und die lokale L. s.3.1) werden dieselben Methoden verwendet, der Einsatzkontext unterscheidet sich jedoch grundlegend. Die Implementierungsdetails dazu sind in den Unterabschnitten 4.2.2 und 4.2.3 zu finden.

4.2.1 Die *Particle*-Struktur

Als erstes muss die Partikel-Grundstruktur definiert werden, die angesichts der Speicherknappheit von entscheidender Bedeutung ist. Ein Partikel ist bekanntlich ein Zweier-Tupel, bestehend aus einem Element des Zustandsraums und dem dazugehörigen Belief-Wert, der die Aufenthaltswahrscheinlichkeit repräsentiert. Der Zustandsraum ist dreidimensional, seine (diskretisierten) Zustandsgrößen sind: die beiden kartesischen Koordinaten x und y in cm und der Orientierungswinkel in Grad. Die Abbildung 4.2.1 zeigt das gewählte

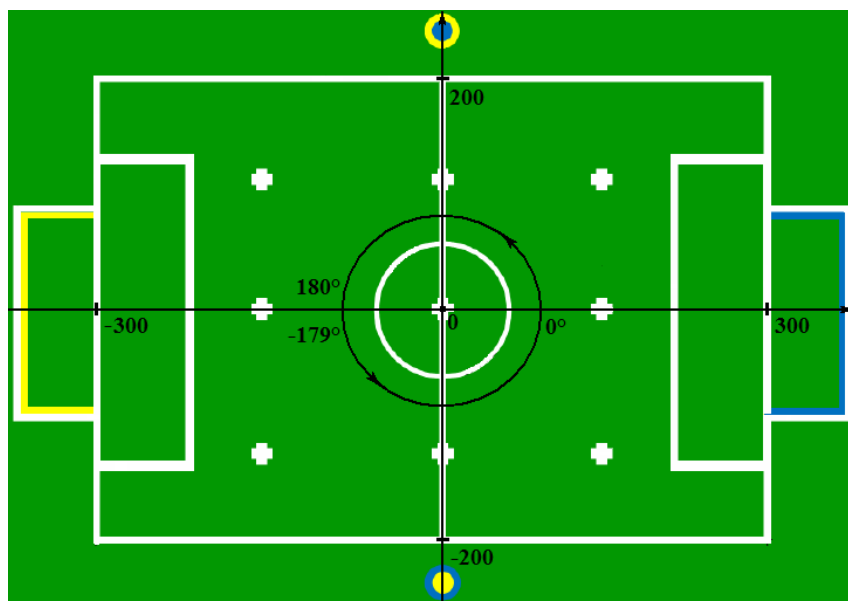


Abb. 4.2.2: Koordinatensystem

Koordinatensystem. Es wurde so definiert, dass der in der Mitte des Feldes stehende und frontal in die Richtung des blauen Tores schauende Roboter, sich im Zustand $(0,0,0)^T$ befindet. Leider passt keine der drei Größen in den Wertebereich eines Bytes, selbst wenn man annimmt, dass nur Werte von innerhalb des Feldes relevant sind. Das bedeutet, dass alleine für die Speicherung von dem Zustandsterm des Tupels 6 Bytes pro Partikel fällig sind, wobei 7 von 8 Bits des jeweils zweiten Bytes ungenutzt blieben. Alternativ kann man entweder die einzelnen Bytes mehrfach (bitweise) nutzen oder die Werte skalieren, wobei Skalierung eine größere Speichersparnis, aber auch Genauigkeitsverluste mit sich bringt. Später wurde entschieden, die Werte zu skalieren, zum einen, weil es sich herausgestellt hat, dass die 1cm-Auflösung grundsätzlich zu fein ist, da der durchschnittliche Lokalisierungsfehler größer ist und zum anderen aus implementierungs-technischen Gründen.

Der Belief-Wert, der eigentlich dem MCL-Modell entsprechend eine reelle Zahl aus dem Intervall $[0,1]$ sein soll, wurde ursprünglich als Fließkommazahl implementiert. Weil das allerdings, angesichts des fehlenden FPU, zu regelrechten Leistungseinbrüchen geführt hat (s. 5.3 Tests), wurde die Float-Variable durch eine Festkommazahl ersetzt. Aus Implementierungsgründen (s. Abschnitt 4.5) werden 2 Bytes pro Partikelstruktur für das Feld reserviert.

Aufaddiert ergibt das $1+1+1+2=5$ Bytes pro Partikel und folglich 500 Bytes pro 100 Partikel. Damit bleibt man innerhalb der vorgegebenen Grenzen, was den Speicherverbrauch angeht.

4.2.2 Globale Selbstlokalisierung

Die globale Selbstlokalisierung erfordert von dem Roboter die Fähigkeit seine Position nur anhand der Kameramessungen und (im Allgemeinen) ohne jegliches Vorwissen zu bestimmen. Diese Situation tritt nur nach Spielunterbrechungen, sowie in Folge gravierender Fehler auf. In unserem Programm wird globale Selbstlokalisierung als selbstständiges Verhalten realisiert, dessen Aufgabe es ist, die für die Selbstlokalisierung notwendige Menge an Daten (mindestens zwei

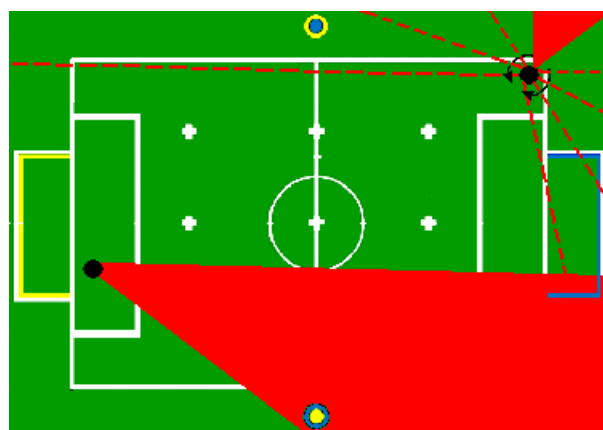


Abb. 4.2.3: Der beste und der schlechteste Fall für die globale Selbstlokalisierung

Landmarken oder Teile davon s.4.3) zu erheben und unter Verwendung des Sensormodells die Position des Roboters zu bestimmen. Dafür müssen in den meisten Fällen Kopfdrehungen (s. Abb. 4.2.3) ausgeführt werden, was dazu führt, dass die Lokalisierungsdauer bis in den Sekundenbereich reicht. Die Ursache für die Verzögerung ist die Wartezeit von etwa 200 ms bis die Kamera nach der Repositionierung zur Ruhe kommt, d.h. bis der Kopfmotor die vorgegebene Position einnimmt. Durch die schnellen Bewegungen der Kamera können nämlich die Kanten verschwimmen, was eine halbwegs zuverlässige Messung unmöglich macht. Der horizontale Öffnungswinkel der Kamera beträgt 40° und um einen 160° -breiten Bereich zu durchsuchen, muss man 4 Zwischenstopps machen. Mit der maximalen Zwischenstoppdauer von 250 ms erhält man die Gesamtzeit von 1s, wobei die meiste Zeit einfach verschwendet wird. Um die durchschnittliche Laufzeit der Methode zu verkürzen wurde ein Trick angewandt. Abhängig davon, wo man gerade steht, enthalten einige der aufgenommenen Bildern keinerlei nützliche Informationen, weil es sich keine relevanten Objekte im Bild befinden oder nur ein nicht weiter identifizierbarer Teil eines Objekts zu sehen ist. Um das festzustellen braucht man keine scharfen Aufnahmen, die 200 ms Wartezeit entfällt und die Kamera kann sofort um eine Position weiter gedreht werden. Erkennt man auf dem Bild brauchbare Objekte, so müssen Wartezeiten nach wie vor einkalkuliert werden. Allerdings reichen in der Regel 2 gute Aufnahmen, um sich einigermaßen zuverlässig lokalisieren zu können, was die Laufzeit von 500 ms ergibt(+ in etwa 50 ms pro ein „leeres“ Bild). Von außen sieht ein solches Verhalten intelligenter aus: die Roboter scheinen bestimmte Objekte anzuvisieren.

Je mehr relevante Objekte man in verschiedenen Bildern sieht, umso zuverlässiger und genauer ist das Ergebnis der Lokalisierung. Es ist gefährlich, die globale Selbstlokalisierung anhand nur eines einzigen Bildes durchzuführen, selbst wenn ausreichend viele Informationen im Bild zu sehen sind, denn die Messfehler eines Bildes sind naturgemäß stark korreliert. Die Zuverlässigkeit der globalen Selbstlokalisierung ist indessen insofern sehr wichtig, als, angesichts der eingeschränkten Eignung der Methode zur Lösung des Kidnapped-Robot-Problems, die in dieser Phase gemachte Fehleinschätzung sich nur sehr mühsam korrigieren lässt. Partikel, die nach der Erstlokalisierung in der Nähe der lokalen Maxima der Likelihood-Funktion landen, bleiben häufig über mehrere Iterationen hinweg dort gefangen. Werden sie durch zufällige Messfehler zusätzlich bestätigt, so sind Fehleinschätzungen der eigenen Position in gleich mehreren aufeinander folgenden Lokalisierungsversuchen häufig die Folge. Um die Zuverlässigkeit der Erstlokalisierung zu steigern, werden die dafür bestimmten „nicht-leeren“ Bilder mit Überlappung aufgenommen, sodass eine und dieselbe Messung zweifach durchgeführt wird. Die Messwerte werden über eine Durchschnittsbildung (Kalman Filter) kombiniert.

Bei der diesjährigen Feldkonstellation¹⁷ kommt es relativ selten vor, dass man gleich 3 Landmarken und somit ausreichend viele Daten für die Selbstlokalisierung in einem einzigen Bild sieht (s Abb. 4.2.2). Aus diesem Grund wird die Position der Objekte über mehrere Bilder hinweg in einem speziellen Array *features* gespeichert. Auf diese Weise kommen maximal 8 für die Selbstlokalisierung relevante Messwerte(s. 4.3) stufenweise zusammen, was eine bis zu 6-fache Ausführung der Selbstlokalisierungsroutine ermöglicht, wobei jeweils eine bestimmte Anzahl der zuletzt gesehenen Merkmale, typischerweise 3, berücksichtigt werden. Alternativ kann man bei jedem Aufruf der Sensor-Modell-Methode alle bis dahin durchgeführte Messungen in die Berechnung miteinbeziehen, was zwar die Robustheit der Methode gegenüber gelegentlichen Messfehlern erhöhen, jedoch auch die Laufzeit der globalen Selbstlokalisierung negativ beeinflussen würde. Klar ist, dass die Erstlokalisierung nicht erfolgen sollte, bis man mindestens 3 Messungen aus 2 verschiedenen Bildern zur Verfügung hat. Durch die mehrfache Wiederholung der Routine, wird vor allem die Genauigkeit der Schätzung erhöht, weil die Partikel dadurch dichter aneinander rücken(s. 3.4).

Nach dem Vollzug der globalen Selbstlokalisierung wird sichergestellt, dass die meisten Partikel sich an einem Maximum der Likelihood versammelt haben(s. 4.3.5), was für das anschließende Tracking sehr wichtig ist. Die Methode der globalen Selbstlokalisierung wird nach einem Positionsverlust im Laufe des Spieles aufgerufen, was anhand der massenhaften Tracking-Fehler festgestellt werden kann. Auch am Anfang des Spiels und in den out-of-play-Situationen könnte dies erforderlich sein, allerdings kann man in solchen Standardsituationen auf das explizite regelbasierte Wissen zugreifen und sich dadurch die zeitaufwendige Lokalisierungsprozedur sparen.

4.2.3 Lokale Selbstlokalisierung

Das Tracking erfordert eine ganz andere Herangehensweise als die globale Selbstlokalisierung. Die lokale Selbstlokalisierung ist kein eigenständiges Verhalten mehr und soll möglichst beiläufig stattfinden. Das größte Problem dabei ist der Mangel an verwertbaren Messungen, denn die Kamera wird nun von anderen Verhalten gesteuert und man keine Garantien bzgl. der Objektentdeckungshäufigkeit hat. An der Stelle kommt das Motion-Modell ins Spiel, mit dem die Roboterbewegung zwischen zwei aufeinanderfolgenden für die Selbstlokalisierungszwecke brauchbaren Bildern grob verfolgt werden kann. Wie

¹⁷ Die Anzahl der Polen wurde in diesem Jahr von 4 auf 2 reduziert.

bereits angesprochen, ist Odometriemessung bei Humanoiden extrem problematisch, sodass diese „blinde“ Objektverfolgung nur eine sehr begrenzte Zeit (in etwa 5 Schritte) sinnvoll ist. Danach muss sichergestellt werden, dass der Roboter endlich eine Landmarke sieht, mit der er seine Positionsschätzung verbessern kann und eine ausufernde Fehlerakkumulation verhindert. Aus diesem Grund wurde ein spezielles Verhalten entwickelt, das dafür sorgt, dass der Roboter seine aktuelle Tätigkeit unterbricht, eine Landmarke anvisiert, die Messung durchführt und dort weiter macht, wo er aufgehört hat. Das unterbrochene Verhalten darf nicht allzu sehr kritisch sein, was allerdings dadurch sichergestellt wird, dass nur die Laufschriffe gezählt werden und nicht die Schieß- oder Aufstehbewegungen. Das Anvisieren der Landmarken erfolgt gezielt, indem der Roboter die erwartete Position von der nächsten Landmarke anhand der Positionsschätzung und des Feldmodells bestimmt und die notwendige Kopfbewegung im voraus ausrechnet, wobei man die entsprechende Methode aus dem Sensormodul wiederverwendet (s.4.3.2). Das Verhalten kann sowohl im Laufen als auch stehend ausgeführt werden, was jedoch die Verwendung der Messungen aus verschiedenen Bildern wie bei der globalen Selbstlokalisierung unmöglich macht. Außerdem werden die Messfehler beim Laufen größer. All das führt dazu, dass die Streuung der Partikel beim Tracking zunimmt, weswegen es an der Stelle noch mal auf die Bedeutung der vorausgegangenen erfolgreichen globalen Selbstlokalisierung hingewiesen sei.

Die Tests haben ergeben, dass auf die beschriebene Prozedur nur selten zugegriffen wird. Bei den meisten Verhalten (vor allem Suchverhalten) bekommt der Roboter genügend Umgebungsinformationen mit, um sich lokalisieren zu können. Falls eine brauchbare Messung entdeckt wird, benachrichtigt die Vision-Routine sofort das Selbstlokalisierungsmodul, welches dann über die Notwendigkeit der erneuten Selbstlokalisierung entscheidet. Allzu sehr häufige Ausführung der Sensormodel-Routine wird angesichts der erheblichen Laufzeit vermieden, genauso wie bei solchen kritischen Verhalten wie das Schieß-Verhalten, die kein genaues Wissen über die eigene Position benötigen. Sobald die Lokalisierung erfolgt ist, wird der Zähler der blinden Schritte zurückgesetzt und man kann eine Zeitlang wieder „blind“ Laufen. Das in der Hinsicht problematischste Verhalten ist das Dribbel-Verhalten, bei dem der Roboter sich auf den Ball konzentriert und dabei kaum Landmarken zu sehen bekommt. Zugleich ist es beim Dribbeln sehr wichtig die eigene Position zu kennen, während man keine Zeit hat, um sich umzuschauen. Das Problem kann eventuell durch die Hinzunahme der Feldlinien gelöst werden.

4.3 Implementierung des Sensormodells

Wie bereits mehrfach angesprochen, stellt das Sensormodell das Kernstück der MCL dar. Insbesondere die Humanoiden sind vor allem auf die visuelle Sensorik bei der Selbstlokalisierung angewiesen, denn die Odometriedaten der zweibeinigen Roboter sind häufig ungenau und wenig aufschlussreich. Die globale Lokalisierung ist ausschließlich von dem Sensormodell abhängig, das Tracking-Verfahren steht und fällt mit der Genauigkeit des Modells. Die Implementierung des Modells war angesichts der besonderen Rahmenbedingungen der Humanoiden allgemein und der FHumanoiden (Stichwort Ressourcenknappheit) im Speziellen der zentrale Gegenstand dieser Diplomarbeit und es hat mehrere Anläufe gebraucht, bis wir mit dem Ergebnis zufrieden waren. Unsere erste Implementierung orientierte sich streng an der MCL-Theorie und enthielt keine wesentlichen Modifikationen im Ablauf des Verfahrens. Die folgende Beschreibung der ersten Implementierung gibt einen Überblick darüber, wie weit die Möglichkeiten der Methode reichen, welche Schwächen sie hat, hilft die Teilprobleme zu identifizieren und führt uns an die endgültige Lösung heran.

4.3.1 Überblick

Das Sensormodell ist aus der Sicht der SIR-Theorie eine Likelihood-Funktion, die, gegeben eine Messung y , für jeden Zustand x aus dem Zustandsraum des Systems die Wahrscheinlichkeit angibt, die Beobachtung y im Zustand x zu machen. In unserem Sensormodell werden *nur die x-Koordinaten* der Objekte im Bild als Messungen in die Berechnung einbezogen, somit verzichtet man genau auf die Hälfte der gesamten sensorischen Information. Der Grund dafür ist die extreme Abhängigkeit der Messungsgenauigkeit von der Körperneigung des Roboters (im Falle der y -Koordinate handelt es sich um eine nicht-lineare Abhängigkeit!) und während man die seitliche Neigung einigermaßen unter Kontrolle hat, ist die frontale Neigung der humanoiden Roboter, die vor allem einen Einfluss auf die Höhe der Objekte im Bild hat, sehr viel schwieriger zu überwachen. Das hat in erster Linie damit zu tun, dass die Breite der Stützfläche des Roboters mehr als doppelt so groß ist, wie die Länge. Außerdem haben 4 kaum bewegte Motoren einen Einfluss auf die seitliche Neigung des Roboters und 7 höchst aktive Motoren auf die frontale Neigung und folglich auf die y -Messung (s. Abb.4.3.1). Von allen senkrechten Objektgrenzen werden nur

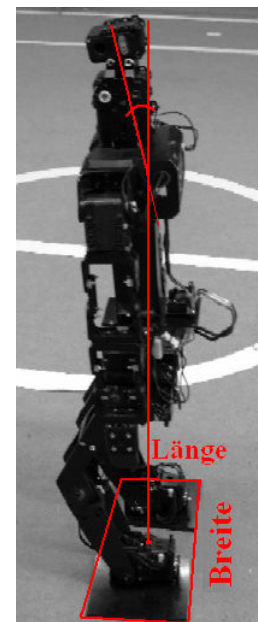


Abb. 4.3.1:
Neigungshandicap
der FHumanoiden

solche berücksichtigt, die nicht mit dem Bildrand „verklebt“ sind, was auf die Fortsetzung des Objektes hinter den Bildgrenzen hindeutet. Welche Bildinformation auch Messungen im Sinne der Selbstlokalisierung sind, wird von der merkmalsextrahierenden Funktion bestimmt. Auch die Wahl des geeigneten Zustandsraums ist nicht so selbstverständlich, wie es möglicherweise erscheint. Die naheliegende Lösung wäre sicherlich die durchgehende Verwendung eines und desselben Zustandsraums (x, y, ω) im gesamten Selbstlokalisierungsmodul. Allerdings mussten wir im zweiten Implementierungsversuch, angesichts der unzureichenden Größe der Partikelmenge auf eine Dimension des Raums verzichten (s. Abschnitt 4.3.7).

Die grundlegende Herangehensweise an das Problem wurde im Abschnitt 3.3.3 „Sensormodell“ erläutert. Wie man bereits festgestellt hat, lässt sich das Gesamtproblem in kleinere Teilprobleme zerlegen kann, bei denen nur jeweils ein Element des Beobachtungsvektors berücksichtigt wird. Das Ergebnis der Berechnungen sind die Wahrscheinlichkeiten für die einzelnen Messungen. Geht man davon aus, dass es sich bei den Messungen um stochastisch unabhängige Ereignisse handelt, so lassen sich die Teillösungen durchs Aufmultiplizieren kombinieren.

Gegeben sei ein beliebiger fester Zustand x aus dem Zustandsraum und ein Element des Beobachtungsvektors y_i . Dann kann die Wahrscheinlichkeit $p(y_i|x)$ folgendermaßen berechnet werden:

1. Anhand eines Umgebungsmodells wird der erwartete Wert der Messung y_i' im Zustand x bestimmt.
2. Die Differenz $y_i' - y_i$ zwischen der erwarteten und tatsächlich gemessenen Größen wird berechnet.
3. Anhand eines Fehlermodells wird die Wahrscheinlichkeit für die zufällige fehlerbedingte Abweichung $y_i' - y_i$ ermittelt.

Es ist einfach zu sehen, dass die gefundene Lösung der gesuchten Wahrscheinlichkeit $p(y_i|x)$ entspricht. Geht man z. B. von einer fehlerfreien Messung aus, so ist die Wahrscheinlichkeit für die Beobachtung y_i im Zustand x ist genau dann gleich 1, wenn der erwartete Wert y_i' mit dem gemessenen y_i übereinstimmt. Misst man etwas anderes als y_i' , so ist es ausgeschlossen, dass man sich zur Zeit der Messung im Zustand x befand und die Wahrscheinlichkeit $p(y_i|x)$ ist gleich 0. Im folgenden werden die einzelnen Schritte genauer beschrieben.

4.3.2 Punkt 1: Feldmodell

Der zeitintensivste Teil der Berechnung ist die Bestimmung der zu erwartenden fehlerfreien Messung y_i' anhand eines Umgebungsmodells mithilfe der Triangulation. Das Umgebungsmodell enthält die Koordinaten der 4 Torpfosten und 2 Säulenzentren im gewählten Koordinatensystem. Das Modell ist somit im wesentlichen metrisch, allerdings spielt die Reihenfolge der Objekte für die Methode eine wichtige Rolle, was man als ein topologischer Ansatz ansehen kann.

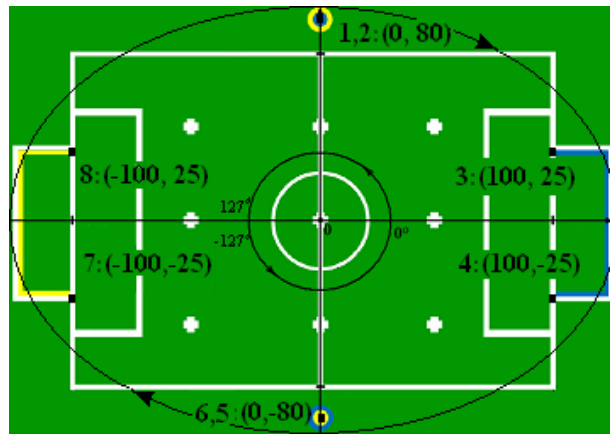


Abb.: 4.3.2: Feldmodell mit der 3-fachen Skalierung der Koordinaten und 360/256-fachen Skalierung der Orientierung

Die Reihenfolge entspricht der Anordnung von den Objekten, wenn man das Feld im Uhrzeigersinn betrachtet (s. Abb. 4.3.2). Auf diese Weise erreicht man, dass das Torminimum (die linke Seite des Tores in einem Kamerabild) immer vor dem Maximum kommt. Für die runden Säulen kann man die Maxima/Minima natürlich nicht ohne weiteres angeben, denn welche der Säulenpixel am Rande liegen, hängt von dem jeweiligen Sichtwinkel des Betrachters ab. Nicht desto trotz werden die Säulen von jeweils 2 Punkten repräsentiert, um die Min-Max-Reihenfolge nicht zu stören.

Die Berechnung der erwarteten Position der Objekte erfolgt in 3

Schritten:

1. Bestimmung der in dem gegebenen Zustand erkennbaren Randpunkte des Objekts. Entfällt im Falle der Tore, da die Punkte, dank der rechteckigen Form der Tore, immer an den Pfosten liegen.

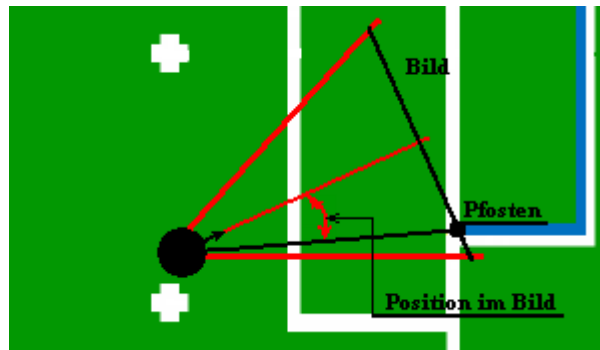


Abb. 4.3.3a: Gesuchte Position im Bild

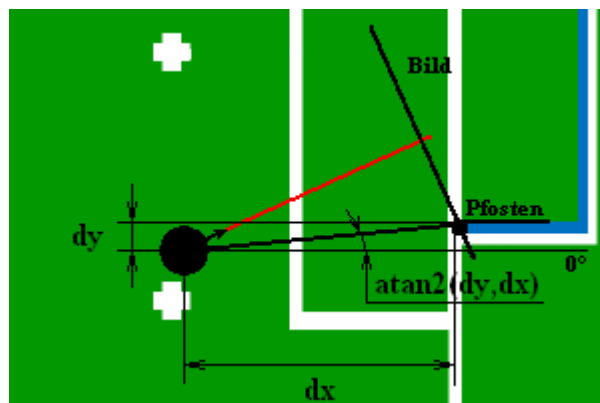


Abb. 4.3.3b: Schritt 2

2. Berechnung des Winkels, unter dem die Randpunkte der Landmarke von der gegebenen Position aus zu sehen ist, in dem globalen Koordinatensystem. (s. Abb. 4.3.3b)
3. Transformation in das lokale Koordinatensystem des Roboters. Der gefundene Winkel entspricht der erwarteten Position der Objektkanten im Bild, allerdings in Gradeinheiten. (s. Abb. 4.3.3c)

Die Berechnung der Tormaxima/ -minima erfolgt direkt anhand der im Feldmodell gespeicherten Koordinaten der Torpfosten. Die Abbildungen 4.3.3a-c schildern den Ablauf der Prozedur. Problematisch an der Stelle ist die Verwendung von $atan2$ Funktion mit ihren Fließkommparametern.

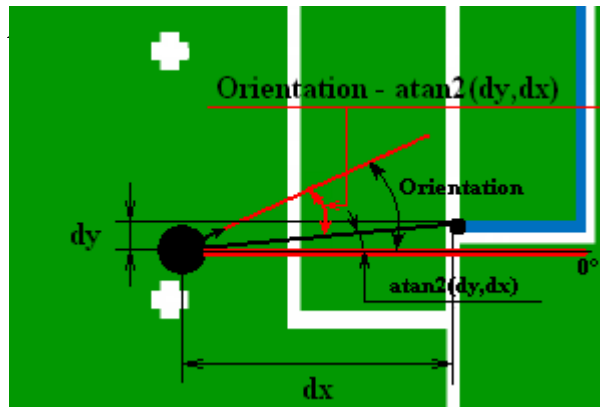


Abb. 4.3.3c: Schritt 3

Die Funktion wird bis zu $8 \cdot 100$ Mal pro einen Aufruf der Sensormodell-Routine ausgeführt. Die Folgen für die Laufzeit sind dramatisch und werden im Kapitel *Tests* aufgeführt.

Bei der Lokalisierung der Randpunkte von den beiden Säulen muss man bedenken, dass, angesichts der runden Form dieser Landmarken, die Position der Kanten, die man sieht, von dem Blickwinkel des Beobachters abhängt. Deswegen erfolgt die Positionsbestimmung der Säulen in 2 Schritten: als erstes wird die zu erwartende Position des Mittelpunktes der Polen mit dem oben beschriebenen Triangulationsverfahren bestimmt,

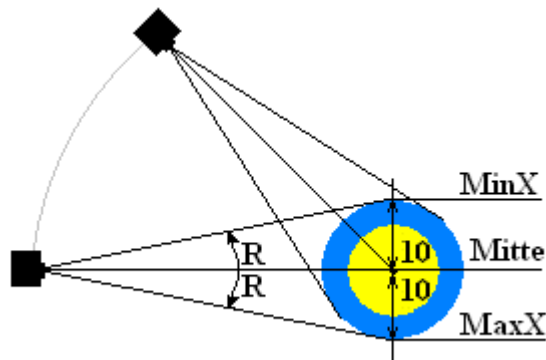


Abb.4.3.4: Positionsbestimmung der Säulen

danach wird der sichtbare Radius der Säulen ausgerechnet und rechts/links mit dem Mittelpunkt addiert/subtrahiert. Die Abbildung 4.3.4 verdeutlicht das Vorgehen. Die Berechnung des Radius erfordert einen weiteren Aufruf der Arkustangens-Methode, sowie die Berechnung des euklidischen Abstandes von der Partikelposition zum Mittelpunkt der Säule.

Die Berechnung erfolgt nur für die Landmarken, die im aktuellen Bild identifiziert wurden. Leider ließ es sich nicht vermeiden, dass für die infolge der vorsätzlichen Bildüberlappung bei der Prozedur der globalen Selbstlokalisierung mehrfach erkannten Objekte die Lokalisierungsroutine wiederholt werden muss, denn die Speicherung der gefundenen Lösungen würde $100 \cdot \text{Anzahl der Objekte Bytes}$ Speicherplatz benötigen.

4.3.3 Punkt 2: Differenzbestimmung

In diesem Schritt erfolgt die Berechnung der Abweichung zwischen den gemessenen Positionen der Landmarken und ihren im vorangegangenen Schritt bestimmten erwarteten Positionen. Spätestens an dieser Stelle ist eine eindeutige Zuordnung der gefundenen Landmarken den entsprechenden Modellkomponenten notwendig. Wie bereits angesprochen, liefert die Bildverarbeitungsroutine aus Effizienzgründen keine expliziten Informationen zur Anzahl und Eigenschaften der im Bild vorhandenen Säulen. Diese Aufgabe wird von der merkmalsextrahierenden Funktion des Moduls übernommen. Dabei macht man sich die bekannte Reihenfolge der Landmarken zunutze. Ist die relative Position eines der Tore bekannt, so kann man ggf. unter Zuhilfenahme der Odometriedaten auf die Identität der gesehenen Säule schließen. Ist das nicht der Fall, so wird die Position der Landmarke gespeichert und nach einem der beiden Tore gesucht, wobei die unterdessen gemachte Orientierungsänderungen aufgezeichnet werden, damit man anschließend bestimmen kann, ob sich die Säule rechts oder links von dem Tor befindet. Die beschriebenen Identifizierungsmaßnahmen sind in aller Regel nur im Falle der globalen Selbstlokalisierung notwendig, nämlich wenn überhaupt kein Vorwissen über die gegenwärtige Ausrichtung des Roboters bekannt ist, denn schon die größten Orientierungsschätzungen reichen meist zur Unterscheidung zwischen den beiden Säulen aus. Ein Mehraufwand ist es auch im Falle der globalen Lokalisierung nicht, da die Erstlokalisierung nicht vor der Identifizierung von mindestens 3 Landmarken (z.B. Min/Max einer Säule + Min eines Tores) erfolgt.

Bevor die Differenz zwischen den gemessenen und den berechneten Werten ermittelt werden kann, muss die Anpassung der Maßeinheiten stattfinden. Das Problematische an dem elementaren Vorgang ist die Umrechnung von den Pixelwerten der Kameramessung in die Gradeinheiten anhand eines spezifischen Umrechnungsfaktors, dessen Ermittlung experimentell erfolgen muss. Dabei wird eine mögliche Verzerrung des Bildes vernachlässigt und die Transformation erfolgt linear, was jedoch keine großen Konsequenzen angesichts der guten optischen Eigenschaften der Kamera hat. Viel problematischer ist die Tatsache, dass die Öffnungswinkel von verschiedenen Kameras (alle von demselben

Modell) sich zum Teil dramatisch unterscheiden. Die Werte variieren zwischen 42 und 48° was ca. 15% des gesamten Sichtfelds ausmacht. Das entspricht ungefähr dem daraus resultierenden zusätzlichen Selbstlokalisierungsfehler. Aus diesem Grund muss die Ermittlung des Umrechnungsfaktors für jede neue Kamera gesondert erfolgen, worauf der Wert im kameraeigenen Register gespeichert wird. Damit die Konvertierung trotz der Ganzzahlrechnung möglichst verlustlos abläuft, werden alle Winkel in Viertel-Grad ($160 \text{ Pixel} / 40^\circ$)-Einheiten dargestellt.

4.3.4 Punkt 3: Messfehlermodell

Bei der Fehlermodellierung gingen wir von der Normalverteilung der Fehler aus, was sicherlich eine naheliegende Lösung für diese Sensorart ist. Der produzierte Fehler ist das Ergebnis vieler Faktoren. Der mit Abstand größte Einflussfaktor ist die schwer zu kontrollierende Neigung der Kamera, ob infolge der Körperneigung oder Dekalibrierung des Sensors (s. Abb.4.3.5). Ein weiteres Problem ist die Verschwimmung der Konturen infolge der raschen Kamerabewegungen, was eine genaue Koordination zwischen den Bewegungs- und Messungsabläufen erfordert und trotzdem nicht ganz auszuschließen ist. Schließlich beeinflussen auch die hardware-bedingten Ungenauigkeiten von 1-2 Pixel sowie allerlei Rundungsfehler das Messergebnis. Alle diese Faktoren werden durch die

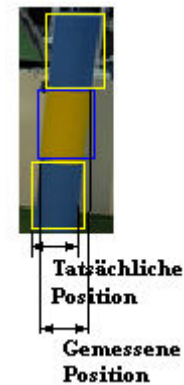


Abb. 4.3.5:
*Neigungsbedingter
Messfehler*

Festlegung der Varianz der Normalverteilung quantitativ zusammengefasst. Die Bestimmung der geeigneten Standardabweichung erfolgte empirisch (s. 5.1 Tests) anhand der Bewertung des Lokalisierungsergebnisses unter Verwendung verschiedener Varianz-Werte im Hinblick auf die Optimierung des Verhältnisses zwischen der Genauigkeit und der Robustheit der Schätzung. Dieser zugegebenermaßen nicht ganz analytisch-methodische dafür aber sehr zielgerichtete Ansatz basiert auf der Überzeugung, dass eine komponentenweise Analyse des Fehlermodells angesichts der Komplexität der Vorgänge sowie ihrer Zusammenhänge die dafür notwendige Mittel nicht rechtfertigen würden. Allerdings wäre die Berücksichtigung einer explizit gemessenen Körperhaltung in Zukunft denkbar, im Moment steht uns die dafür notwendige Sensorik nicht zur Verfügung.

Die Verwendung unterschiedlicher Varianz-Werte für die Toren und Säulen ist vor allem aus der theoretischen Sicht sinnvoll, denn bei den aus mehreren Segmenten zusammengesetzten Säulen entstehen in der Regel größere Messfehler. Auch der infolge der Werte-Skalierung gemachte Fehler ist bei den Polen größer.

Die Verwendung einer distanzabhängigen Varianz hat dagegen keine signifikanten Verbesserung der Genauigkeit gebracht, dafür hat sich die zusätzliche Distanzschätzung negativ auf die Laufzeit der Methode ausgewirkt und wurde unterlassen. Die entsprechenden Testergebnisse (s.5.3) sowie die ermittelten Varianz-Werte sind im Abschnitt Tests zu finden. Zu beachten dabei ist die erneut auftretende Fließkommazahlen-Problematik der Exponentenberechnung.

4.3.5 Zusammenfassung der Schätzung

Die berechneten Wahrscheinlichkeiten zu den einzelnen Landmarken werden zur Gesamtwahrscheinlichkeit der von dem Partikel repräsentierten Pose aufmultipliziert und in dem *Belief*-Feld gespeichert. Die Prozedur wird für alle Partikel wiederholt, wobei gleichzeitig das Partikel mit der maximalen Wahrscheinlichkeit ermittelt wird. Dies geschieht aus zweierlei Gründen. Zum einen wird der Belief-Wert zur Beurteilung des Ergebnisses der Schätzung genutzt, um beispielsweise fehlgeschlagene Lokalisierungsversuche zu identifizieren. Ist der größte Belief-Wert gleich 0, dann deutet das auf einen gravierenden Messfehler hin. Die zuletzt ermittelten Messergebnisse müssen in dem Fall gelöscht, die Messung wiederholt und die Schätzung erneut durchgeführt werden. Liegt der größte Belief-Wert über mehrere Lokalisierungsanläufe hinweg unterhalb einer bestimmten Schwelle, so wird von einem kompletten Versagen der Methode ausgegangen, wobei die große Mehrheit der Partikel sich an einem lokalen Maximum der Likelihood-Funktion zusammengezogen hat. In dem Fall werden die Partikel wieder gleichmäßig auf dem Feld verteilt und auf die nächste Positionsabfrage hin die Routine der globalen Selbstlokalisierung gestartet. Der zweite Grund für die Ermittlung des maximalen Belief-Werts ist das Vorgehen beim Zusammenfassen der Partikelmenge zur Positionsschätzung.

Gemäß der SIR-Theorie kann die Positionsschätzung als gewichtetes Positionsmittel (Erwartungswert) definiert werden. Allerdings ist diese Strategie ungeeignet für den multimodalen Fall. Andererseits sind Verfahren mit denen sich aus einer Stichprobe, wie sie in der Abbildung 4.3.6 dargestellt ist, mehrere unimodalen Komponenten identifizieren lassen (z.B. Kerndichteschätzung), für unsere Zwecke unverhältnismäßig aufwändig,

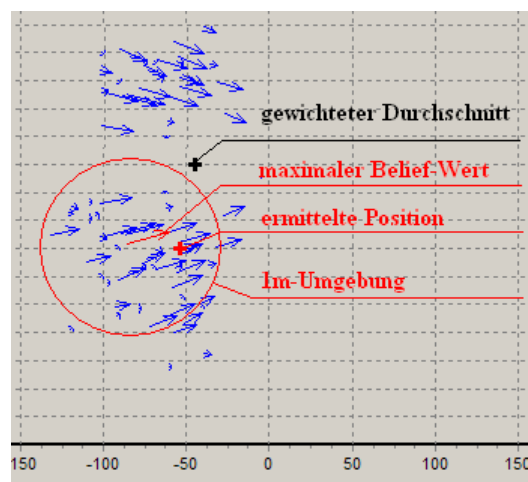


Abb. 4.3.6: Positionsschätzung anhand der Partikelmenge

denn sollte es zu einer solchen

„multimodalen“ Stichprobe kommen, dann gilt das Lokalisierungsergebnis eher als nicht sicher und falls die Mehrdeutigkeit innerhalb mehrerer Anläufe nicht aufgelöst wird, muss die Partikelmenge reinitialisiert werden. Die Erkennung der beschriebenen Situation geschieht anhand des Abstandes zwischen dem Mittelwert der gesamten Partikelmenge und dem Mittelwert der Partikel, die sich in der 1m-Umgebung des Partikels mit dem maximalen Belief-Wert befinden (s. Abb. 4.3.6). Diese heuristische Methode hat den Vorteil, dass die Problemerkennung ganz beiläufig stattfindet. In den allermeisten Fällen befindet sich die große Mehrheit der Partikel ohnehin innerhalb der festgelegten Grenzen und sollte es trotz unserer defensiven Erstlokalisierungsstrategie zu den Mehrdeutigkeiten kommen, so stellt die beschriebene Methode immer noch eine gute Positionsschätzung dar.

4.3.6 Eigenschaften der Implementierung

Offensichtlich lässt sich die Pose des Roboters anhand von 3 Messungen eindeutig bestimmen. Allerdings gibt es fast immer eine Reihe von Posen, die die entsprechenden Gleichungen annähernd erfüllen und somit ebenfalls einen großen Likelihood-Wert aufweisen. Die Abbildung 4.3.7 illustriert, wie die Likelihood

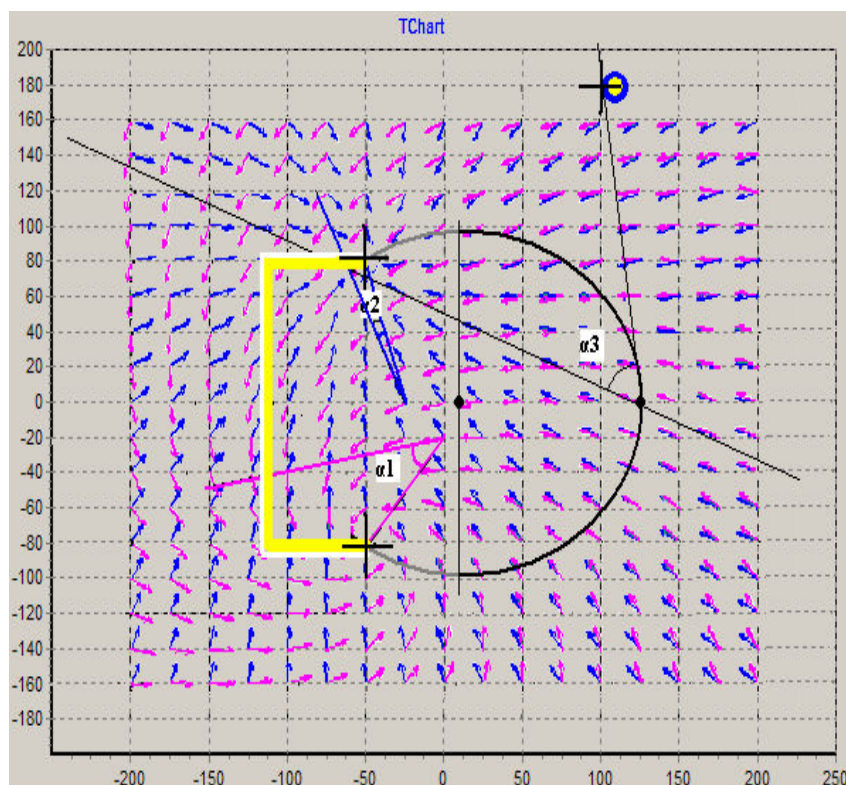
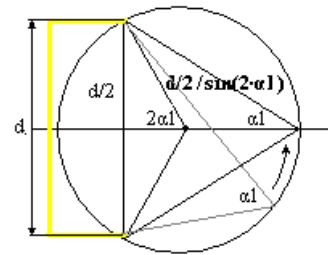


Abb. 4.3.7: Die Likelihood-Funktion des Sensormodells

Funktion zustande kommt, wobei der jeweilige Funktionswert als Summe der

blauen und magenta Vektoren definiert ist. Eine Landmarke alleine liefert kaum Informationen über die aktuelle Position, denn für jedes Koordinatenpaar x, y findet man einen Blinkwinkel unter dem man die Einzelmessung machen könnte. Immerhin lässt sich damit der Orientierungswinkel der Partikel korrigieren, die für das Bewegungsmodell sehr wichtig ist. Betrachtet man nun die beiden Tor-Features, dann stellt man fest, dass die Lösung des Gleichungssystems einen in der Abb. 4.3.7 dargestellten Halbkreis festlegt. Alle Partikel, die sich auf dem eingezeichneten Halbkreis befinden, haben den gleichen maximalen Likelihood-Wert¹⁸. Genau das passiert wenn, man die Erstlokalisierung anhand von lediglich 2 Landmarken durchführt: die Partikel verteilen sich auf dem Halbkreis und bilden somit die Grundlage für die mehrdeutigen Positionsschätzungen. Der Radius des Halbkreises lässt sich ganz einfach mithilfe der Kreiswinkel- und Peripheriewinkel-Sätze bestimmen (s. Abb. 4.3.8). Gemäß Formel:

$$\text{Radius} = \frac{d/2}{\sin(2 \cdot \alpha_1)} \quad (\text{F. 4.3.1})$$



vergrößert sich der Radius und somit die Partikelstreuung mit der steigenden Entfernung zur Landmarke (der Winkel α_1 wird kleiner). Der Problematik begegnet man vor allem beim Tracking, wenn man nur auf Daten eines einzigen Bildes angewiesen ist, um die Schätzung vorzunehmen. Typischerweise hat man dann weniger als 3 Messungen zur Verfügung und der Winkel α_1 ist in dem Fall kleiner als der Öffnungswinkel der Kamera $\approx 40^\circ$. Somit sind weiter entfernte Objekte weniger informativ als naheliegende.

Abb. 4.3.8: Die Halbkreis-Verteilung der Partikel nach 2-Features-Schätzung

Auch Partikelwolken, die in der Nähe des Halbkreises liegen, werden verstärkt, wenn auch in einem kleineren Ausmaß. Somit ist die lokale Selbstlokalisierung nur bedingt zur Auflösung der Mehrdeutigkeiten bei der Positionsschätzung geeignet – die Wahrscheinlichkeit, dass „falsche“ Partikel gefördert werden ist zu hoch. Somit hängt das Erfolg der Tracking-Prozedur ganz entscheidend von der Zuverlässigkeit der vorangegangenen globalen Selbstlokalisierung, die ja mindestens 3 Landmarken in die Berechnung einbezieht. Wie die Abbildung 4.3.7 zeigt, legt die 3. Messung (der Winkel α_3) die Pose eindeutig fest, zumindest in der Theorie.

Damit auch die MCL-Methode zur richtigen Schätzung gelangt, müssen ausreichend viele Partikel sich in der Nähe des globalen Likelihood-Maximums

¹⁸ Die blauen und magenta Vektoren an den Stellen sind kollinear

befinden, sonst werden die Partikel aus der Umgebung der lokalen Minima durch ihre Anzahl oder Nähe zu den jeweiligen Extrema ein größeres Gewicht bekommen und die Schätzung nachhaltig beschädigen. Je größer die Partikeldichte ist, umso größer ist die Auflösung der Dichteschätzung (s.3.4.3) und umso weniger ausgeprägt ist die geschilderte Problematik. Die Tests haben ergeben, dass mit etwa 200 bis 250 Partikel und somit fast 100 Partikel mehr als bei den vierbeinigen Robotern, die erforderliche Partikeldichte und dadurch die gewünschte Zuverlässigkeit und Robustheit der Schätzung erzielt werden können. Da uns für diese Partikelzahl nicht genügend Speicher zur Verfügung stand, musste die notwendige Partikeldichte auf eine andere Weise erreicht werden.

4.3.7 Optimierung des Sensormodells

Die einzige Möglichkeit die Partikeldichte zu erhöhen, ohne den Speicher für zusätzliche Partikel zu verschwenden, ist die Dimensionalität des Raums in dem die Partikel verteilt werden zu verringern. Da man jedoch die Dimensionalität des Zustandsraums sicherlich nicht verkleinern kann, müssen die fehlenden Zustandsraum-Komponenten der Partikel rechnerisch ermittelt werden. Dabei kann man beispielsweise die bereits angesprochene deterministische Scan-Matching-Methode einsetzen,

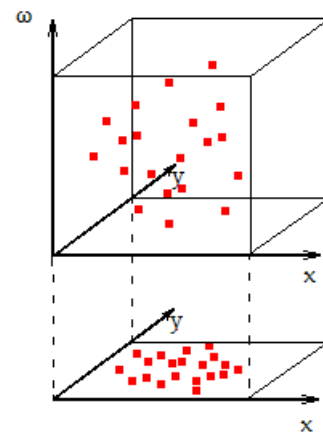


Abb. 4.3.9: Dimensionsreduktion des Partikelraums

bei der man die Zustandsparameter übernimmt, die am besten zu den aktuellen Messdaten passen. Der Preis für den erzwungenen Determinismus ist die stochastische Abhängigkeit der berechneten Komponenten, wodurch die selektierende Wirkung des "Survival of the fittest"-Prinzips abgeschwächt wird.

Es bietet sich an, die Orientierung als Dimension des Raums, in dem die Partikel verteilt werden, zu eliminieren. Zum einen lässt sich die Scan-Matching Methode für die Bestimmung der Orientierung nahezu direkt anwenden, zum anderen wird die fehlende Komponente parametrisch und dadurch i. A. zuverlässiger berechnet, wobei die Genauigkeit der Orientierungswerte wichtiger als Positionsgenauigkeit ist. Die restlichen beiden Dimensionen eines Partikels - die Koordinaten des Roboters auf dem Feld - werden nach wie vor mit der „klassischen“ MCL abgeschätzt. Gegeben eine Position auf dem Feld, sowie eine Menge von Landmarkenmessungen, dann lässt sich die zugehörige Orientierung bestimmen, indem man für jede Landmarke die Orientierung des Roboters

bestimmt mit der er die Landmarke unter dem gegebenen Winkel sehen würde (s. Abb. 4.3.10), diese Orientierung und die Koordinaten des Roboters als Pose zusammenfasst und für diese Pose den zugehörigen Likelihood-Wert mithilfe des in den vorangegangenen Unterabschnitten beschriebenen Verfahrens berechnet. Die Landmarke, die zur Bestimmung der jeweils angenommenen Orientierung genutzt wurde, hat immer den Likelihood-Wert gleich 1 und wird nicht berücksichtigt. Diese Routine wiederholt man für alle Landmarken, wobei die Orientierung mit der der maximale Likelihood-Wert erreicht wurde, schließlich als die Partikel-Orientierung übernommen wird. Durch die mehrfache Auswertung der Likelihood-Funktion je ein Partikel entsteht ein Mehraufwand, der von der Anzahl der Messungen abhängt. Im Gegenzug erzielt man dadurch signifikant höhere Genauigkeit bei der Ermittlung der Partikelorientierung, sowie insgesamt höhere Zuverlässigkeit der Sensormodell-Ergebnisse. Es mussten zwar einige Optimierungen im Programmablauf, sowie vollständige Eliminierung der Fließkomma-Werte vorgenommen werden, damit die Laufzeit der Funktion innerhalb der spezifizierten Grenzen bleibt, aber letztendlich hat sich die Kombination der probabilistischen MCL-Methode mit dem analytischen Scan-Matching-Verfahren gelohnt. Die konkreten Zahlen zum Thema Sensormodell, sowie der quantitative Vergleich der beiden Methoden sind im Kapitel Tests (Abschnitt 5.3) nachzulesen.

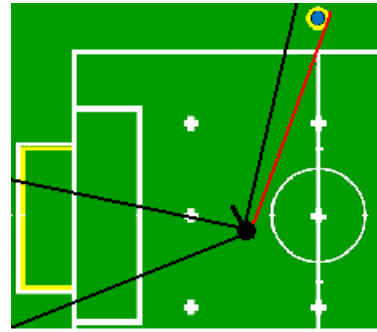


Abb. 4.3.10a: Ausrichtung nach dem Maximum der Säule

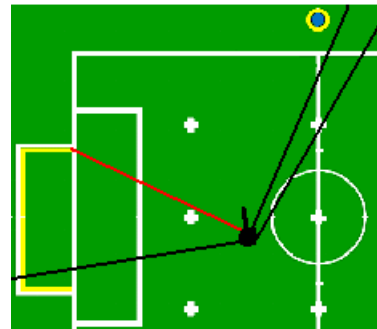


Abb. 4.3.10b: Ausrichtung nach dem Maximum des Tores

4.4 Implementierung des Bewegungsmodells

Die Implementierung des Bewegungsmodells orientiert sich im Großen und Ganzen an den theoretischen Vorbetrachtungen im Kapitel 3.3.2. Alle Motions, die eine Veränderung der Pose verursachen, übergeben die erwarteten Veränderungen der 3 Zustandsparameter (dx -Strecke in die Orientierungsrichtung, dy -Seitwärtsdrift, $d\omega$ -Orientierungsänderung) in dem lokalen Koordinatensystem des Roboters an die Bewegungsmodell-Funktion, die die partikel-spezifische Transformation ins globale Koordinatensystem vornimmt und den transformierten Werten entsprechend die Partikel aktualisiert. Die Parameter-Werte wurden für die meisten Bewegungen experimentell ermittelt und sind fest encodiert. Problematisch wird es allerdings bei der Modellierung unserer dynamischen Laufmethode, die eine besondere Herangehensweise benötigt. Dabei macht man sich zunutze, dass bei der Berechnung der einzelnen Schritte ein gewissermaßen invertiertes Problem gelöst wird, nämlich anhand eines Zielzustandes die dafür notwendigen Motion-Befehle bestimmen. Allerdings stirbt man dabei meistens keine bestimmte Pose des Roboters auf dem Feld an, sondern man trifft verhaltensspezifische Entscheidungen anhand der "rohen" Messdaten, ohne dass das Ziel der Handlung eine Repräsentation im Weltmodell bekommt. Wenn man beispielsweise zum Ball geht, dann werden die Bewegungen anhand der Ballposition im Bild und nicht anhand der Ballposition im globalen Koordinatensystem berechnet. Das macht die Verwendung der Informationen über den angestrebten Zielzustand für die Modellierung der auszuführenden Bewegung schwieriger. Auf der anderen Seite hat sich eine starre Abbildung der berechneten Schritt-Parameter auf die tatsächlich stattgefundenen Zustandsänderung des Roboters in der Praxis als nicht präzise genug erwiesen, wobei vor allem die längeren Phasen des „blinden“ Laufens Probleme bereiten. Deswegen haben wir für viele Verhalten eine alternative Methode der Bewegungsmodellierung entwickelt, bei der nicht nur Effekte der einzelnen Motions auf den Zustand des Roboters berücksichtigt werden, sondern auch die Effekte der ganzen Verhaltensaussführungen. Wenn man beispielsweise zum Ball geht, dann wird die notwendige Ausrichtungsänderung noch während der ersten Schritte anhand der Ballpositionsmessungen bestimmt(s. Abb. 4.4.1). Alle nachfolgenden Schrittbewegungs-Informationen werden lediglich zur Präzisierung der Orientierungsänderungen verwendet. Man geht also davon aus, dass am Ende der Verhaltensaussführung die Ausrichtung der Roboter gleich der Richtung des Vektors ist, der die Startposition der Laufbewegung mit der Zielposition verbindet. Das gilt natürlich nur, wenn der Ball während der Ausführung des

Verhaltens nicht bewegt wurde, was man anhand der großen Ballsprünge innerhalb des Bildes, ohne dass im letzten Schritt bedeutende Ausrichtungskorrekturen gemacht wurden, feststellen kann. Falls der Ball bewegt wurde, so kann man immer noch auf die Daten zugreifen, die die Änderungen der Pose nach den einzelnen Schritten akkumulieren. Im abschließenden Teil der Verhaltensausführung vergleicht man die erwartete Pose mit der Pose (s. Abb. 4.4.1), die mithilfe der Schrittdaten berechnet wurde und, falls die Differenz innerhalb der vordefinierten Grenzen ist, übernimmt man die verhaltensbasierte Schätzung. Diese Strategie kann bei allen Verhalten eingesetzt werden, wo die erwarteten Posenparameter von Anfang an bekannt sind.

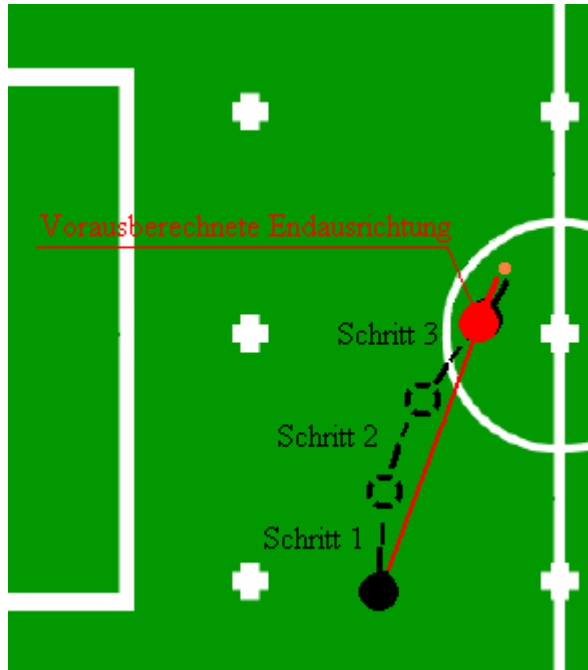


Abb. 4.4.1: Motionsbasierte Bewegungsverfolgung gegen die verhaltensbasierte Schätzung

Auch bei der Modellierung des Odometrie-Fehlers spielten die Effizienzüberlegungen eine herausragende Rolle. Der Bewegungsverlauf bei den humanoiden Robotern wird von zahlreichen äußeren und inneren Faktoren beeinflusst: von den reaktiven Maßnahmen zur Gleichgewichtserhaltung bis zum Durchrutschen der Füße und Stolpern. Die Auswirkungen der Faktoren sind gravierend, lassen sich allerdings kaum formalisieren. Deswegen modellieren wir den Fehler ganz einfach durch eine normalverteilte Zufallsvariable, deren Varianz experimentell bestimmt wurde. Zur Ermittlung der Fehlerverteilung wurden Testläufe gemacht, wobei die Motions Anlaufen/Stoppen, Vorwärtslaufen, Seitwärtslaufen, auf der Stelle Drehen und Aufstehen mehrfach wiederholt wurden (s. 5.2 Tests). Die entsprechenden Messungen wurden manuell durchgeführt. Interessanterweise zeigt der Q-Q-Plot der Laufdaten ein Bild, das typisch für normalverteilte Zufallsvariablen ist. Die Schätzung der Varianz erfolgte mit dem Standardschätzer:

$$S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2.$$

Die normalverteilte Zufallsvariable wird mithilfe der Zwölferregel simuliert, wobei man die Summe von zwölf gleichverteilten Zufallsvariablen als annähernd normalverteilt betrachtet. Die Methode hat den Vorteil, dass sie lediglich Additionen (sowie rand - Operationen) beinhaltet und sehr effizient bei der für unsere Zwecke vollkommen ausreichenden Qualität der simulierten Zufallsvariablen ist. Die Bewegungsmodell-Prozedur wird unter allen Selbstlokalisierungsprozeduren am häufigsten aufgerufen, wobei für jedes Partikel 3 normalverteilten Zufallsvariablen simuliert werden müssen. Um die Laufzeit der

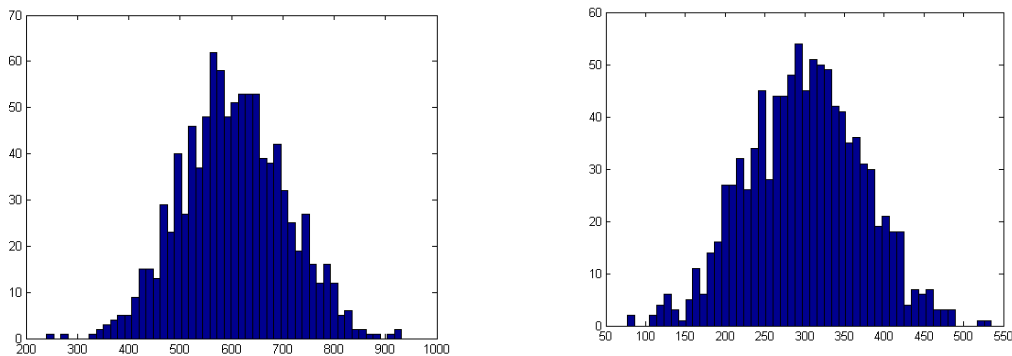
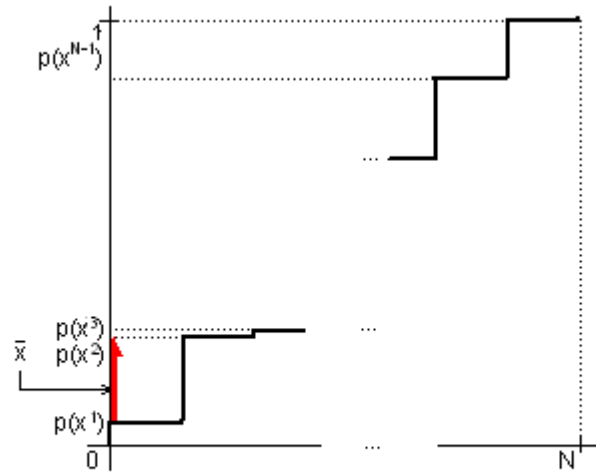


Abb. 4.4.2: Simulation der normalverteilten Zufallsvariablen durch Summe von 12(links) und 6 gleichverteilten Zufallsvariablen

Methode zusätzlich zu verringern, werden die Fehler durch lediglich 6 statt 12 gleichverteilten Zufallsvariablen simuliert. Die Folgen für die Verteilung werden auf der Abbildung 4.4.2 dargestellt.

4.5 Implementierung der Sampling-Methode

Eine effiziente Implementierung der simulierten nicht-deterministischen Probeziehungsmethode ist nicht trivial und numerisch problematisch. Daher schauen wir uns zunächst einmal einen naheliegenden Algorithmus an, dessen Laufzeit $O(N \cdot \log N)$ beträgt. Die Idee dabei ist die kumulative Verteilungsfunktion der Schätzung zu bestimmen, diese als Unterteilung des Intervalls $[0, \sum w^N (=1, \text{falls normiert})]$ in Abschnitte: $\{[0, w^1], \dots, (\sum w^i, \sum w^{i+1}], \dots, (\sum w^{N-1}, \sum w^N)\}$ aufzufassen (s. Abb. 4.5.1), gleichverteilte Pseudo-Zufallszahlen u_i aus $[0, \sum w^N]$ zu erzeugen und unter Verwendung von der binären Suche die Zufallszahlen in die Intervalle einzusortieren. Offensichtlich ist die Wahrscheinlichkeit, eine Pseudo-Zufallszahl aus dem Intervall $(\sum w^i, \sum w^{i+1}]$ zu generieren, proportional zu der normierten Länge des Teilintervalls und somit zum normierten Gewicht w^{i+1} . Das Verfahren ist einfach zu implementieren und funktioniert einwandfrei, allerdings belastet die N-fache Suchroutine die Laufzeit, was bei einer großen Partikelzahl N zum Problem werden kann.



*Abb.4.5.1: Partikel Filter, Sampling
 x^2 wird ausgewählt.*

Die zweite Methode, die wir in Betracht gezogen haben¹⁹, basiert auf einer bekannten Methode aus der Ordnungsstatistik und hat eine Laufzeit von lediglich $O(N)$. Das Verfahren besteht aus folgenden Schritten:

1. Simulation von $N+1$ exponentiellverteilten Variablen mit $t_i = -\log(u_i)$, wobei u_i gleichverteilte Pseudo-Zufallszahlen, wie im letzten Verfahren.
2. Bestimmung der kumulierten Verteilungsfunktionen von t^i (T_i) und w^i (W_i).
3. Verschmelzung der beiden Funktionen derart, dass:
für $i = 0..N-1$ und $j=0..N-1$
wenn $W_j T_N > T_i$ dann ziehe Stichprobe x_j , $i=i+1$
sonst $j=j+1$.

Intuitiv entspricht die Methode einer Art Simulation der binären Suche für N gleichverteilte Zufallsvariablen in nur einem Durchlauf. Die Laufzeit der Methode auf einem Rechner mit FPU ist selbst bei einer geringen Partikelzahl von 100 Partikel messbar kürzer (s. Tests 5.4). Das sieht allerdings ganz anders aus, wenn man die gleiche Berechnung auf dem roboter-eigenen Atmel-Prozessor durchführt: die notwendige Logarithmierung bewirkt eine (für die geringe Partikelzahl) erhebliche Laufzeitsteigerung, wobei diese mit einem durch die fließkommazahlen-freie Implementierung bedingten gravierenden Genauigkeitsverlust einhergeht.

¹⁹ An Improved Particle Filter for Non-linear Problems, James Carpenter, Peter Clifford, Paul Fearnhead

Auf der Abb. 4.5.1 sieht man die beiden Sampling-Methoden in Aktion. Wir haben eine willkürliche Dichtefunktion (rote Linie) mit den beiden Methoden durch eine Stichprobe approximiert und das Ergebnis verglichen. Typisch für die zweite Methode (unten) in unserer Implementierung sind häufige Ausreißer, einer davon ist auch auf dem Bild zu sehen. Durch die Portierung der Methode könnte sich dieses Problem zusätzlich verschärfen, was unsere Entscheidung, zugunsten der ersten Methode, trotz der möglichen und für die Gesamtlaufzeit der SMC-Implementierung kritische Laufzeitsteigerung bekräftigt hat. Mit der auf der binären Suche basierenden Sampling-Methode steigt die Laufzeit der MCL auf $O(N \log N)$ in der Anzahl der Partikel, was immer noch um vielfaches effizienter ist als die Markow-Lokalisierung bei der gleichen Genauigkeit, zumal die

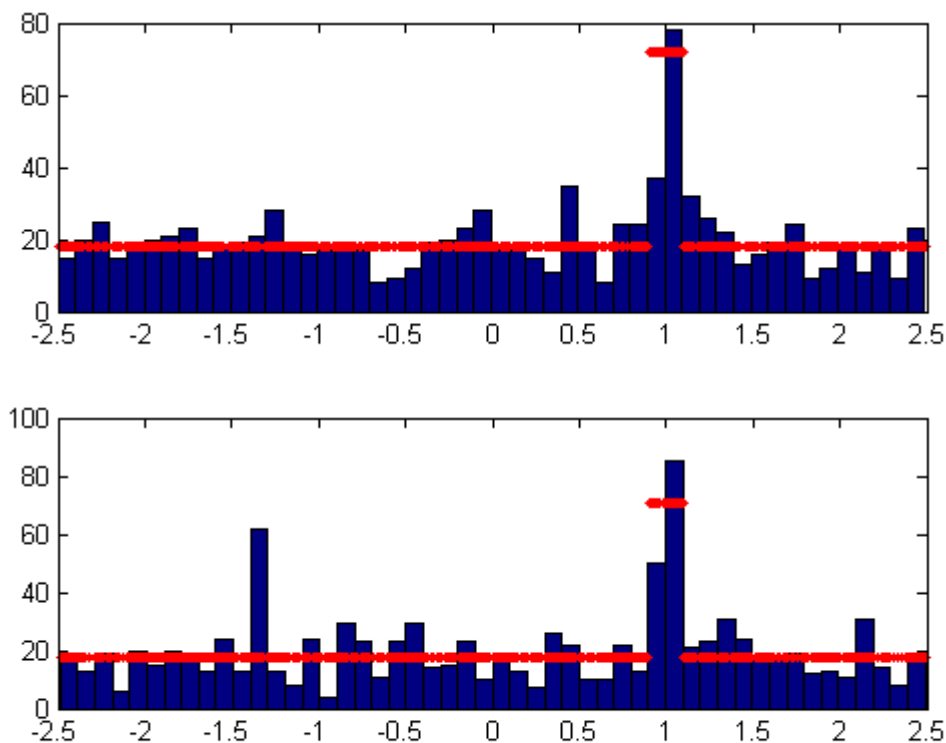


Abb.4.5.2: Partikel Filter, Vergleich der beiden Sampling-Methoden: mittels Binäre Suche(oben), mittels Verschmelzung zweier Verteilungsfunktionen

Sensormodell-Routine, wenn auch mit der Komplexität von $O(N)(N < 500)$ nach wie vor die Gesamtlaufzeit dominiert.

Die Implementierung der gewählten Methode unter den Bedingungen der knappen Ressourcen erfordert weitere Überlegungen. Die bei der Methode notwendige kumulative Verteilungsfunktion wird in den Partikeln gespeichert, wobei das Belief-Feld verwendet wird²⁰, wodurch kein zusätzlicher Speicherplatz verbraucht wird. Die Zahlen werden so skaliert, dass sie in insgesamt 10 Bits hineinpassen,

²⁰ Aus diesem Grund werden 2 Bytes für das Feld verwendet.

denn die restlichen 6 Bits des zweiten Bytes werden für andere Zwecke verwendet. Die Erzeugung neuer Partikel erfolgt in 2 Schritten: im ersten Schritt wird ein Partikel gemäß dem vorgestellten Algorithmus zufällig gezogen, dann wird eine simulierte normalverteilte Zufallsvariable $N(0,\sigma)$ zu den Zustandskomponenten des Partikels dazu addiert. Auf diese Weise wird eine höhere Streuung der Partikel erzielt und dem Degenerationsproblem entgegengewirkt, bei dem nur wenige Partikel eine Wahrscheinlichkeit $\gg 0$ haben. Diese zusätzliche Streuung ist vor allem dann wichtig, wenn der Roboter stillsteht, denn bewegte Partikel enthalten bereits eine nicht-deterministische Komponente (s. 4.4). Der Wert von σ wurde experimentell ermittelt, sodass ein Kompromiss zwischen dem Genauigkeitsverlust und den Konvergenzeigenschaften geschlossen wird. Damit auch bei der Erzeugung der neuen Partikel kein zusätzlicher Speicherplatz „verschwendet“ wird, werden die beiden Schritte von einander getrennt: zuerst zieht man alle 100 Stichproben, dann generiert man die neuen Partikel. Die Ergebnisse des ersten Schrittes werden in den 6 letzten Bits des zweiten Bytes des *Belief*-Feldes gespeichert, indem jedes mal wenn ein Partikel gezogen wird, der Zähler im *Belief*-Feld des Partikels inkrementiert wird. Nach dem man die benötigte Anzahl der Proben gezogen hat, geht man alle Partikel durch und generiert für jedes Partikel die entsprechende Anzahl neuer Partikel. Ist der Zähler = 1, so wird der Partikel gleich ersetzt, indem man eine Zufallsvariable zu den einzelnen Zustandskomponenten des Partikels dazu addiert. Ist der Zähler >1 , so werden die neuen Partikel an der Stelle der Partikel gespeichert dessen Zähler =0 ist. Offensichtlich findet sich immer ein 0-Partikel in der Menge, wenn es ein Partikel mit dem Zähler >1 gibt, denn die Anzahl der gezogenen Partikel ist gleich der Anzahl der alten Partikel. Damit nicht jedes mal das gesamte Particles-Feld durchsucht werden muss, markiert ein Zeiger die Position des nächsten leeren Partikels.

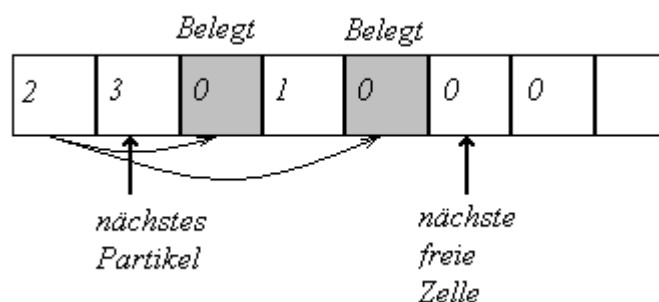


Abb. 4.5.3: Algorithmus für die Partikel-Speicherung

Man beachte, dass durch die Platzbeschränkung für den Zähler auf 6 Bits kein Partikel mehr als 63 Mal gezogen werden kann, was nicht weiter dramatisch ist und als zusätzliche Vorkehrung gegen das Generationsproblem angesehen werden

kann. Die Effizienz-Orientiertheit der Implementierung hat allerdings mit der Spaltung der Bytes ihren Höhepunkt erreicht, obwohl das gesamte Bild des Programms von der Ressourcenknappheit geprägt ist. Trotz der Maßnahmen zur Kontrolle des Speicherverbrauchs bleibt auch die Laufzeit der Methoden innerhalb der spezifizierten Grenzen, nicht zuletzt durch den Einsatz der Nachschlagtabellen für die trigonometrische Funktionen und Verwendung der Festkommazahlen.

Kapitel 5

Tests

In diesem Kapitel werden sowohl dynamische Qualitätssicherungsmaßnahmen als auch Versuche zur Ermittlung der optimalen Parameter vorgestellt. Zu den Qualitätssicherungsmaßnahmen gehören in erster Linie Probeläufe zur Kontrolle der Korrektheit und Vollständigkeit des Programms, sowie der Einhaltung der spezifizierten Laufzeit- und Speicherverbrauch-Werte. Eine Testautomatisierung stellt angesichts der realitätsnahen Umgebung eine große Herausforderung dar. Vor allem die Rekonstruktion der Testbedingungen und die Auswertung des Testergebnisses ist unter diesen Umständen problematisch. Deswegen wurden die meisten Tests manuell durchgeführt. Auch die Debugging-Routine gestaltet sich schwierig, da diese von der Hardware des Roboters nicht unterstützt wird. Die Übermittlung von Informationen über den Programmzustand müssen im Code als Ausgaben implementiert werden, worauf das Programm auf den Roboter geladen wird. Die Zeichenausgabe über die serielle Schnittstelle nimmt allerdings ziemlich viel Zeit in Anspruch ($1.7 \cdot 10^{-4}$ s pro Byte) und beeinträchtigt spürbar den Programmverlauf.

Getestet wurden sowohl die einzelne Funktionen des Moduls, wie das Sensor- und Bewegungsmodell, als auch das komplette System. Beim Systemtest wurden lokale und globale Selbstlokalisierungsverfahren getrennt und als Hintereinander-Ausführung auf ihre Korrektheit und Robustheit getestet. Die Komponententest wurden entwicklungsbegleitend unter anderem als Entscheidungsgrundlage zwischen verschiedenen Implementierungen oder für die Parameterbestimmung genutzt. In diesem Kapitel gehen nach dem Bottom-Up-Prinzip vor. Als erstes werden die Experimente zur Ermittlung der Sensormodell- und Bewegungsmodell-Fehlervarianzen präsentiert. Danach werden Vergleiche zwischen den in dieser Diplomarbeit vorgestellten Implementierungsalternativen angestellt und die Bewegungsmodell-, Sensormodell und Sampling-Funktionen getestet. Schließlich werden die Systemtests beschrieben mit dem Kidnapped-Robot-Problem-Test als Höhepunkt des Kapitels. Der ultimative Test wird der Wettbewerb 2008 sein, der in Suzhou, China stattfindet.

5.1 Sensormodell-Fehlervarianz

Hier wird die Ermittlung des im Abschnitt 4.3.4 angesprochene Fehlervarianz des Sensormodells beschrieben. Die größte Schwierigkeit dabei bestand darin, ein Kompromiss zwischen der Genauigkeit der Messung und Robustheit der Sensormodellfunktion zu finden. Wählt man zu kleine Fehlervarianz, dann werden

die meisten Partikel mit 0 bewertet und man riskiert, dass das Degenerationsproblem verschärft wird oder gar die Lokalisierung häufiger fehlschlägt, weil alle Beliefs gleich 0 werden. Auf der anderen Seite führt eine zu große Fehlervarianz dazu, dass Partikel überbewertet werden. Die tatsächlich gute Partikel verlieren sich unter allen denjenigen die den maximalen Belief-Wert bekommen, was zu falschen bzw. ungenauen Positionsschätzungen führt.

Der Testverlauf ist denkbar einfach. Der Roboter wird an verschiedenen Positionen auf dem Feld platziert und die Sensormodell-Funktion wird ausgeführt. Dabei muss man darauf zu achten, dass der Roboter genügend Landmarken in einem Bild zu sehen bekommt. Dafür mussten in einigen Fällen die Säulen verschoben und das Feldmodell entsprechend verändert werden. Anschließend vergleicht man die geschätzten Positionen mit den gemessenen Werten. Die Varianz, mit der der kleinste Gesamtfehler gemacht wurde, wird übernommen. Die Varianzen wurden nach dem Prinzip der binären Suche präzisiert. Hier ist ein Überblick der Fehlerwerte für Varianzen: 1: Var=0, 2: Var=10, 3: Var=20, 4: Var=50.

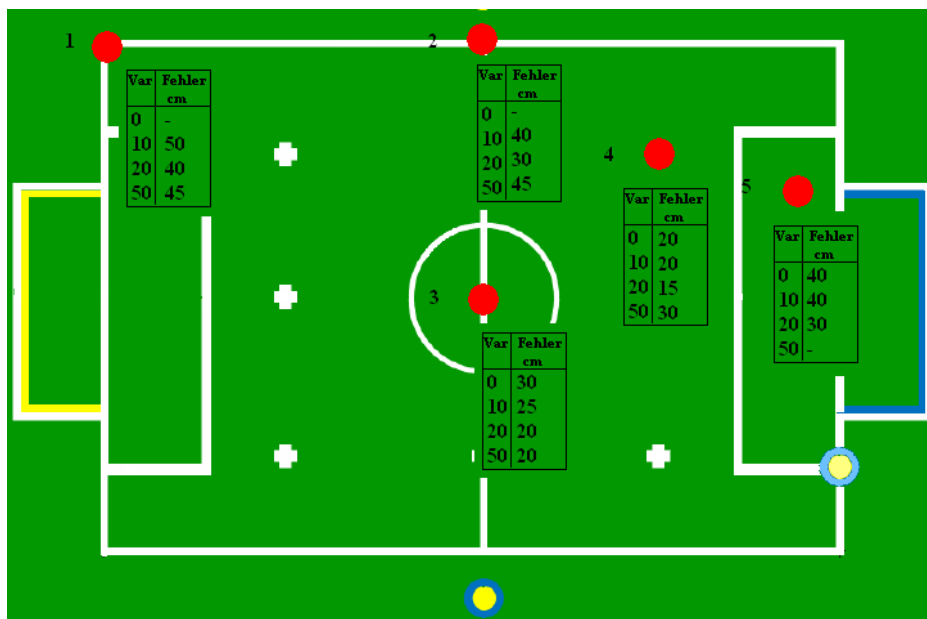


Abb. 5.1: Tests, Bestimmung der Sensorfehlermodell-Varianz

Gesamtwerte:

Varianz	Gesamtfehler (x+y) cm (5 Messungen)
0	-
10	175
20	145
50	>200

Der Strich bedeutet in dem Fall, dass die Schätzung misslungen ist, was auf die Qualität der Messung bei sehr nahen Objekten zurückzuführen ist. Das Bild zeigt die ausgewählten Stellen, die die entsprechenden Äquivalenzklassen repräsentieren. Es ist auffällig, dass die Ergebnisse am schlechtesten ausfallen, wenn man sich zu weit von bzw. zu nah zu den jeweiligen Landmarken befindet. Gelegentliche Fehlschläge des Sensormodells zeigen uns, dass die Robustheit der Methode sinkt, wenn man sich anhand von Objekten lediglich eines Bildes zu lokalisieren versucht. Problematisch wird es ferner, wenn man sich im Grenzbereich bewegt, wenn man sich beispielsweise am Rande des Feldes befindet und die Varianz zu klein gewählt wurde. Mit dem Varianzwert = 20 $\approx 4.5^2$, hat man praktisch überall auf dem Feld gute Positionsschätzungen bekommen.

5.2 Bewegungsmodell-Parameter, Bewegungsmodell-Fehlervarianz

Bei diesem Test/Experiment geht es darum die Parameter verschiedener Motions, sowie die Varianz der Bewegungsfehlermodellierung abzuschätzen, indem die ausgewählte Motions ausgeführt und das Effekt gemessen wird. Auf diese Weise werden die an die Bewegungsmodell-Methode zu übergebende Parameter bestimmt.

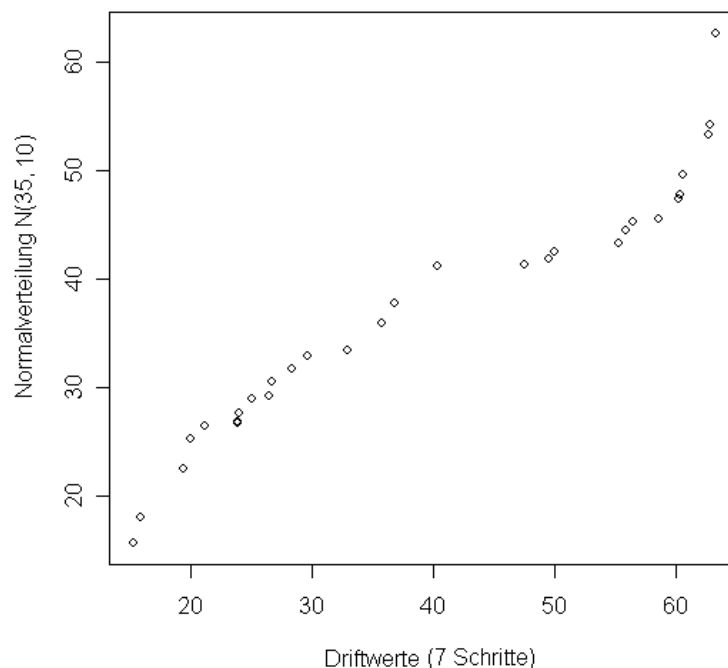


Abb.5.2: Bewegungsmodell-Fehlervarianz: (vermutliche) Normalverteilung der Drift-Abweichung

Bezeichnend ist, dass alle diese Experimente einen Stichprobensatz ähnlich dem auf der Abb. 5.2 abgebildeten ergaben. Die kleinen bis mittelgroßen Fehlerwerte sehen auf dem Q-Q-Plot aus, als wären sie annähernd normalverteilt, wohingegen die großen Fehlerwerte nicht mehr normalverteilt zu sein scheinen.

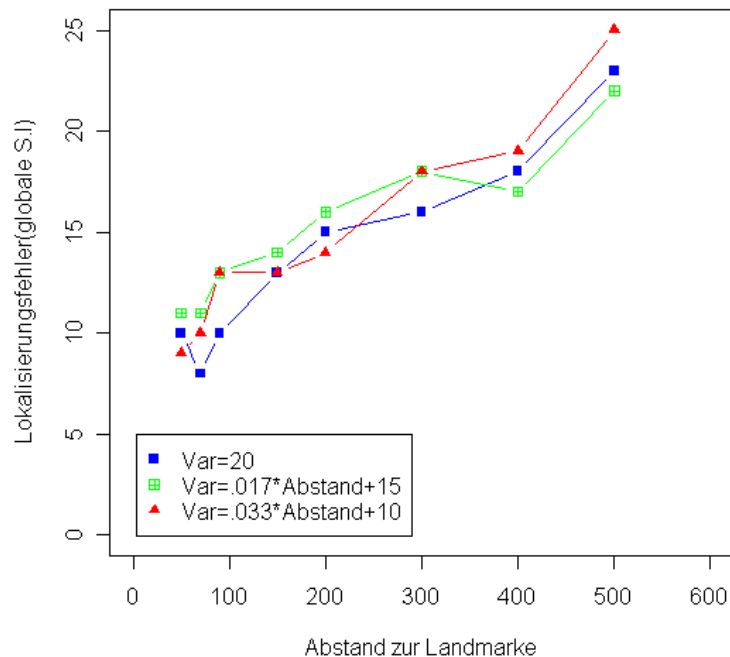
Die Bestimmung der Laufparameter erfolgte durch die Methode der (multiplen) linearen Regression, wobei das Effekt der Bewegungsausführung mit den Parametern a, b, c sich durch:

$$\begin{pmatrix} dx \\ dy \\ d\omega \end{pmatrix} = \begin{pmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 \\ \alpha_8 & \alpha_9 & \alpha_{10} & \alpha_{11} \end{pmatrix} \begin{pmatrix} 1 \\ c \\ b \\ a \end{pmatrix}$$

bestimmen lässt.

5.3 Sensormodel

Bei diesen Tests wurden die funktionalen und nicht-funktionalen Eigenschaften der verschiedenen Sensormodell-Implementierungen getestet. Dabei ging es vor allem um die Wahl zwischen mehreren alternativen Implementierungen so wie: *Fester gegen adaptiven Fehlervarianz-Wert (s. 4.3.4):*



Direkte Implementierung gegen die modifizierte Implementierung(s.4.3.7):

Durchschn.Laufzeit der Methode(ms) /Durchschn.Fehler(cm) (Globale S.-l.)	Mit Fließkommazahlen	Mit Festkommazahlen
Direkte Implementierung	70ms/43	25ms/44
Modifizierte Implementierung	100ms/18	45ms/20

5.4 Sampling Funktion

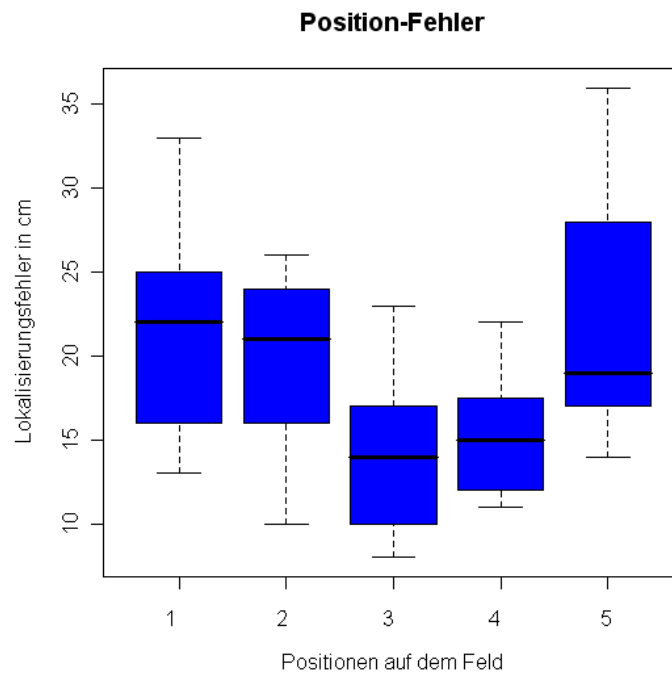
Vergleich der beiden Sampling-Verfahren (s.4.5):

Laufzeit, Matlab	Über die kumulierte Verteilung	Über die Verschmelzung der zwei Verteilungen
100	0.01	0.01
200	0.01	0.02
1000	0.02	0.06

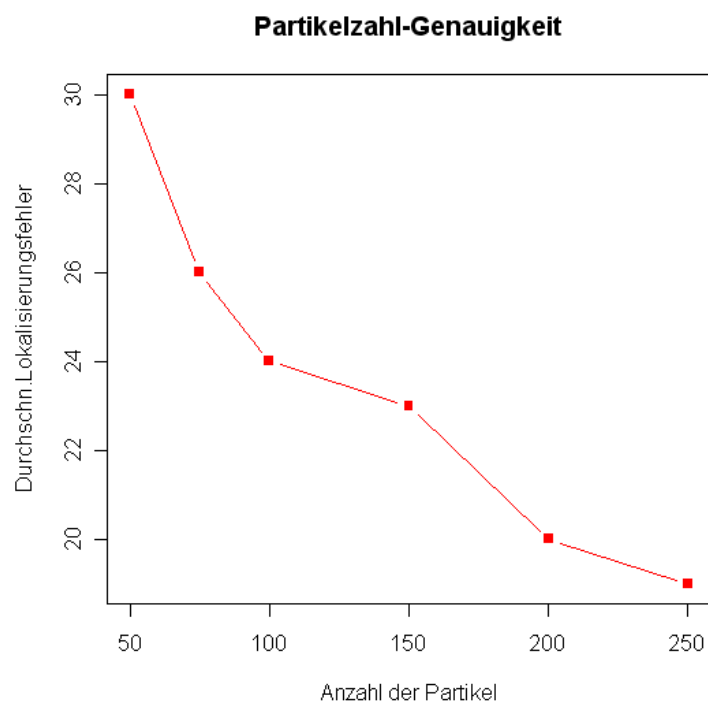
5.5 Systemtests

5.5.1 Globale Selbstlokalisierung

Getestet wurde die im Kapitel 4.2.2 beschriebene Implementierung des globalen Lokalisierungsverfahrens. Der Testablauf gestaltet sich denkbar einfach: die Roboter wurden an verschiedenen Positionen auf dem Feld platziert und die Selbstlokalisierungsroutine wird ausgeführt. Bei diesem Test wurden die auf der Abbildung 5.1 präsentierten Äquivalenzklassen übernommen, die Orientierung wurde zufällig gewählt.



Die Tests haben die im Abschnitt 4.3.6 präsentierte Überlegung bzgl. der Abhängigkeit der Genauigkeit der Schätzung von dem Abstand zu den Landmarken bestätigt. An der Position 1 wurden die schlechtesten Ergebnisse erzielt (15-25 cm Lokalisierungsfehler). Interessanterweise stellte sich auch die Position 5 (am nächsten zum Tor) als problematisch heraus, was jedoch auf die Fehler bei der Objekterkennung zurückzuführen (Säule nicht erkannt, das große Tor wurde geteilt) ist. Der durchschnittliche Fehler betrug in etwa 19 cm. Ein absolutes Versagen der Lokalisierungsmethode ist während der Tests nicht aufgetreten.



Außerdem wurde die Abhängigkeit der Schätzungsgenauigkeit von der Partikelanzahl untersucht. Erwartungsgemäß stieg die Genauigkeit mit der steigenden Anzahl der Partikel.

Auch die Laufzeit der Lokalisierungsmethode wurde getestet:

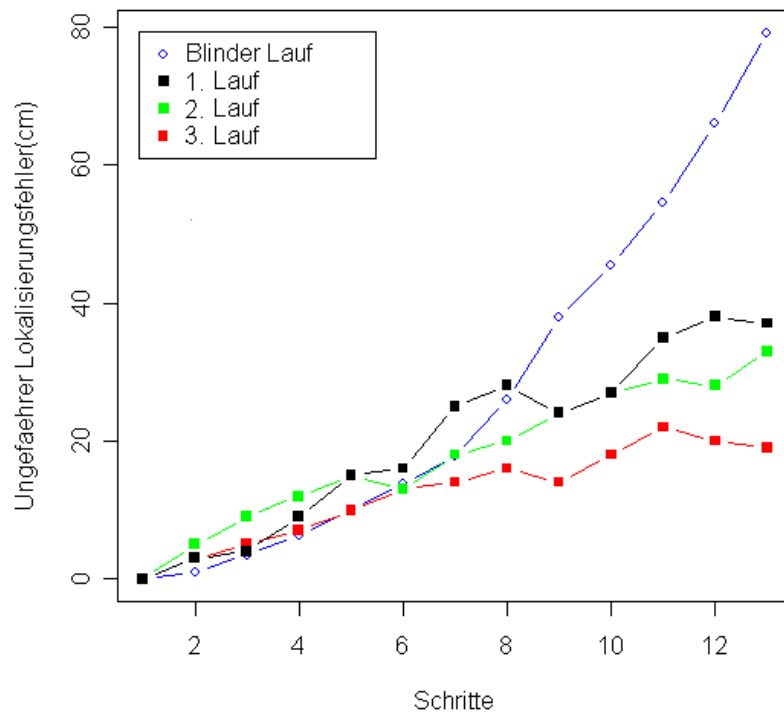
Laufzeit/ Lokalisierungsfehler	Nur die letzten 3 Objekte einbeziehen	Jeweils alle Objekte einbeziehen
4 Landmarken, 4 Bilder	1.55s/20cm	1.58s/15cm
5 Landmarken, 4 Bilder	1.58s/15cm	1.62s/10cm
5 Landmarken, 5 Bilder	2.13s/10cm	2.19s/10cm

Wie aus der oberen Tabelle ersichtlich ist, beträgt die durchschnittliche Lokalisierungszeit fast 2 Sekunden, wobei die meiste Zeit bei der Positionierung der Kamera verbraucht wird. Die Laufzeit lässt keine häufige Durchführung der globalen Selbstlokalisierung zu, zumal diese als selbstständiges Verhalten ausgeführt wird und alle anderen Verhalten werden für die Zeit angehalten.

5.5.2 Lokale Selbstlokalisierung

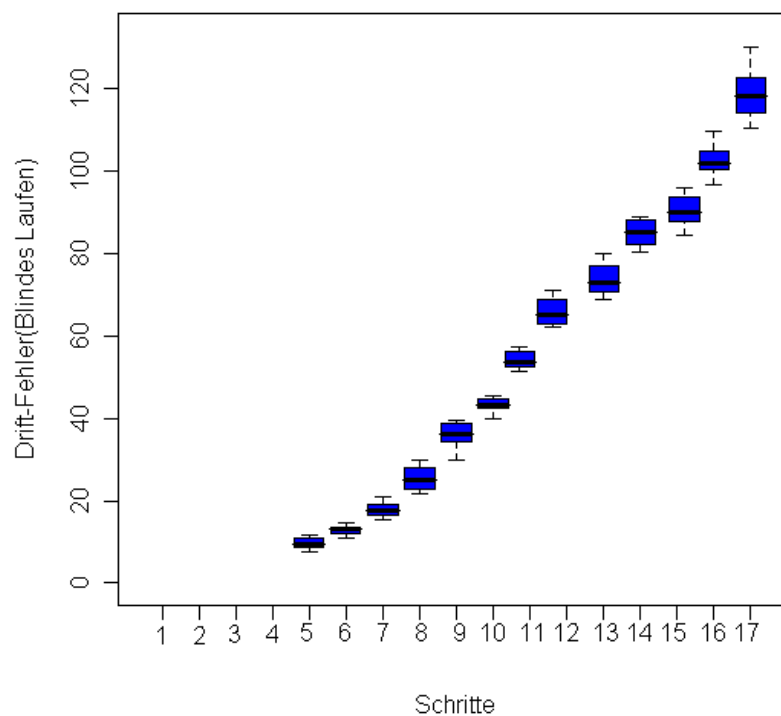
Das Testen der lokalen Selbstlokalisierung erwies sich als wesentlich schwieriger durchzuführen. Problematisch ist vor allem die Ermittlung der Sollwerte zu den verschiedenen Zeitpunkten. Dafür muss die Trajektorie des Roboters mit den dazugehörigen Zeitmarkierungen(Schritte) aufgezeichnet werden. Da die Tests manuell durchgeführt wurden, handelt es sich bei den nachfolgenden Daten um Schätzungen.

Für diesen Test musste der Roboter eine vorgegebene Bewegungsabfolge ausführen. Die Initialposition ist bekannt. Die Trajektorie wurde aufgezeichnet(mit einem Marker) und mit der von dem Roboter berichteten Trajektorie verglichen.



Die obere Abbildung präsentiert den berechneten Fehler. Die blaue Linie zeigt, wie der sich Fehler beim „blinden“ Laufen(ohne Landmarken) beliebig hoch akkumuliert, während gelegentliches Eingreifen des Sensormodells den durchschnittlichen Fehler auf 30 bis 35 cm beschränkt. Auf dem Bild ist die Entwicklung der seitlichen Abweichung abgebildet, bei den anderen Komponenten des Fehlervektors(Distanzfehler, Orientierungsfehler) sind ähnliche Kurven herausgekommen.

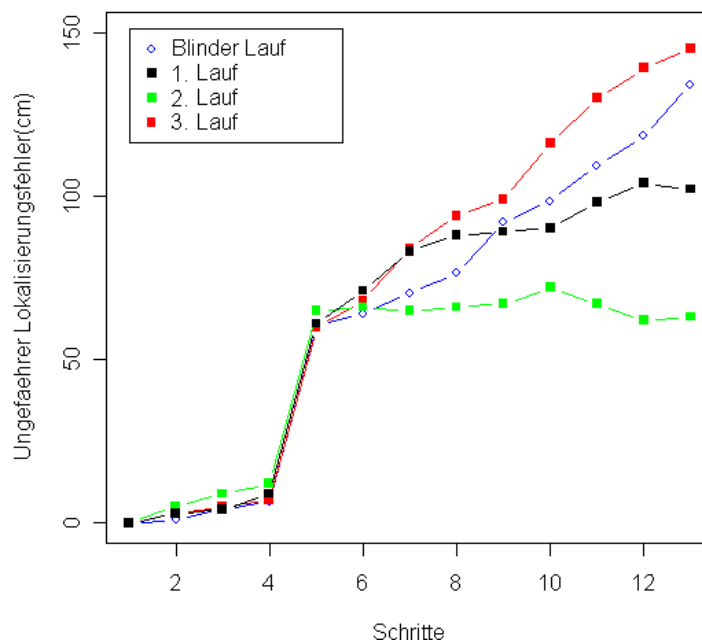
Die nachfolgende Abbildung zeigt die Drift-Fehlerentwicklung beim blinden gradlinigen Laufen, (20 Laufversuche):



Wie man deutlich sehen kann, beträgt der durchschnittliche Driftfehler bereits nach 10 Schritten über 40 cm. Aus diesem Grund sind längere Phasen des „blinden“ Laufens, wenn man nur auf das Bewegungsmodell angewiesen ist, nicht sinnvoll und können, wie der nächste Abschnitt zeigt, zu nicht reparablen Fehleinschätzungen der Position führen, wobei der Misstand dann nur durch die Reinitialisierung der Partikelmenge behoben werden kann. Aus diesem Grund stellt das Lokalisierungsmodul sicher, dass nach maximal 5 Schritten die Partikelmenge von dem Sensormodell bewertet wird, auch wenn dafür das aktuell ausgeführte Verhalten zwecks Informationsbeschaffung angehalten werden muss.

5.6 Kidnapped Robot

Nach den positiven Ergebnissen der funktionalen Systemtests wurde die Robustheit der Implementierung getestet. Wie bereits im Abschnitt 3.1 erläutert, eignet sich das Kidnapped-Robot-Problem als der ultimative Robustheitstest einer Lokalisierungsmethode, wobei das „Entführen“ des Roboters als Simulation gravierender Fehleinschätzungen dient. Auch dieser Test wurde manuell durchgeführt, indem die Trajektorie des Roboters per Hand vermessen werden musste. Die „Entführung“ erfolgte nach 4 Schritten durch die künstliche Verschiebung der Partikelwolke um 50 cm seitwärts. Der nachfolgende Graph zeigt das Ergebnis des Experiments:



Bei der Verfolgung der Bewegungen von der Partikelwolke mithilfe des Visualisierungsprogramms (Abb. 5.6.1), konnte man beobachten wie die Partikel in die Richtung des nächsten größeren Maximums der Likelihood-Funktion drifteten, wo sie meistens auch blieben. Auch wenn einige wenige Partikel zur richtigen Position gelangen konnten, wurden sie früher oder später von der Hauptmasse der Partikel an sich gezogen. Zum Glück wurde die Situation meistens erkannt und die Prozedur der globalen Selbstlokalisierung wurde gestartet. Nichtsdestoweniger gehört die Problematik zu den größten Schwächen der regulären MCL insgesamt sowie der implementierten Methode insbesondere. Abhilfe könnte allerdings eine drastische Erhöhung der Partikelzahl schaffen, was bei der aktuellen Roboterausstattung leider nicht zulässig ist.

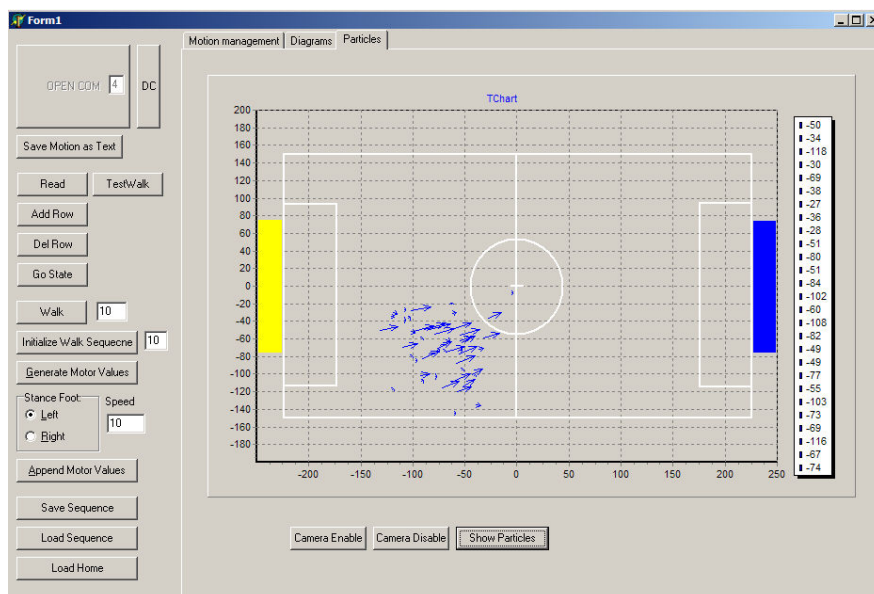


Abb.5.6.1: Das Visualisierungsprogramm

Kapitel 6

Ergebnis und Ausblick

6.1 Ergebnis

Die vorliegende Diplomarbeit beschäftigt sich mit dem Entwurf und der Implementierung einer Selbstlokalisierungsmethode für die humanoiden Fußballroboter in der RoboCup Umgebung. Es wurde nach einer gründlichen Betrachtung von zwei klassischen Verfahren - Kalman Filter und Monte-Carlo-Lokalisierungsmethode – im Hinblick auf die spezifische Anforderungen der Hardwareausstattung der Roboter die flexiblere der beiden Methoden ausgewählt. Speziell haben wir uns für die SIR-Variante der MCL entschieden. Die SIR-MCL stellt ein mächtiges Werkzeug der Zustandsschätzung dar und bietet gleichzeitig eine besondere Freiheit hinsichtlich der Modellierungsentscheidungen. Ein großer Nachteil der Methode hingegen ist die speicheraufwendige nicht-parametrische Repräsentation der Zustandsschätzung durch eine gewichtete Stichprobe, weswegen ein besonderer Augenmerk für uns auf der Effizienz der Implementierung lag. Auch die nicht-funktionale Anforderungen wie die Robustheit der Methode spielten eine wichtige Rolle, wobei die naheliegende Lösung zur Steigerung der Robustheit durch die Vergrößerung der Partikelmenge einen erhöhten Speicherverbrauch voraussetzt und damit im Widerspruch zur Speichereffizienzanforderung steht. Dieses Konflikt ist erfolgreich aufgelöst worden, wie die abschließenden Systemtests zeigen.

Eine besondere Herausforderung war die effiziente Integration des Moduls in das Steuerungsprogramm des Roboters. Die starke Verflochtenheit des Lokalisierungsmoduls mit praktisch allen anderen Programmteilen führte zur Entstehung eines komplexen Abhängigkeitsnetzes, wobei auch hier ein Kompromiss zwischen der in Anspruch genommenen Prozessorzeit und der Genauigkeit/Robustheit der Schätzung gefunden werden musste. Insgesamt weist das Programm das „best effort“- Verhalten auf, d.h. die zur Verfügung stehende Ressourcen werden komplett ausgeschöpft und die funktionale Tests zeigen dann inwiefern das Ergebnis ausreichend ist. Diese bezeugen sehr gute Ergebnisse bei der globalen Selbstlokalisierung und immerhin gute Resultate, was die lokale Selbstlokalisierung angeht. Verantwortlich für das schlechtere Abschneiden der Tracking-Methode ist ein Problem prinzipieller Natur, das darin besteht, dass es keine Garantien bzgl. der Messdatenqualität- und Menge gibt. Dem Konzept nach soll die lokale Selbstlokalisierung beiläufig stattfinden, während andere

Programmmodule Kontrolle über die Kamera haben und mit ihren eigenen Messungen beschäftigt sind. Das resultiert in den längeren Phasen des „blinden“ Trackings, wobei man nur auf die (unzuverlässige) Odometrie angewiesen ist. Durch die Aufweichung des Beiläufigkeits-Konzepts (s.4.2.3) konnte man schließlich doch noch zufriedenstellende Ergebnisse erzielen. Eine zusätzliche Verbesserung versprechen wir uns von der geplanten Implementierung der Linienerkennung, mit denen man viel mehr Orientierungsobjekte zur Verfügung hätte.

Zu den im wesentlichen nicht gelösten Problemen gehört das harte Kidnapped-Robot-Problem, da sich die Methode als anfällig für die lokalen Maxima der Likelihood-Funktion erwiesen hat. Die Kidnapped-Robot-Konstellationen hat man in der Praxis als Folge größere Fehleinschätzungen der vorausgegangenen Schritte. Man beugt dem Problem vor, indem man besonders hohe Anforderungen an die Erstlokalisierung stellt und dadurch von vorne herein so nah an der richtigen Lösung zu landen versucht, wie möglich. Zusätzlich versucht man die entsprechenden Situationen zu erkennen, indem man das Ergebnis auswertet und, falls man verdacht hat, zu weit von dem richtigen Ergebnis abgedriftet zu sein, fängt man wieder von vorne an. Leider bedeutet das, dass auch richtige Schätzungen verworfen werden können, denn man kann schlechte Messungen von schlechten a priori Schätzungen prinzipiell nicht unterscheiden.

Das Ergebnis der Implementierung lässt sich nur im Gesamtkontext richtig bewerten. Die Entwicklung der Lokalisierungsmethode erfolgte parallel zu anderen Programmteilen nach dem Spiralmodell-Prinzip und es hat mehrere Anläufe gebraucht bis das Ergebnis stimmte. Die letzten „Feldversuche“ waren erfolgreich, obwohl die Entwicklung mancher Module noch nicht abgeschlossen ist. Die prinzipielle Möglichkeit der Selbstlokalisierung mit den uns zur Verfügung stehenden Ressourcen wurde gezeigt. Aber letztlich ist eine erfolgreiche RoboCup-Teilnahme das erklärte Ziel des Projekts und erst im Vergleich mit den anderen Teilnehmer lässt sich das Ergebnis des Projekts insgesamt und der Lokalisierungsmethode insbesondere abschließend bewerten.

6.2 Ausblick

Wie bereits mehrmals angesprochen, hoffen wir darauf, dass die Fähigkeit Linien zu erkennen uns bei der Lösung des Tracking-Problems weiter bringt. Im Moment unterstützt unser Vision-System keine Linienerkennung. Problematisch daran ist vor allem die Übertragung der Ergebnisse von dem Kameramodul zum

Hauptkontroller, denn man kann die Linien ja kaum in der Bounding-Box-Form darstellen. Überhaupt ist die Übertragung der hochauflösenden Linieninformation in irgendeiner Form über die serielle Schnittstelle so gut wie ausgeschlossen. Auch wenn die Übertragung der Daten möglich wäre, wäre die Linienidentifikation für unseren Mikrokontroller viel zu aufwendig. Deswegen haben wir uns überlegt die Bilddaten soweit herunterzusampeln, dass das Problem des Datenvolumens nicht mehr besteht. Allerdings stellt sich die Frage, was wir mit den Daten dann noch anfangen können. Irgendwie müssen wir ja den Likelihood-Wert eines gegebenen Partikels nur mit einer Handvoll (geplant sind bspw. $16 \cdot 12 = 192$) von Pixel einschätzen. Die ersten Experimente haben ergeben, dass das möglich ist. Dafür wird das Kamera-Bild für die gegebene Pose anhand des entsprechend angepassten Weltmodells simuliert und mit dem gemessenen und heruntergesampelten Bild verglichen. Das gemessene Bild ist ein 16×12 Zellen großes Array mit 0 und 1 darin, wobei eine 1 für „Linie“ steht. Das simulierte Array ist weniger restriktiv und enthält die Wahrscheinlichkeiten dafür in der vorgegebenen Pose an einer bestimmten Bildposition eine Linie vorzufinden.

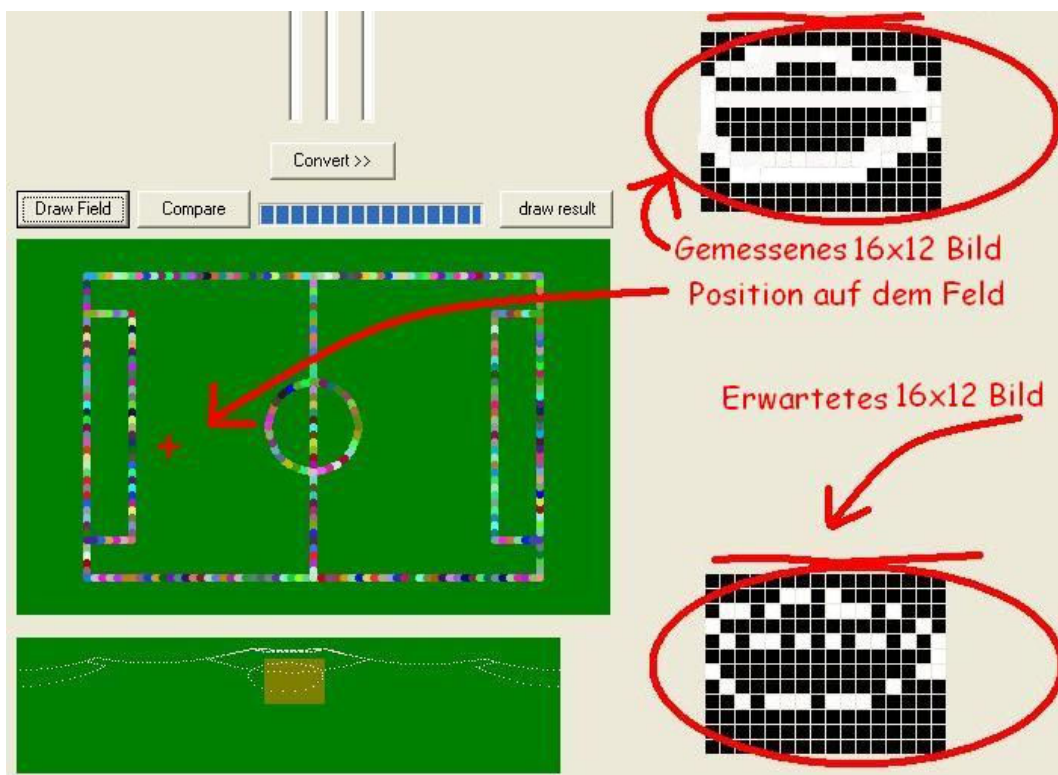


Abb. 6.1: Experimentelles Positionsschätzungsprogramm

Trotz der prinzipiellen Machbarkeit müssen noch einige schwerwiegende Probleme gelöst werden, bis man sich an die Implementierung wagen kann. Dazu

gehört, z. B. eine speicherschonende Liniendarstellung im Weltmodell oder, was noch sehr viel problematischer ist, eine effiziente Bildsimulation für eine gegebene Pose. Außerdem kann das Verfahren erst nach der Implementierung erschöpfend getestet werden und es ist nicht ausgeschlossen, dass man dann weitere Nachteile der Methode feststellen muss. Sollte das Vorhaben allerdings klappen, dann würde eine Kombination der alten und der neuen Methoden ein robustes und mächtiges Lokalisierungswerkzeug darstellen.

Ein weiterer Ansatz zur Erhöhung der Landmarken-Entdeckungsfrequenz ist die Vergrößerung des Sichtwinkels des Roboters durch den Einsatz von Linsen. Bereits mit dem Sichtfeld von etwa 80° in der horizontalen und vertikalen Richtung würde man sicherstellen, dass man von nahezu beliebiger Feldposition aus 3 Messungen durch eine Aufnahme durchführen kann. Das würde eine spezielle Routine der globalen Selbstlokalisierung unnötig machen, allerdings würde dadurch die Erkennung von kleineren Objekten (bspw. eines weit entfernten Balles) problematischer.

Eine weitere Verbesserung des Sensormodells wäre die Hinzunahme der y-Positionswerte von den Landmarken im Bild. Dafür muss ein Körperhaltungsmodell entwickelt werden, mit dessen Hilfe vor allem die frontale Neigung kompensiert werden kann. Denkbar wäre an der Stelle der Einsatz eines Neigungssensors. Durch den Einsatz eines Kompass ließe sich die Orientierung des Roboters bestimmen oder zumindest die Orientierungshypothesen verifizieren.

Bis jetzt gingen wir davon aus, dass der Roboter auf die Informationen angewiesen ist, die er selbst sammelt. In vielen Fällen können aber zusätzliche Daten dem Roboter über WLAN übermittelt werden. Am Anfang des Spiels, sowie nach Spielunterbrechungen kann dem Roboter seine Initialposition von außerhalb des Feldes mitgeteilt werden und während des Spiels kann einer der anderen Roboter zusätzliche Messdaten liefern. Insbesondere der Torwart hat meist einen guten Feldüberblick und kann den Spielern bspw. die Position des Balls zuschicken, mit dessen Hilfe sie ihre Posenschätzungen verifizieren können.

Wie man aus den Ergebnissen der Tests sieht, kommt die lokale Selbstlokalisierung mit 100 Partikel ziemlich gut zurecht, während die globale Selbstlokalisierung mit der zunehmenden Partikelzahl mit 200 Partikel wesentlich zuverlässiger funktioniert. Die naheliegende Idee ist, die Partikelzahl in der Phase der globalen Selbstlokalisierung vorübergehend zu erhöhen, indem man den Speicherplatz von anderen Modulen „ausleiht“. Nachdem die Erstlokalisierung erfolgt ist, übernimmt man die 100 besten Partikel und gibt den Speicher wieder frei. Weil die globale Selbstlokalisierung nur in Ausnahmefällen durchgeführt

wird, entsteht dadurch auch kein großer Mehraufwand, allerdings muss man dafür ganz genau wissen, welche Module und zu welchem Zeitpunkt über freien Speicherplatz verfügen.

Die Tests auf dem experimentellen Spielfeld haben ergeben, dass, solange das Lokalisierungsmodul ausreichend viele Orientierungsdaten bekommt, die Methode meist gute Positionsschätzungen liefert. Sollte es sich jedoch im Laufe des Wettbewerbs herausstellen, dass man ein zuverlässigeres Verfahren braucht, so lässt sich die Methode durch den Einsatz einer der zahlreichen Weiterentwicklungen von MCL entsprechend modifizieren. Zu erwähnen wäre an der Stelle bspw. die Mixture-MCL²¹, bei der die Schätzung einmal direkt mit der regulären MCL und einmal mit der Dual-MCL, die die Rollen von proposal Verteilungsfunktion (als Sampling-Funktion) und der Gewichtung (als Likelihood-Funktion) vertauscht, durchgeführt wird. Die beiden Schätzrichtungen haben für sich genommen ihre Vor- und Nachteile, während eine Kombination von den beiden einige Problemstellen der regulären MCL behebt, wodurch die Robustheit und Genauigkeit des Algorithmus erhöht werden.

²¹ Robust Monte Carlo Localization for Mobile Robots
Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert

Literaturverzeichnis

[1] Robust Monte Carlo Localization for Mobile Robots

Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert

[2] Monte Carlo Localization: Efficient Position Estimation for Mobile Robots (1999),

Dieter Fox, Wolfram Burgard, Frank Dellaert, Sebastian Thrun

[3] Robot Localization and Kalman Filters On finding your position in a noisy world,

Rudy Negenborn

[4] Kalman Filters and Robot Localization,

Rudy R. Negenborn

[5] Active global localisation for a mobile robot using multiple hypothesis tracking.

P. Jensfelt and S. Kristensen

[6] A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking, Arulampalam, S., Maskell, S., Gordon, N., Clapp, T

[7] Monte Carlo filter and smoother for non-Gaussian nonlinear state space

Kitagawa, G.

[8] Filtering via Simulation: Auxiliary Particle Filters

Michael K. Pitt and Nen Shephard

[9] Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids

Wolfram Burgard and Dieter Fox and Daniel Hennig and Timo Schmidt

[10] Globale Selbstlokalisierung für mobile Service Roboter,

Joachim Weber

[11] An Improved Particle Filter for Non-linear Problems,

James Carpenter, Peter Clifford, Paul Fearnhead

[12] Kalman filter for vision tracking,

Erik Cuevas, Daniel Zaldivar and Raul Rojas

[13] Analysis by Synthesis, a Novel Method in Mobile Robot Self-Localization

Alireza Fadaei Tehrani, Raúl Rojas, Hamid Reza Moballegh, Iraj Hosseini and Pooyan Amini

[14] People Tracking with a Mobile Robot Using Sample-based Joint Probabilistic Data Association Filters

Dirk Schulz, Wolfram Burgard, Dieter Fox, Armin B. Cremers, 2003

[15] On Sequential Monte Carlo Sampling Methods for Bayesian Filtering,

Arnaud Doucet, Simon Godsill, Christophe Andrieu

[16] Architectures for Efficient Implementation of Particle Filters,

Miodrag Bolic

[17] Landmark-based Navigation of Industrial Mobile Robots,

Huosheng Hu And Dongbing Gu

[18] A Modified Particle Filter for Simultaneous Robot Localization and Landmark Tracking in an Indoor Environment,

Wenyan Hu, Tom Downs, Gordon Wyeth, Michael Milford, David Prasser

[19] Adapting Proposal Distributions for Accurate, Efficient Mobile Robot Localization

Patrick Beeson, Aniket Murarka, and Benjamin Kuipers

[20] Mathematical Foundations of Navigation and Perception for an Autonomous Mobile Robot

James L. Crowley

[21] Combining Kalman Filtering and Markov Localization in Network-Like Environments

S. Thiebaut, P. Lamb