Selbstlokalisierung humanoider Fußball-Roboter auf der Basis von Feldlinien

Roman Schulte-Sasse
r.schulte-sasse@fu-berlin.de

Bachelorarbeit
Freie Univesität Berlin
Fachbereich Mathematik und Informatik

Gutachter
Prof. Dr. Raúl Rojas
Dr. Hamid Reza Moballegh

Berlin, den 11.10.2013

Zusammenfassung

Während die meisten Menschen unbewusst und scheinbar ohne jede Mühe ihre eigene Position im Raum abschätzen können, stellt diese Problematik in der Robotik trotz intensiver Forschung nach wie vor eine Herausforderung dar. Damit ein Roboter im Team agieren und effektiv spielen kann, muss er seine eigene Position auf dem Spielfeld kennen. Partikelfilter bilden einen klassischen Ansatz, um das Problem der Selbstlokalisierung zu lösen. Sie verwalten viele Hypothesen, an denen sich der Roboter befinden könnte und generieren aus ihnen regelmäßig eine Schätzung der aktuellen Position. Dafür werden unterschiedliche Sensoren des Roboters ausgewertet und zusammengeführt, wobei die von der Kamera erkannten Spielfeldlinien in dieser Arbeit die Grundlage der Selbstlokalisierung bilden. Diese Feldlinien eignen sich gut für das Problem, da sie in den meisten Spielsituationen teilweise sichtbar und von Regeländerungen im RoboCup unabhängig sind.

In dieser Arbeit wurde ein Partikelfilter entwickelt, welcher die erkannten Feldlinien mit allen Hypothesen vergleicht. Eine Metrik, die hieraus die wahrscheinlichsten Positionen ermitteln kann, sowie diverse Erweiterungen des klassischen Partikelfilters werden vorgestellt.

Abstract

While most humans have the ability to always know where they are, that task seems to be a challenging one for humanoid robots. Even after almost a decade of research, that problem remains and even becomes more challenging when rules change and less distinct features are available on the field of play.
However, self localization is an important feature for any soccer player when it comes to the ability of team play or efficient goal approaches.
Particle filters have become the standard solution to solve the problem by maintaining multiple hypotheses about the current robot's location and extracting the most probable one from that population. In order to do this, the particle filter uses different sensoric data from the robot, while focussing mainly on the images from the robot's camera.

This bachelor thesis focusses on the use of fieldlines to determine a particle's fitness. A particle filter with various extensions has been developed that uses only fieldlines and odometry data to solve the problem of self localization. For this, a metric has been introduced that determines a particle's fitness by the use of fieldlines.

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Berlin, den 11.10.2013

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit der Selbstlokalisierung humanoider Roboter in einer stark limitierten Umgebung.

1.1 Motivation und Problemstellung

> „It is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility.“[16]

Selbstlokalisierung ist ein Paradebeispiel für Probleme, welche Menschen extrem gut lösen können, während Roboter nur unter großen Mühen eine ungenaue Approximation liefern können. Sich auf einem Spielfeld beim Fußball zu orientieren und auf das richtige Tor zu spielen ist selbst für ein vier-jähriges Kind eine vergleichsweise einfache Aufgabe, während Roboter dafür rechenintensive statistische Modelle benötigen.
Wie kommt es also, dass das Kind noch nie von einer rekursiven Zustandsschätzung gehört hat, gleichzeitig aber seine eigene Position im Raum besser bestimmen kann, als jeder Roboter? Um diese Frage zu beantworten, versucht die Wissenschaft der künstlichen Intelligenz Systeme wie die Selbstlokalisierung nachzubauen, in der Hoffnung daraus Erkenntnisse über die Funktionsweise dieser Mechanismen beim Menschen ziehen zu können.

Für das Szenario der Fußball spielenden Roboter ist die Lokalisierung ebenfalls sehr wichtig, da sie die Basis für jegliche Team-bezogene Strategie ist. Ein Ziel für die nahe Zukunft ist es beispielsweise, die Spieler untereinander Informationen austauschen zu lassen. Dazu gehört die aktuelle Ballposition in absoluten Koordinaten, die eventuell für den einen Roboter leicht zu sehen ist, während ein anderer den Ball aufgrund eines Hindernisses nicht erkennen kann.
Die Selbstlokalisierung humanoider Fußballroboter ist also sowohl rein wissenschaftlich von größtem Interesse als auch für den Wettbewerb von spielerischem Vorteil. Je akkurater das Problem gelöst werden kann, desto besser können andere Teile der Software darauf aufbauen. Durch eine gute Schätzung der eigenen Position auf dem Spielfeld lässt sich nicht nur ein Teamplay realisieren, sondern auch das Weltmodell der Roboter stark verbessern und Eigentore können verhindert werden.

1.2 Der RoboCup und die FUmanoids

Die Selbstlokalisierung soll im Rahmen der Fußball-Ligen des *RoboCups* stattfinden, einem Wettbewerb, in dem Universitäten und Forschungseinrichtungen Roboter gegenein-

ander antreten lassen. Ziel des RoboCups ist eine Beschleunigung der Forschung und ein jährlicher Wissensaustausch im Bereich Robotik und künstliche Intelligenz.
Fußball ist dabei nur eine von mehreren Sparten des RoboCup, andere versuchen beispielsweise Bergung und Rettung bei Naturkatastrophen mit Robotern möglich zu machen oder alltägliche Aufgaben im Haushalt zu automatisieren.[1]

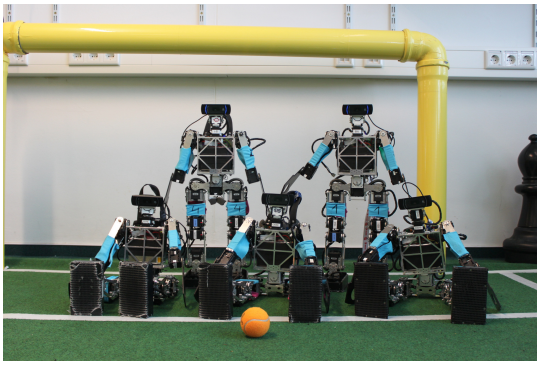


Abbildung 1: Das Team der FUmanoids in Eindhoven 2013)

RoboCupSoccer ist der Wettbewerb des RoboCups, in dem gegeneinander Fußball gespielt wird. Die Idee dafür kommt aus dem Jahre 1993, in dem eine Gruppe von zunächst japanischen Wissenschaftlern einen Wettbewerb für humanoide Roboter gründen wollte. Tatsächliche Gründung des Wettbewerbs fand im Jahre 1997 statt, dem Jahr in dem der damals amtierende Schachweltmeister Garri Kasparow von IBMs Computer Deep Blue geschlagen wurde.

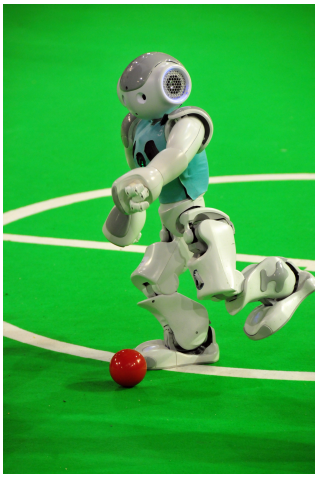


Abbildung 2: Ein NAO-Roboter beim Schuss

Nach diesem Ereignis musste eine neue Herausforderung für die Wissenschaft der künstlichen Intelligenz gefunden werden, denn als wirklich intelligentes System konnte der Schachcomputer Deep Blue nicht aufgefasst werden. Es stellte sich heraus, dass es ein viel größeres Problem darstellt, einen Roboter in einer Umgebung agieren zu lassen, die nicht oder nur schwer erfassbar ist, als ihn ein Problem wie Schach lösen zu lassen. Denn hier muss ständig mit Unsicherheiten umgegangen werden und die Wahrnehmung ist selten zu 100 Prozent akkurat.
Ein Sport als neue Zielsetzung war eine naheliegende Entscheidung, da dort die dynamische Umgebung mit motorischen Herausforderungen gewährleistet ist und gleichzeitig Ergebnisse und Fortschritte relativ einfach zu messen sind.

[1] siehe www.robocup.org für weitere Informationen

Fußball eignet sich gut, da der Sport sehr bekannt ist und gleichzeitig auf sehr verschiedenen Niveaus gespielt werden kann. So müssen die Teams in der Anfangsphase des RoboCups noch nicht mit hoch gespielten Bällen umgehen, wie es in fast allen Rückschlagspielen erforderlich wäre. Dennoch existiert das Potential für ein sehr hohes motorisches Niveau in einer mit Unsicherheiten behafteten Umgebung.

Das Ziel der Initiative RoboCup ist es, dass bis 2050 humanoide Roboter in der Lage sein sollen, die amtierenden FIFA-Weltmeister zu schlagen.

> „By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup.“[2]

Diese Setzung der erfolgreichen Teilnahme an einem Sport als Benchmark für die Forschung in der künstlichen Intelligenz allgemein steht beabsichtigt im direkten Widerspruch zum vorherigen Ansatz, dass Probleme gelöst werden müssen, die auch für Menschen schwierig sind.

1.3 Ziel dieser Arbeit

Das zu lösende Problem ist also die Findung der aktuellen Position eines Roboters auf dem Spielfeld. Je nach eingebauter Sensorik kann diese Aufgabe mehr oder weniger schwer sein. So kann beispielsweise ein Roboter mit verbautem GPS-Modul und Kenntnis über die Umgebung seine Position relativ einfach bestimmen, während dies für die im Roboterfußball eingesetzten Maschinen deutlich schwieriger ist.

Denn hier sind nur passive Sensoren wie Kameras und Berührungssensoren erlaubt, aktive Systeme wie Laser, Radar oder GPS sind verboten, damit die gesamte Sensorik der Roboter „menschlich“ ist.

Ziel dieser Arbeit ist es, das Problem der Selbstlokalisierung über einen beliebig langen Zeitraum hinweg mithilfe eines Partikelfilters zur Zustandsschätzung zu lösen, wobei eine fehlerbehaftete Messung Y in jedem Zeitschritt vorliegt. Diese besagt, wie weit sich der Roboter seit dem letzten Zeitschritt bewegt hat. Zur Korrektur der Messungen werden die analysierten Kamerabilder hinzugezogen, insbesondere die momentan erkannten Feldlinien. Diese beiden Messungen sollen so miteinander kombiniert werden, dass am Ende die aktuelle Position des Roboters geschätzt und auch die Bewegung mitverfolgt werden kann.

Nach der Definition von sowohl dem Problem als auch der Art der gesuchten Lösung im ersten Kapitel (1) wird sich der darauf folgende Abschnitt 2 zunächst mit den grundle-

[2] http://www.robocup.org/about-robocup/objective/

genden mathematischen Modellen beschäftigen, die für diese Arbeit vonnöten sind.
Wir werden hier sehen, dass alle besprochenen Grundlagen die Basis für den in dieser Arbeit verwendeten theoretischen Ansatz, den *Partikelfilter*, bilden. Ein tieferes Verständnis von dessen Funktionsweise erfordert dabei eine Einführung in die ihm zugrunde liegenden Modelle.
Anschließend werden verwandte Arbeiten sowie ihre Probleme und Lösungswege in Kapitel 3 diskutiert, bevor Kapitel 4 die eigentliche Umsetzung mit all ihren Erweiterungen des ursprünglichen Partikelfilters vorstellt. Hier wird auch auf eine Metrik zum Vergleich von Feldlinien detailliert eingegangen.
Abschließend werden Testergebnisse und Resultate in Kapitel 5 analysiert und mit einem Ausblick auf mögliche Verbesserungen der entwickelten Software in Kapitel 6 abgeschlossen.

2 Grundlagen

Das folgende Kapitel wird sich mit den theoretischen Grundlagen der Arbeit beschäftigen. Während hier die klassischen Algorithmen teilweise losgelöst von der Problematik der Selbstlokalisierung besprochen werden, wird sich Kapitel 4 mit der tatsächlichen Umsetzung und Anwendung auf das Problem beschäftigen und Erweiterungen vorstellen.

2.1 Mathematische Grundlagen

Das folgende Kapitel wird sich mit den mathematischen Grundlagen des Partikelfilters beschäftigen. Dabei werden vom Allgemeinen zum Speziellen hin Lösungen aus der Mathematik für bestimmte Teilprobleme vorgestellt, die dann in Kapitel 2.2 genutzt werden.

2.1.1 Markov-Ketten

Eine Markov-Kette ist ein stochastischer Prozess, in welchem eine Reihe von Zufallsvariablen (ZV) über eine Zeit hinweg untersucht werden, um zukünftige Zustände der Variablen vorauszusagen.[9] Dabei ist der Ergebnisraum der ZVs endlich; es handelt sich also um diskrete Zufallsvariablen. Zusätzlich erfüllt die Kette die *Markov-Eigenschaft*, welche besagt, dass der Zustand der Zufallsvariable zu einem Zeitpunkt t nur vom Zu-

Abbildung 3: Eine Markov-Kette

stand $(t-1)$ abhängt und nicht von früheren Zeitpunkten.
Formal lässt sich diese Eigenschaft schreiben als:

$$Pr(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = Pr(X_{n+1} = x | X_n = x_n) \qquad (1)$$

Dadurch sind Markov-Ketten bis zu einem gewissen Grad *memory-less*, besitzen also kein Gedächtnis an frühere Zustände der Variablen. Allerdings sind in den vorhergegangenen Zuständen ihre jeweiligen Vorgänger enthalten, weshalb sich teilweise die früheren Zustände rekonstruieren lassen. Wir werden später noch sehen, dass die Markov-Eigenschaft positive Auswirkungen auf den Speicherverbrauch der Selbstlokalisierung hat.

2.1.2 Hidden Markov Models (HMM)

Wir haben nun eine Basis für die Modellierung von aufeinander folgenden Zeitschritten einer Zufallsvariable durch die Markov-Kette. Zu einem gewissen Zeitpunkt t suchen wir

einen Vektor $\vec{p}$, der die Position des Roboters zu dieser Zeit beschreibt. Da es nicht möglich ist, diese Position direkt zu messen, bietet es sich an, sie als Zufallsvariable zu modellieren.
Diese Variable X_t ist dann die gesuchte Größe in dem Problem der Selbstlokalisierung.

Die vorhandenen Messdaten kommen aus verschiedenen Quellen und sind zu jedem Zeitpunkt fehlerbehaftet, weshalb das Modell der Markov-Kette hier nicht ausreichen kann. Allerdings lässt sich dieses Szenario durch ein *Hidden Markov Model* beschreiben, welches Aussagen über die Zustände in einem Prozess erlaubt, auch wenn diese nicht messbar sind.[6] Anstatt X_t direkt zu betrachten, schreiben wir unsere Markov-Kette in Abbildung 3 um, so dass X_t eine *latente Variable* oder *hidden variable* ist, also eine Größe, deren Wert im System existiert und wichtig ist, dessen Messung jedoch nicht direkt möglich ist.
Zu jedem Zeitpunkt t existiert dafür eine Messung Y_t, die mit der versteckten Variable in einem Zusammenhang steht. Es ist dadurch möglich, über ein Modell, welches die Wahr-

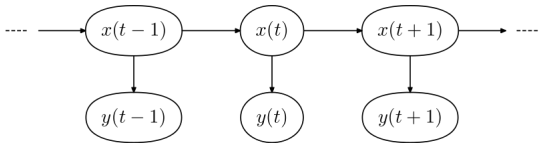


Abbildung 4: Beispiel eines Hidden Markov Models

scheinlichkeiten von Messfehlern abbildet, von Y_t auf X_t schließen. Dieser Zusammenhang von Messung und verstecktem Zustand des Systems muss gut modelliert werden, damit Aussagen über X_t gemacht werden können.

Ein Hidden Markov Model lässt sich schreiben als ein $5-Tupel$:

$$\lambda = (S, V, \psi, \phi, \pi) \tag{2}$$

mit

- $S = s_1, \ldots, s_n$, der Menge aller möglichen Zustände der latenten Variablen X_t.
- $V = v_1, \ldots, v_m$ der Menge aller Zustände der Messungen Y_t.
- $\psi : S \times S \to \mathrm{R}$, der Übergangsfunktion für X, definiert als: $\psi(s_i, s_j) = Pr(s_j | s_{1,\ldots i})$, gibt die Wahrscheinlichkeit an, von s_i nach s_j zu kommen.
- $\phi : S \times V \to \mathrm{R}$, definiert als $\phi(x_i, y_i) = Pr(y_i | x_i)$, gibt die Wahrscheinlichkeit an, in einem Zustand x_i die Beobachtung y_j zu machen.
- $\pi \in \mathrm{R}^n$, die Anfangsverteilung, also die Wahrscheinlichkeit, dass x_i der Startzustand des Modells ist.

Die Mengen S, V können äquivalent sein, denn sie beschreiben alle möglichen Messungen und Zustände von X_t. Für die Selbstlokalisierung beispielsweise könnten S, resp. V, alle möglichen Positionen auf dem Spielfeld darstellen.
Problematisch ist die Bestimmung der Übergangswahrscheinlichkeiten, denn diese können sich pro Zeitschritt t ändern. Aufgrund der Markov-Eigenschaft kann ψ sofort umgeschrieben werden als:

$$\psi(s_i, s_j) = Pr(s_j|s_{1,\dots i}) = Pr(s_j|s_i).$$

Da das HMM eine Markov-Kette zwischen den einzelnen X_t für $t = 1, 2, \dots$ annimmt, dann gilt zwischen ihnen die Markov-Eigenschaft, die besagt, dass ein Zustand immer nur von seinem Vorgänger abhängt.

Wenn das Modell und eine Reihe von Messungen Y_t bekannt sind, lassen sich die latenten Variablen schätzen. Gesucht ist also die Wahrscheinlichkeit, dass sich das System unter einer Reihe von gegebenen Messungen in einem bestimmten Zustand befindet:

$$Pr(X_t = x|y(1), \dots, y(t))$$

Für den Fall, dass die Wahrscheinlichkeitsverteilung der latenten Variable als Gauß-Verteilung vorliegt, ist es möglich, eine analytische Gleichung zur Berechnung des oben genannten *Posteriors* anzugeben.
Dadurch kann die Wahrscheinlichkeit für eine bestimmte Roboterposition in einem Kalman-Filter durch verschiedene Differenzialgleichungen analytisch bestimmt werden. Der Ansatz bleibt jedoch probabilistisch, da die Messdaten nur ungenau und somit als Wahrscheinlichkeiten vorliegen.

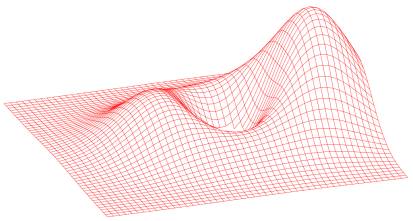


Abbildung 5: Beispielhafte Gauß-Verteilung der ZV X_t

Obwohl wir für die versteckte Variable durchaus eine Normalverteilung annehmen können, wird in dieser Arbeit von einer Lokalisierung durch einen *Extended Kalman Filter* abgesehen und mit dem Partikelfilter gearbeitet.
Statt also hier nach einer analytischen Lösung für das Inferenzproblem im Hidden Markov Model zu suchen, werden wir versuchen, mittels einer *Monte Carlo Simulation* pro Zeitschritt die Lösung zu approximieren.

2.1.3 Markov-Chain Monte Carlo Verfahren (MCMC)

Wir haben gesehen, dass zwar Modelle vorhanden sind, die unser Problem beschreiben können, jedoch ist es schwierig, die Selbstlokalisierung analytisch zu lösen.

Monte Carlo Verfahren bilden eine Möglichkeit, um von einer Wahrscheinlichkeitsverteilung effizient Stichproben oder *Samples* zu ziehen. Diese sind definiert als eine Menge

$$\{x^{(r)}\}_{r=1}^{R} \tag{3}$$

die aus der Wahrscheinlichkeitsverteilung unserer bekannten Zufallsvariable X_t, also Pr_{X_t}, gezogen werden.
Mithilfe dieser Stichproben wollen wir dann den Zustand von X_t in unserem diskreten Zustandsraum schätzen, wobei das Maximum der Wahrscheinlichkeitsverteilung von X_t gesucht ist, denn dort sollte sich der Roboter mit größter Wahrscheinlichkeit befinden.

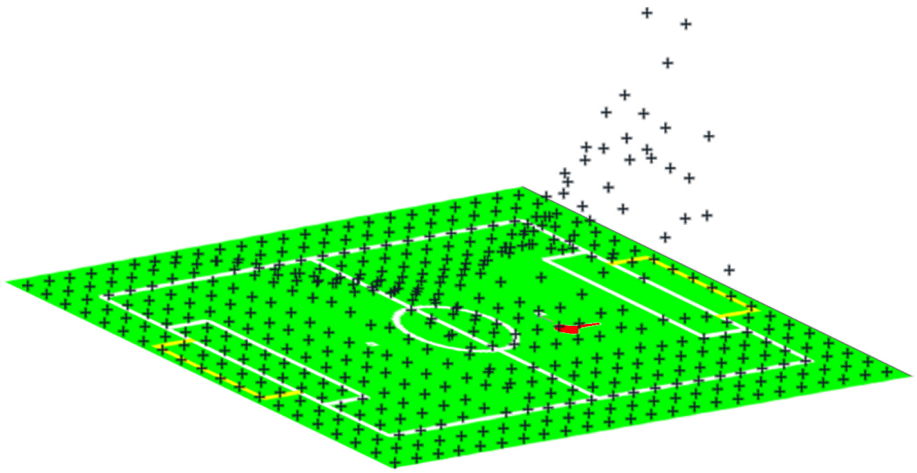


Abbildung 6: Wahrscheinlichkeitsverteilung über dem Spielfeld

Leider ist es einerseits schwierig, die Samples aus der Verteilung repräsentativ zu generieren und andererseits müssen sie so bewertet werden, dass sie auch tatsächlich die Wahrscheinlichkeitsverteilung von X_t abbilden.

Ziehen eines Samples Das Problem, aus einer Wahrscheinlichkeitsverteilung in einem prinzipiell großen Zustandsraum Samples zu ziehen, welche möglichst gut $arg\max(Pr_{X_t}(x))$ finden können, ist dann schwer, wenn wir die Verteilung über der Zufallsvariable X_t nicht kennen.[15]
Denn woher können wir wissen, ob ein Maximum gefunden wurde, wenn nicht bekannt ist, wie hoch dieses sein wird. Gleichzeitig ist es aus Laufzeitgründen oft nicht möglich, aus dem gesamten Zustandsraum Stichproben zu nehmen.
Angenommen wir wollen die Position des Roboters bei einer Feldlänge von 6×4 Meter auf 10 cm bzw. Grad genau errechnen, dann wäre es unumgänglich - um den gesamten Zustandsraum zu durchsuchen - $(600 \times 400 \times 36) = 8640000 = 8,64 * 10^6$ Stichproben

pro Zeitschritt zu analysieren.
Da diese Anzahl an Operationen aus Laufzeit-technischen Gründen in aller Regel nicht möglich ist und zusätzliche Informationen vorhanden sind, die es uns erlauben, nicht den gesamten Zustandsraum zu durchsuchen, wird eine andere Sampling-Strategie vonnöten sein. Diese sollte direkt nur an Orten Samples ziehen, welche eine hohe Wahrscheinlichkeitsdichte aufweisen, um damit eine Effizienzsteigerung gegenüber der klassischen Monte-Carlo-Simulation zu erreichen.[9]

Diese Strategie des *Importance Sampling* wird vom Partikelfilter zusammen mit der Markov-Eigenschaft der Kette der Zustände genutzt, um nicht unnötig viele Samples ziehen zu müssen, aber gleichzeitig eine gute Approximation von X_t zu erlangen.

2.1.4 Importance Sampling

Um also nur verhältnismäßig wenige Samples aus der Verteilung ziehen zu müssen, muss die Varianz unserer Stichproben $\{x^{(r)}\}_{r=1}^{R}$ geringer werden. Denn eine geringe Varianz würde bedeuten, dass wir mit den Samples die *wichtigsten* Positionen des Zustandsraums abdecken.[15]
Das bedeutet, dass die Standardabweichung des Durchschnitts der Stichproben von der tatsächlichen Verteilung minimal sein sollte, also:

$$\frac{1}{R}\sum_{i=1}^{R} Pr_{X_t}(x^{(i)}) \tag{4}$$

möglichst nah an der tatsächlichen Wahrscheinlichkeitsverteilung von Pr_{X_t} liegen sollte.

Wir können also Samples von unserer Verteilung ziehen, jedoch ist uns nicht bekannt, welche Punkte sich dafür eignen könnten.
Angenommen, dass Pr_{X_t} zu komplex ist, um die wichtigen Punkte zu bestimmen, ist also eine Funktion vonnöten, die Aufschluss über diese interessanten Punkte der Verteilung gibt. Diese *sampler density* $Q(x)$ muss keine perfekte Schätzung der eigentlich gesuchten Verteilung sein, sondern lediglich eine ungefähre Information über die wichtigen Punkte liefern.[15]
Je besser jedoch Q die Dichte approximiert, desto schneller werden die relevanten Punkte von Pr_{X_t} gefunden. Damit konvergiert die Monte-Carlo-Simulation deutlich schneller.

Die Samples werden nun von $P(x)$ gezogen, allerdings werden die Punkte der Wahrscheinlichkeitsverteilung an denen gesampled wird, anhand der Schätzung $Q(x)$ festgelegt.
Ist nun aber für einen Punkt x_i die sampler density $Q(x_i)$ höher als die Auswertung der eigentlichen Verteilung $Pr_{X_t}(x_i)$, so werden die Punkte um x_i *überrepräsentiert* sein. Gleiches gilt auch für den umgekehrten Fall, dass $P(x_i)$ größer als $Q(x_i)$ ist (siehe Abbildung 7).

Um diesen Fehler zu kompensieren, erhält jede Stichprobe ein Gewicht, welches die „Wichtigkeit“ des Samples für die Wahrscheinlichkeitsverteilung der Zufallsvariable X_t angibt. Das Gewicht ω berechnet sich aus dem Verhältnis der Schätzung Q zur tatsächlichen Auswertung am Punkt x_i:

$$\omega_i = \frac{Pr_{X_t}(x_i)}{Q(x_i)}. \tag{5}$$

Dadurch sind Samples mit hohen Gewichten immer auch bessere Estimates der Wahrscheinlichkeitsverteilung Pr_{X_t}.[15]

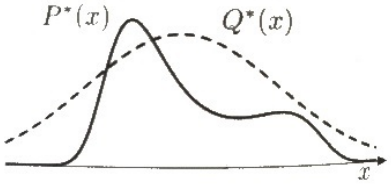


Abbildung 7: Importance Sampling mit nicht-optimale Dichte-Approximation $Q^*(X)$[15]

Die Idee des Importance Sampling in Monte-Carlo Simulationen lässt sich sehr gut mit Markov-Ketten und damit auch mit Hidden Markov Modellen zusammenführen, da wir $Q(X)_t$ aus dem letzten Zeitschritt $t-1$ übernehmen können.

2.2 Partikelfilter

Diese Zusammenführung von HMMs mit Monte-Carlo Approximation und Importance Sampling führt uns zum Konzept des Partikelfilters, welches von Dellaert et al. in [5] zuerst auf das Lokalisierungs-Problem bezogen wurde.

2.2.1 Einleitung

Partikelfilter sind eine *sequentielle Monte-Carlo Methode*, die schrittweise Monte Carlo Simulationen pro Zeitschritt durchführen. Ihnen liegt, wie bereits in Kapitel 2.1.2 erwähnt, ein Hidden Markov Modell (HMM) zugrunde, bei dem die Übergangswahrscheinlichkeiten dynamisch verändert werden (siehe Abbildung 4). Dabei gilt im Wesentlichen, dass für jeden Zeitschritt des HMMs die latente Variable X_t geschätzt werden muss und die Daten aus der letzten Iteration für die Schätzung herangezogen werden. Anschließend wird in jeder Iteration die Wahrscheinlichkeitsverteilung von X_t durch das Ziehen von Samples neu berechnet.

Die einzelnen Samples werden wir für den Filter fortan als *Partikel*, *Hypothesen* oder auch teilweise *Individuen* bezeichnen. Diese Begriffe sind zueinander äquivalent und dokumentieren die Herkunft der unterschiedlichen verwendeten Algorithmen.

Der Filter sollte jetzt zwei Probleme lösen: zum einen die globale Schätzung von X_1 zu einem Startzeitpunkt (*Kidnapped Robot*) an dem keine bisherigen Informationen zu einer

eventuellen Position des Roboters vorhanden oder diese nicht mehr gültig sind, etwa weil der Roboter von einem Schiedsrichter verstellt wurde.
Zum anderen die Schätzung der Position zum Zeitpunkt t auf Basis von $t-1$ (*Tracking*). Dabei sind Informationen über wahrscheinliche Zustände bekannt und die Grenzen, in denen sich der Roboter bewegt haben kann, sind sehr eingeschränkt.
Um beide Probleme zuverlässig zu lösen, müssen wir die aus dem HMM bekannten Funktionen ω bzw. ϕ so bestimmen, dass sich die Partikel immer um die wahrscheinlichen Zustände gruppieren und dennoch immer das globale Maximum finden.

Definition (Motion Model)**.** *Das Motion Model ψ beschreibt die Übergangswahrscheinlichkeit von einem Zustand x_{t-1} zu einem Zustand x_t.*

$$\psi := Pr(x_t|x_{t-1}) \tag{6}$$

Da bei der Lokalisierung ψ meist durch die errechneten Bewegungsdaten des Roboters gegeben ist, sprechen wir von Motion Model.

Gleichzeitig ist ψ auch die Approximation der Dichte von Pr_{X_t} für das Importance Sampling, also Q, denn es gibt Informationen über die wahrscheinlichen bzw. wichtigen Punkte der Wahrscheinlichkeitsverteilung von X_t (siehe auch Kapitel 2.1.4).

Weiterhin ist ein HMM definiert durch die Funktion ϕ, die in der Spezialisierung des Partikelfilters als *Observation Model* bezeichnet wird.

Definition (Observation Model)**.** *Das Observation Model beschreibt die Funktion $\phi(x)$ des HMM, also die Wahrscheinlichkeit für eine bestimmte Beobachtung unter der Annahme, sich in einem bestimmten Zustand zu befinden.*

$$\phi(x_t) = Pr(y_t|x_t) \tag{7}$$

Es wird herangezogen, um bestimmte Hypothesen zu überprüfen bzw. sie zu bewerten.

Das Ziel des Partikelfilters ist es, Inferenz auf den X_t der Markov-Kette für $t = 1, 2, \ldots$ zu betreiben. Wir können diese Schätzung von Pr_{X_t} folgendermaßen schreiben:

$$P(X_t = x) = P(X_t = x|X_{t-1}, X_{t-2}, \ldots, X_1) \tag{8}$$

Durch die Markov-Eigenschaft gilt:

$$P(X_t|X_{t-1}, X_{t-2}, \ldots, X_1) = P(X_t|X_{t-1}) = \psi(x_t) \tag{9}$$

Betrachten wir nun den Satz von Bayes:

$$P(x|y) = \frac{P(y|x) \cdot P(x)}{P(y)}, \tag{10}$$

Er erlaubt uns die Umformulierung unserer Gleichung, wobei wir annehmen können, dass der Nenner immer 1 ist, da die Messung Y_t in jedem Zeitschritt gegeben ist (siehe Abbildung 4). Dadurch fällt die *Marginalization* des Satzes von Bayes weg und wir können die Inferenz-Gleichung im HMM wie folgt umschreiben:

$$P(X_t = x|Y_t = y) = \frac{\phi(x) \cdot \psi(x)}{P(y)} = \phi(x) \cdot \psi(x) \tag{11}$$

Dies zeigt, dass es möglich ist, mithilfe der beiden Funktionen ϕ und ω Inferenz in der Markov-Kette zu betreiben.

Ein Partikelfilter modelliert das Ziehen der Stichproben durch einen Schwarm von n Partikeln (oder Samples)

$$P = \{p^{(i)}\}_i^n, \tag{12}$$

die jeweils mit einem Gewicht ω versehen werden. Ein Partikel lässt sich also definieren als Tupel $(\vec{s}, \omega)$ mit $\vec{s} \in S$ und $\omega \in \mathbb{R}$. Dabei ist S der Ereignisraum der Zufallsvariable X_t (siehe Definition eines HMM in 2.1.2).
Für jedes Partikel wird nun das Gewicht berechnet durch:

$$\omega = \sum_{y_i \in Y_t} \phi(p, y_i) \tag{13}$$

Für jede zum Zeitpunkt t gemachte Beobachtung wird überprüft, wie wahrscheinlich diese im Zustand s ist. Das heißt, dass für jedes Partikel sein Gewicht allein durch die Wahrscheinlichkeit für eine bestimmte Beobachtung unter Annahme des Zustands s bestimmt wird.

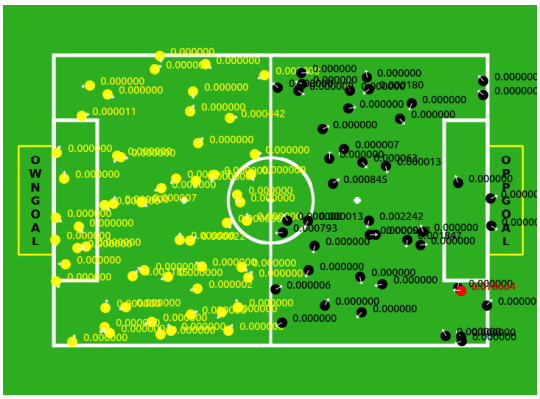


Abbildung 8: Der Filter im initialen Zustand

Der Filter wendet nacheinander die Funktionen ϕ und ψ an. Anschließend durchlaufen die Partikel einen Resampling-Schritt, der dazu dient, dass sie sich um das Maximum der Wahrscheinlichkeitsverteilung über X_t gruppieren. Dieser Vorgang wird wiederholt, bis

die Partikel an einem Ort konvergieren. In einem letzten Schritt wird nun die tatsächliche Ausprägung von X_t geschätzt.

Algorithm 1 Partikelfilter

```
1: while true do
2:    t = t + 1
3:    for all {s⃗, ω} ∈ P do
4:       Motion Update für s⃗_i
5:       ω_i = φ(X_t, Y_t) (Update Schritt)
6:    end for
7:    Resampling der Partikel (Resampling Schritt)
8:    Schätzung der latenten Variable X_t
9: end while
```

Die folgenden Kapitel werden auf die einzelnen Schritte des Partikelfilters genauer eingehen.

2.2.2 Start des Filters

Anfangs werden die Partikel zufällig im Zustandsraum verteilt, da die latente Variable X_1 schon geschätzt werden muss, gleichzeitig aber noch keinerlei Informationen zu $Q(x)$ aus vorhergegangenen Iterationen vorliegen. Hier ist eine Stelle, um Expertenwissen in das System einfließen zu lassen, denn oft ist bei der Lokalisierungsproblematik die Startposition des Roboters teilweise bekannt. Wenn die Partikel an Stellen initialisiert werden, an denen die Wahrscheinlichkeitsdichte von Pr_{X_t} anfangs schon hoch ist, so konvergiert der Filter schneller. In Kapitel 4.6 wird auf die Nutzung von Expertenwissen für eine geschickte Positionierung der Partikel noch eingegangen.
Liegen jedoch keinerlei Informationen zu den Wahrscheinlichkeiten für bestimmte Zustände vor, so sind die Partikel auf dem Spielfeld normalverteilt, d.h. ungewichtet.
Nach einer ersten Iteration steigt die Verteilung an den Punkten der Samples an, jedoch unterschiedlich stark und anschließend konvergieren die Partikel durch das in Kapitel 2.2.4 beschriebene Resampling an dem Punkt der höchsten Wahrscheinlichkeit.
Für die kommenden Iterationen kann dieser Schwarm der Partikel als Ausgangspunkt für die nächste Schätzung genommen werden. Das heißt, wir gehen davon aus, dass die Wahrscheinlichkeitsverteilung von X_{t+1} nicht viel anders als die von X_t sein wird. Jedoch ist sie mit einer großen Wahrscheinlichkeit um einen bestimmten Vektor verschoben. Diese Verschiebung soll durch das Motion Update respektiert werden.

2.2.3 Motion Update

Der erste Schritt des Algorithmus' nach Initialisierung der Partikel ist das Update der Positionen der Hypothesen auf Basis des Motion Models. Wie bereits erwähnt, gibt dieses Modell *psi* die Wahrscheinlichkeit an, von einem Zustand x_{t-1} in einen neuen Zustand x_t überzugehen.

Also wird im Motion Update die Position jedes Partikels entsprechend dem Motion Model verschoben. Da dieses Update meist fehlerbehaftet ist, wird anschließend noch ein zufälliger *Noise* mit hinzuaddiert, um diesen Fehler zu simulieren.
Angenommen, alle Partikel wären vorher auf der exakt richtigen Position gewesen, ist die Wahrscheinlichkeit, dass sich wiederum eine der Hypothesen auf einer guten Position befindet, sehr groß, da der Noise für einen Partikel wahrscheinlich genau mit dem realen Fehler korrespondiert.
Es ergibt sich also für jedes Partikel die folgende Gleichung: [20]

$$\vec{s_{new}} = \vec{s_{old}} + \Delta_{motion} + \Delta_{error}, \tag{14}$$

wobei der Fehler zufällig und auf Basis des Gewichts des Partikels addiert wird:

$$\Delta_{error} = \begin{bmatrix} \alpha_{trans} \cdot (1-q) \cdot rand \\ \alpha_{trans} \cdot (1-q) \cdot rand \\ \alpha_{rot} \cdot (1-q) \cdot rand \end{bmatrix}$$

Dabei ist *rand* eine Zufallszahl im Intervall [-1, ..., 1]. Die Werte für α_{trans} und α_{rot} können auch dynamisch angepasst werden und sind in der Regel abhängig von der Weite der Bewegung seit dem letzten Resampling.

2.2.4 Resampling des Schwarms

Dieses findet nach dem Motion Update und der Bewertung aller Partikel (worauf wir in Kapitel 4.2 noch ausführlich eingehen werden) statt. Durch die vorangegangene Bewertung lässt sich die Wahrscheinlichkeitsdichte von der gesuchten Verteilung Pr_{X_t} approximieren. Wir kennen also $Q(x)$ zumindest an jenen Punkten, wo sich Partikel befinden. Je besser die Samples schon vorher positioniert waren, umso mehr Informationen haben wir schon über Pr_{X_t}.

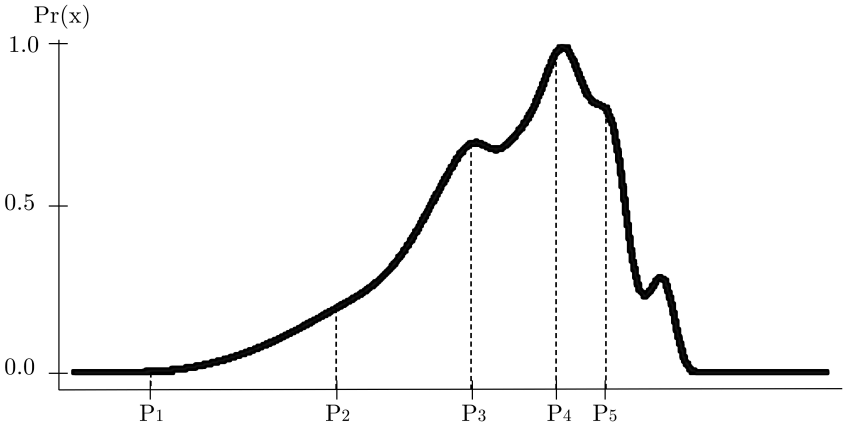


Abbildung 9: Gezogene Samples schätzen die Wahrscheinlichkeitsverteilung von X_t

Das Resampling versucht nun, die Kenntnis, die wir von $Q(x)$ erlangt haben, zu nutzen, indem solche Partikel mit hohen ω in die nächste Iteration des Filters übernommen werden, während solche mit kleinen ω nach Möglichkeit aussortiert werden. Es wird damit versucht, die Hypothesen um das Maximum von Pr_{X_t} zu gruppieren.

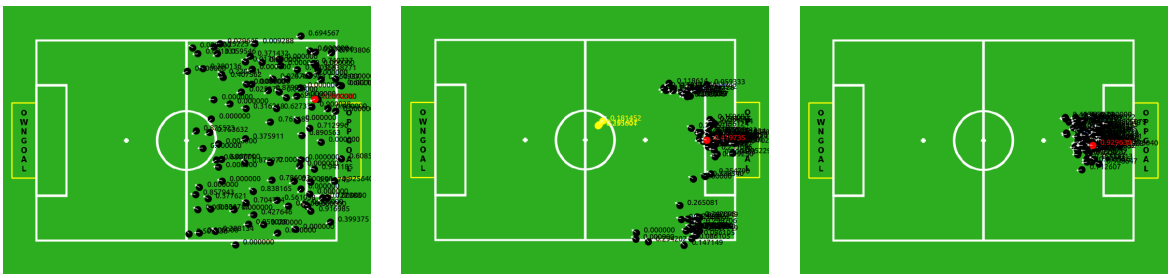


Abbildung 10: Das Resampling nach 0, 3 und 7 Iterationen, anfangs war die Feldhälfte schon bekannt

Dabei sollte das Resampling mehrere Iterationen durchlaufen und kann als eigenständige Monte-Carlo Simulation innerhalb eines Zeitschritts t betrachtet werden. Sobald die Konvergenz erreicht ist, wird die versteckte Variable X_t errechnet. Das Resampling verfährt ebenfalls probabilistisch, d.h. Hypothesen mit größerem Gewicht haben auch eine größere Chance in der nächsten Iteration des Resamplings aufzutreten, während solche mit geringem Gewicht „aussterben“ werden.

Als Analogie legen wir die Gewichte aller Partikel unsortiert aneinander auf einen Zahlenstrahl (siehe Abb.11) und wählen nun zufällig eine Zahl, die zwischen 0 und $\sum_{i=1}^{N} \omega_i$, also der Gesamtsumme aller Gewichte liegt.

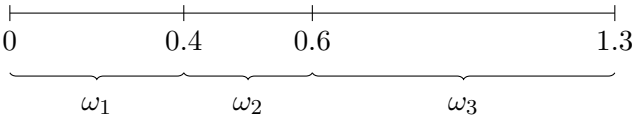


Abbildung 11: Anordnung der Partikelgewichte auf einem Zahlenstrahl

Das durch die Zufallszahl „getroffene“ Partikel wird dabei in die nächste Generation übernommen. Nach dem Gesetz der großen Zahlen wird die nächste Generation insgesamt deutlich höhere Gewichte haben als die Elterngeneration und so lässt sich das Maximum von Pr_{X_t} schnell finden.

Allerdings ist es möglich, dass beim Resampling gegen ein lokales Minimum konvergiert wird, da wir die gesamte Funktion nicht kennen und nicht den gesamten Zustandsraum absuchen wollen (siehe Kapitel 2.1.4). Um dem entgegenzuwirken, wird ein *Explorationsfaktor* eingeführt, also eine Anzahl von Hypothesen, die zufällig auf dem Feld verteilt werden und nicht Teil des Resampling sind. Diese sollen, wenn sie eine hohe Dichte von Pr_{X_t} „treffen“, und dementsprechend ein hohes Gewicht ω erhalten, in der nächsten Generation einflussreicher sein. Damit wird der Schwarm an diesem neu entdeckten Punkt konvergieren, sollte dieser wirklich das Maximum der Wahrscheinlichkeitsverteilung über X_t repräsentieren.

Die Anzahl der in jeder Iteration für die Exploration verwendeten Partikel lässt sich je nach „Sicherheit“ des Systems dynamisch bestimmen. Diese nimmt beispielsweise in einem Kidnapped Robot-Szenario rapide ab (siehe [28]), sodass mehr zufällige Hypothesen gestreut werden müssen. Der Algorithmus zur Lösung dieses Problems zur Erkennung der Sicherheit im System und der entsprechenden Reaktion nennt sich *Augmented MCL* und wurde Thrun und Burgard in [27] vorgeschlagen.

2.2.5 Augmented MCL

Rufen wir uns noch einmal in Erinnerung, dass der Augmented MCL-Algorithmus versucht, die Sicherheit zu errechnen, mit der die Partikel tatsächlich das Maximum der Wahrscheinlichkeitsverteilung gefunden haben. Dabei wird angenommen, dass ein steigender Durchschnitt aller Partikel-Gewichte eine große Sicherheit bedeutet, während ein sinkender Durchschnitt für mehr Unsicherheit spricht.
Diese Annahme ist leicht zu rechtfertigen, wenn wir uns in Erinnerung rufen, dass die Hypothesen sich ja an den Punkten sammeln, an denen die Wahrscheinlichkeit von X_t hoch ist. Wenn sich nun alle Partikel an solchen Punkten befinden, dann muss ihr durchschnittliches Gewicht:

$$\frac{1}{N} \sum_{\{\vec{s},\omega\} \in P} \omega_i$$

höher sein, als wenn sich die Partikel normalverteilt im Raum befinden.

Der Augmented MCL-Algorithmus beobachtet nun die Veränderungen des Durchschnitts-Gewichts über einen kurzen und einen längeren Zeitraum hinweg und streut die zufälligen Partikel entsprechend dem Verhältnis der langen Beobachtung zur kurzen. Der erste Teil

Algorithm 2 Augmented MCL Algorithmus nach [27]

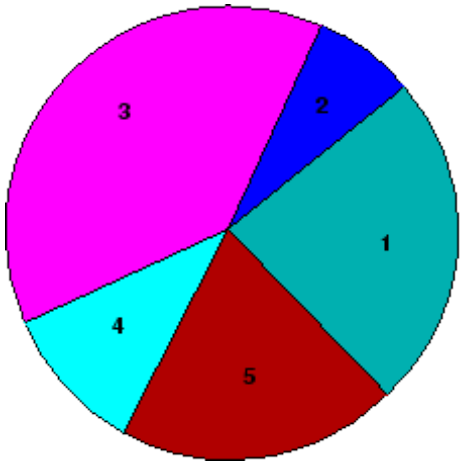
1: w_{slow}, w_{fast}
2: **for** $t = 0 \rightarrow \infty$ **do**
3: $\bar{\chi}_t = \chi_t = \emptyset$
4: $w_{avg} \leftarrow 0$
5: **for** $m = 1 \rightarrow M$ **do**
6: $x_t^{[m]} \leftarrow$ Update durch Motion-Model$(u_t, x_{t-1}^{[m]})$
7: $w_t^{[m]} \leftarrow$ Gewichtung durch Sensor-Model$(z_t,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

des Algorithmus' ist äquivalent zu dem klassischen Partikelfilter. Lediglich die Berechnung des durchschnittlichen Gewichts ω_{avg} und ihrer Veränderung ω_{fast} und ω_{slow} ist neu. Im zweiten Teil wird der Prozentsatz der Exploration dynamisch durch das Verhältnis von $\frac{\omega_{fast}}{\omega_{slow}}$ ausgedrückt.
Die beiden Faktoren $\alpha_{slow}, \alpha_{fast}$ bestimmen, wie schnell das System auf Änderungen reagiert[3]. α_{fast} ist dabei um ein Vielfaches größer als α_{slow}, wodurch sich ω_{fast} deutlich schneller an Veränderungen des durchschnittlichen Gewichts anpasst, während ω_{slow} dies nur sehr viel langsamer tut.

2.2.6 Roulette Selection

Zeile 17 des Augmented MCL-Algorithmus sieht vor, dass ein Partikel ausgewählt wird und in dessen Nähe ein neues erzeugt wird. Dieser Schritt ist nicht trivial, denn es muss zu einer zufälligen Zahl noch herausgefunden werden, zu welchem Partikel diese gehört.

Roulette Selection ist ein Algorithmus aus dem Bereich der *genetischen Algorithmen*, der die Nachkommen einer Generation gemäß der Fitness der einzelnen Individuen generiert.[1] Dabei haben solche Individuen mit einer hohen Fitness eine höhere Wahrscheinlichkeit, Nachkommen zu zeugen, als solche mit einer niedrigen Fitness. Der Name stammt von der Analogie eines Roulette-Rades, bei dem zwar alle 37 Zustände gleich wahrscheinlich sind, es jedoch möglich ist, auf mehrere Zustände gleichzeitig Wetten abzuschließen. Damit ist es möglich, größere Stücke des Rades abzudecken, wodurch die Gewinn-Wahrscheinlichkeit für den Spieler steigt. Als erstes wird hierbei die Gesamtfitness der



Population	Fitness
1	25.0
2	5.0
3	40.0
4	10.0
5	20.0

Abbildung 12: Der roulette selection Algorithmus mit einer fiktiven Generation aus fünf Individuen

Generation errechnet, also die Summe aller einzelnen Gewichte der Individuen.

$$f_{total} = \sum_{i=1}^{N} \omega_i \tag{15}$$

Im Anschluss wird eine zufällige Zahl zwischen 0 und der Gesamtfitness f_{total} gewählt (also quasi das Rad gedreht) und geschaut, welches Individuum getroffen wurde.
Dieses wird dann in die neue Generation übernommen, bzw. wenn es dort schon vorhanden ist, wird ein neues, ihm ähnliches, Individuum erzeugt.

Algorithm 3 Roulette Selection nach [1]

1: $f_{total} = \sum_{i=1}^{N} \omega_i$
2: $sumEach = 0$
3: **for all** $\{\vec{s}, \omega\} \in P$ **do**
4: $\quad probability_p = sumEach + \frac{\omega_p}{f_{total}}$
5: $\quad sumEach += probability_p$
6: **end for**
7: **for** $i \leftarrow 0$ to N **do**
8: $\quad rnd =$ Zufällige Zahl zwischen 0 und f_{total}
9: $\quad$ binäre Suche von rnd in allen Partikeln, bis p mit $(probability_p - \omega_p) \leq rnd \leq probability_p$ gefunden wurde
10: **end for**

Durch die unterschiedlich großen Gewichte der einzelnen Partikel (oder Individuen) ist sichergestellt, dass solche mit besserer Bewertung auch öfter in der Nachkommen-Generation vertreten sind.

3 Stand der Technik

Durch die Beschränkungen des RoboCups in Hinblick auf die erlaubten Sensoren der Roboter haben die Teams in fast allen Ligen für die Selbstlokalisierung das gleiche Problem zu lösen. Da die Spieler auf zwei Beinen laufen müssen, ist das Motion Model recht ungenau, was bei fahrenden Robotern beispielsweise ein weniger großes Problem darstellt. Weiterhin ist es schwierig, die Position einfach zu triangulieren, weil Abstandsmessungen aus der Perspektive der Kamera auch nur ungenau möglich sind. Dadurch haben sich Modelle als effektiv erwiesen, die fehlerhafte Messungen annehmen und auf deren Basis sowie anhand eines Modells die wahrscheinlichste Position des Roboters schätzen.

3.1 MCMC-Lokalisierung im RoboCup

Partikelfilter sind dabei inzwischen der Standard in fast allen Ligen des Robocups zur Selbstlokalisierung.[3] So nutzt beispielsweise eines der Top-Teams der *Standard Platform League* B-Human, einen Partikelfilter um die Position der Roboter zu approximieren[21], genauso wie die Darmstadt Dribblers [13] und das Team Nimbro.[2] Auch die Lokalisierung der FUmanoids basierte in den Jahren 2009 und 2010 auf einem Partikelfilter.[14, 7] Die Beliebtheit des Ansatzes ist vor allem der Einfachheit der Implementierung und der großen Robustheit geschuldet. Andere Ansätze lösen das Tracking-Problem teilweise etwas besser, können dann aber auf das *Kidnapped Robot Problem* nicht immer adäquat reagieren. Fox und Gutmann [10] formulieren diese Eigenschaft im Vergleich mit einem Kalman-Filter basierten Ansatz wie folgt:

> „Markov localization is more robust than Kalman filtering while the latter can be more accurate than the former."

Inzwischen werden allerdings fast immer erweiterte Ansätze des klassischen Partikelfilters verwendet, wie beispielsweise der auch in dieser Arbeit zum Einsatz gekommene *Augmented MCL Algorithmus*. Andere Teams wie beispielsweise B-Human nutzen einen nachgestellten Kalman-Filter, der die Ergebnisse des Partikelfilters noch zusätzlich glättet. Auch das Symmetrieproblem, also die Frage, auf welcher Feldhälfte sich der Roboter befindet, wird unterschiedlich angegangen und seit der Regelumstellung 2013 [19], die besagt, dass beide Tore dieselbe Farbe tragen müssen, intensiv bearbeitet.

Weiterhin ist die Einstellung aller Parameter eines Partikelfilter-basierten Ansatzes ein großes Problem (siehe dazu auch Kapitel 4.8.1).
So können von der Gewichtung einzelner Sensordaten über die Anzahl der Partikel auf dem Feld bis hin zu verschiedenen Methoden zur Bewertung einzelner Partikel sehr viele Einstellungen vorgenommen werden. Zur Lösung wurde von Burchardt et al. der *Particle Swarm Optimization Algorithm* vorgestellt, ein Blackbox-basierter Optimierer, der nach den optimalen Einstellungen des Filters suchen kann, sobald eine Bewertung für ein Set

an Parametern möglich ist.[3]

3.2 Feldlinien oder Landmarks

Ein *Perzept* ist ein Wahrnehmungserlebnis des Roboters. In unserem Fall beschränkt sich die Wahrnehmung auf die Kamera; hierdurch ist ein Perzept ein erkannter Gegenstand auf dem Spielfeld, wie beispielsweise ein gesehener Torpfosten. Diese Perzepte sind in ihrer Natur vielfältig und auch nicht zwangsläufig statisch (wie z.B. ein rollender Ball). Während einige von ihnen die Basis für das Observation Model darstellen, sind andere von ihnen für die Selbstlokalisierung irrelevant.
In der Literatur werden meist mehrere verschiedene *Landmarks* - also statische, an bekannten Stellen des Feldes installierte Gegenstände wie Pfosten oder Tore - für die Bewertung der Partikel im Observation Model herangezogen, vgl. [21].
Mit den Regeländerungen zum RoboCup 2013 sind allerdings viele der vorher verhältnismäßig leicht erkennbaren Landmarks weggefallen. Damit wurde versucht, das Spielfeld mehr dem menschlichen Fußballs anzupassen.[19] Dadurch ist es schwieriger geworden, dem Observation Model gute Anhaltspunkte zu geben.
Seitdem nutzen die meisten anderen Teams entweder Torpfosten oder die Punkte der Feldlinien zum Update des Observation Model, wie beispielsweise in [23] beschrieben.

3.3 Das Symmetrieproblem im RoboCup

Da im Zuge der Regeländerungen für 2013 auch die Torfarbe beider Tore auf Gelb gesetzt wurde, ist das Spielfeld nun völlig symmetrisch. Das heißt, dass das Observation Model keine Aussagen über die Spielfeldhälfte ohne zusätzliche Informationen treffen kann.
Dadurch mussten die Teams hier andere Lösungen finden, um mit dem Problem der Symmetrie umgehen zu können.
B-human hat beispielsweise ein eigenes Modul zur Schätzung der aktuellen Spielfeldhälfte entwickelt, welches die Sicherheit, sich in der eigenen Hälfte zu befinden, modelliert. Wenn dieser belief kleiner wird, versucht der Roboter sich an seinen Mitspielern zu orientieren. Diese schicken dem unsicheren Spieler zum Beispiel ihre geschätzte Ballposition, die dann helfen kann, die Symmetrie aufzulösen.[22]
Ein anderes Team der SPL, das Dutch-Nao Team, nutzt den Torwart, um einem unsicheren Roboter zu sagen, auf welcher Seite er sich befindet. Denn dieser weiß in aller Regel, dass er sich auf der eigenen Spielfeldhälfte befindet, da er sich im Spielverlauf in der Regel nicht viel bewegt und somit den hohen belief, den er am Anfang des Spiels hatte, nicht verliert.[26]

3.4 Selbstlokalisierung durch Kalman-Filter

Die populärste Alternative zur MC-Lokalisierung sind Ansätze, die auf Kalman-Filtern basieren. Bei diesem Verfahren wird die Zufallsvariable X_t nicht durch ein Monte-Carlo-Verfahren approximiert, sondern sie wird durch einen Kalman-Filter direkt geschätzt. Ein klassischer Kalman-Filter muss allerdings starke Annahmen über die Natur der Wahrscheinlichkeitsverteilung treffen, da er immer eine Gauß-Verteilung für die Verteilung annimmt.
Aufgrund dieser Tatsache werden diese im RoboCup selten verwendet, sondern durch *Extended Kalman Filter* oder *Unscented Kalman Filter* ersetzt. Beide Varianten umgehen die Limitationen des klassischen Ansatzes und können Aussagen über nicht-lineare Systeme treffen.

Kalman-Filter, Extended Kalman Filter(EKF) und Unscented Kalman Filter (UKF)
Ähnlich wie beim HMM benötigen Kalman-Filter eine Übergangswahrscheinlichkeit von einem Zustand in einen anderen (*process model*) und Informationen über die Messfehler (*measurement model*). Und auch wie beim HMM wird eine latente Variable in einer Markov-Kette errechnet.
Allerdings können im klassischen Verfahren von Rudolf E. Kálmán nur lineare Prozesse modelliert werden, was bedeutet, dass sowohl das process model als auch das measurement model lineare Funktionen sein müssen.[12]
Es werden nun keine Samples aus der Wahrscheinlichkeitsverteilung von X_t gezogen, sondern diese wird auf Basis der Messungen und mithilfe der Annahme einer Gauß-Verteilung direkt errechnet.

Extended Kalman Filter haben den Vorteil, die Limitation eines linearen Prozesses umgehen zu können, indem nicht-lineare Funktionen für das Modell genutzt werden, die dann linearisiert werden. Der Extended Kalman Filter nutzt dabei eine Taylor-Linearisierung, während im Unscented Kalman Filter eine Sampling-Strategie über sogenannte *sigma points* zum Einsatz kommt.[3]

Während mit diesen Ansätzen das Tracking-Problem sehr gut gelöst werden kann, ist es schwierig, das Kidnapped Robot Problem zu lösen. Nach einer Umsetzung des Roboters durch einen Schiedsrichter können sich Kalman-Filter basierte Ansätze nur sehr schwer auf eine neue Position hin bewegen.[10]
Da dieses Problem im RoboCup aber häufig vorkommt, eignen sich EKFs und UKFs dort nur bedingt zur Selbstlokalisierung.

[3] Für mehr Informationen über Extended- und Unscented Kalman Filter sei hier der Artikel von Uhlmann und Julier in [11] erwähnt.

Multi-Hypothesen Kalman-Filter (MHKF) Um auch das Kidnapped Robot Problem zuverlässig lösen zu können und gleichzeitig die Vorteile der geringen Laufzeit des Kalman-Filters zu behalten, wurde von Quinlan und Middleton der Multihypothesen-Kalman-Filter vorgeschlagen, der die Ansätze des Partikelfilters und der Kalmanfilter verbindet.[18] Dabei werden wie bei einem Partikelfilter mehrere Hypothesen gleichzeitig verwaltet. Jedoch ist jede von ihnen eine eigene, parametrisierte Gauß-Verteilung mit einem Gewicht ω.[17] Das Gewicht entscheidet über die Wichtigkeit einer Gauß-Verteilung für das System, ist also eine Wahrscheinlichkeit für diese Verteilung. Weiter können einzelne Verteilungen bei ausreichend großer Übereinstimmung zusammengefügt werden, was die Komplexität der Gesamtverteilung verringert ohne zu größerem Informationsverlust zu führen.[17]
Durch MHKFs lassen sich auch multi-modale Verteilungen modellieren, was mit den konventionellen Ansätzen von KF, EKF und UKF nicht möglich ist. Diese Eigenschaft macht auch eine Lösung des Kidnapped Robot Problems einfacher, denn es werden bereits mehrere „Hypothesen“ verwaltet. Auch ein Ansatz wie der Augmented MCL-Algorithmus sind hier denkbar. Weitere Informationen zu dem Thema sind in [17] zu finden.

4 Umsetzung

Die in dieser Arbeit vorgestellte Lokalisierung basiert auf einem Partikelfilter, da sie deutlich einfacher wartbar und implementierbar ist als etwa ein MHKF. Die grundsätzlich höheren Laufzeiten des Ansatzes werden in Kauf genommen, da die Roboter der FUmanoids mit einer Umstellung der Hardware inzwischen in der Lage sind, diese Daten zu verarbeiten. Der grundlegende Algorithmus der Lokalisierung wurde bereits in Algorithmus 1 beschrieben, und die Implementierung folgt diesem zum größten Teil. Dennoch sind einige Änderungen aus der Literatur in die Lösung eingeflossen.

Insbesondere war es schwierig, eine akkurate und gleichzeitig kostengünstige Funktion für das Observation Model ϕ zu finden. So wird pro Frame und für jedes Partikel immer eine *hypothetische Wahrnehmung* generiert, also die erwarteten Perzepte, die der Roboter sehen würde, angenommen er befände sich auf der Position $\vec{s}$. Diese erwartete Wahrnehmung wird anschließend mit der tatsächlich gesehenen verglichen und führt schließlich zur Bewertung des Partikels.
Im Rahmen einer Lokalisierung auf Basis von Feldlinien bedeutet dies, dass wir zunächst also zwei verschiedene Mengen an Kanten E, H erwarten. Einerseits die gesehenen, „wahren“ Kanten E und die hypothetische Wahrnehmung H_p, wobei diese für jedes Partikel existiert. Abbildung 13 zeigt die Hypothese und die dazugehörige Position des Partikels

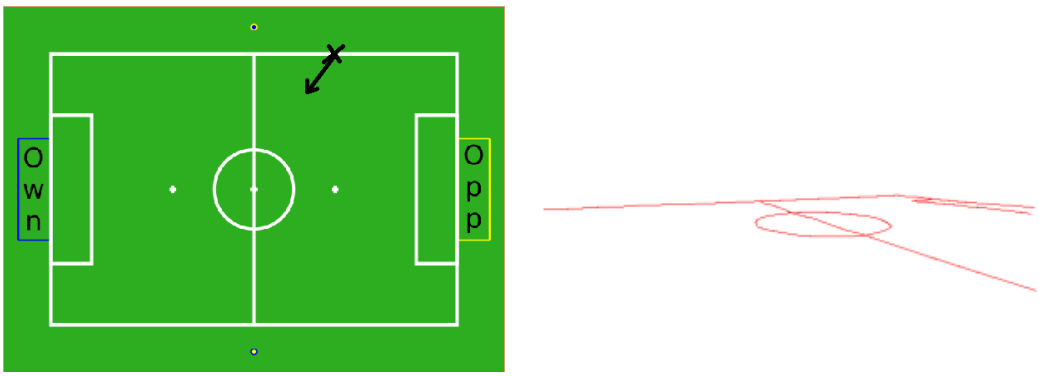


Abbildung 13: Die Position eines Partikels auf dem Spielfeld und die dazugehörige hypothetische Sicht als 2D-Bild

auf dem Spielfeld. Der erste Schritt für die Aufstellung einer geeigneten Fehlerfunktion für ein Partikel muss also die Generierung der hypothetischen Kanten für dieses sein.

4.1 Generierung der Hypothesen

Gesucht ist demnach eine Funktion $f : \vec{s} \rightarrow H$, die uns die hypothetische Wahrnehmung eines Partikels $\{\vec{s}, \omega\}$ errechnet. Dies ist allerdings nur möglich, wenn gewisse Informatio-

nen über die Umgebung schon vorliegen. Es muss zum Beispiel in absoluten Koordinaten bekannt sein, wo welche Linie des Feldes beginnt oder an welcher Stelle genau der Mittelkreis liegt etc.[24] Diese zusätzliche Menge an bekannten Linien bezeichnen wir im Folgenden als E_{field}. Eigentlich suchen wir also

$$f : (\vec{s}, E_{field}) \rightarrow H.$$

Um die erwartete Wahrnehmung, oder *expected view*, aus den beiden Parametern zu errechnen, bedarf es aber noch weiterer Informationen. Denn auch wenn das Spielfeld nur zweidimensional ist, hat der Roboter eine gewisse Höhe, die in die Berechnungen einfließen muss. Gleiches gilt für den Neigungswinkel der Kamera α, den *Pitch* (siehe Abbildung 14).

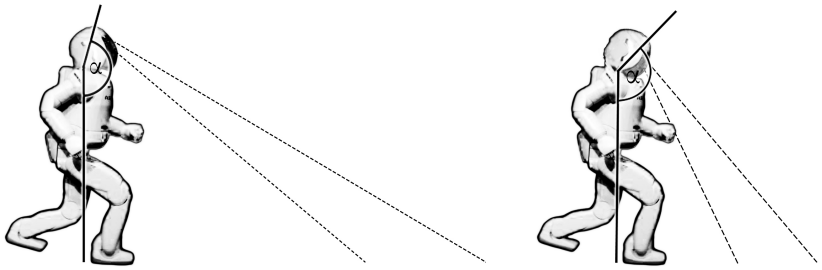


Abbildung 14: Der Neigungswinkel der Kamera bestimmt in großen Teilen die Perspektive der Kamera und damit die hypothetische Sicht eines Partikels

Da diese Daten nicht auch noch vom Partikelfilter simuliert werden sollen, müssen sie aus der Stellung der Motoren des Roboters gewonnen werden. Die Länge sämtlicher Bauteile ist bekannt und auch die Stellungen der Motoren können in jedem Frame ausgelesen werden. Aus diesen Parametern kann dann die aktuelle Höhe der Kamera errechnet werden.[21]
Der Pitch -und Yaw-Winkel des Kopfes lässt sich ebenso aus den Kopfmotoren berechnen. Hierbei wird der Yaw-Winkel, also die Drehung des Kopfes nach links oder rechts, auf die Rotation θ der Position des Partikels einfach hinzuaddiert. Im Weiteren gehen wir davon aus, dass diese Daten korrekt sind, auch wenn es durchaus einige Messfehler bei der Höhe geben kann. Alternativ könnten diese Werte auch noch von der Selbstlokalisierung errechnet werden.

Zur Berechnung wird das Feld in absoluten Koordinaten einfach so verschoben, dass sich die Position des Partikels am Ursprung des Koordinatensystems befindet. Die daraus resultierenden, zu dieser Position relativen Linien werden dann auf das Bild projiziert, also in Bildkoordinaten umgerechnet. Damit wird im restlichen Kapitel im Wesentlichen dem in [24] vorgestellten Ansatz zur Erstellung der hypothetischen Sicht gefolgt.

4.1.1 Translation und Rotation

Zunächst werden die Feldlinien translatiert, also ihr Koordinatensystem additiv verändert. Anschließend werden noch Rotation, Höhe des Roboters und der Pitch-Winkel der Kamera berücksichtigt, was dann zu den erwarteten Linien in relativen Koordinaten führt.

Translation Damit die Feldlinien zunächst relativ zur Position des Partikels erscheinen, müssen die einzelnen Punkte jeder Linie in relative Koordinaten überführt werden. Dadurch wird der Ursprung des Koordinatensystems genau so verschoben, dass der Ursprung anschließend auf der Hypothese liegt. Also lässt sich die Translation als eine einfache Addition auf die absoluten Punkte der Feldlinien formulieren:

$$\vec{p_{relativ}} = \vec{p_{absolut}} + \vec{p_{robot}}$$

Das Resultat sind die Punkte, relativ zur Position des Partikels; dessen eigene Pose wird demnach immer $(0, 0)$ sein.

Rotation Der Winkel θ, der die Rotation des Partikels in Grad beschreibt, sowie der Neigungswinkel (Pitch) der Kamera φ sind hier allerdings noch nicht eingeflossen.
Die Rotation um θ entspricht einer Drehung um die z-Achse, und entsprechend kann φ, also der Pitch der Kamera als Rotation um die x-Achse aufgefasst werden. Um das mathematisch auf einen Punkt (Vektor) anwenden zu können, existieren Drehmatrizen, die, aneinander multipliziert, zur Rotationsmatrix um beide Achsen zusammen führen. Diese muss dann mit jedem Punkt der relativen Feldlinien multipliziert werden, um die perspektivisch gedrehten Linien für die Projektion zu erhalten. Jede Hypothese hat dabei ihre eigene Matrix, da diese von θ und φ des Partikels abhängig ist.

$$M_{rot} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \cdot \cos(\varphi) & -\sin(\theta) \cdot -\sin(\varphi) \\ \sin(\theta) & \cos(\theta) \cdot \cos(\varphi) & \cos(\theta) \cdot -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{pmatrix}$$

Jede Koordinate muss anschließend mit der Rotationsmatrix multipliziert werden:

$$P_{relAndRot} = M_{rot} \cdot P_{relativ}$$

4.1.2 Projektion auf das Bild

Das bisherige Ergebnis zeigt zwar schon die Linien in korrekten relativen Koordinaten zur Position der Hypothese. Jedoch wird die Extraktion der Kanten stets Bildkoordinaten

ergeben, da die Kantenextraktion ja auf einem, von der Kamera aufgenommenen, Bild geschieht. Es stellt sich nun das Problem, diese beiden unterschiedlichen Koordinatensysteme, die dennoch miteinander in Relation stehen, in einander umzurechnen.

Zur Lösung bietet sich das *Pinhole Camera Model* an, welches auf dem Prinzip alter Lochkameras beruht[8] und eine Projektion eines Punktes im Raum durch eine imaginäre Blende auf eine ebenso imaginäre Leinwand ermöglicht (vgl. Abbildung 15).
Bei einer Lochkamera werden Lichtstrahlen auf einer Leinwand (hier: *imaginary plane*) eingefangen. Jedoch werden alle Strahlen vorher an der *Pinhole*, also einem sehr kleinen Loch, gebündelt. Dadurch ist die Richtung der Lichtstrahlen bis auf eine kleine Streuung bekannt und das Bild der dreidimensionalen Umgebung lässt sich zweidimensional abbilden.
Das projizierte Bild ist hier genau verkehrt herum (siehe Abb. 15). Das Pinhole Camera Model geht weiterhin von einer unendlich großen Leinwand aus, was für diesen Fall nicht zutrifft, da die gewünschte Größe des Bildes gleich der Größe des Kamera-Sensors ist.

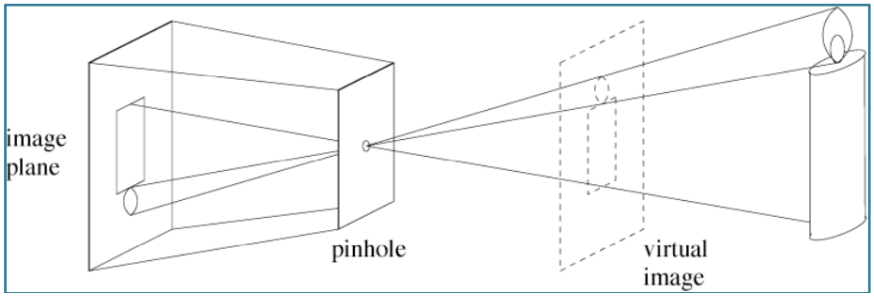


Abbildung 15: Verfahren der Pinhole Projektion[8]

Dabei ist das Modell stark vereinfacht und bestimmte Faktoren, wie zum Beispiel die Krümmung der Linse, werden nicht respektiert. Trotzdem eignet sich das Pinhole-Camera-Model für die Projektion der Hypothesen auf ein Bild durchaus gut, denn die minimalen Fehler durch Verzerrungen in der Linse fallen nicht stark ins Gewicht. Durch die ständige Bewegung der Roboter und die Kantenextraktion sind die Daten sowieso fehlerbehaftet, weshalb das Modell generell mit Fehlern umgehen muss.[24]

Die mathematische Abstraktion des Prinzips der Lochkamera macht es möglich, die Projektion als Matrix-Multiplikation aufzufassen:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

wobei f hier die Brennweite, also einen fixen, bekannten Wert bezeichnet. Für die einzelnen Punkte ergibt sich:

$$P_{Hypothese} = M_{project} \cdot P_{relAndRot}$$

mit der Brennweite f und der Matrix $M_{project}$ wie folgt:

$$M_{project} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Daraus erhalten wir die Bildkoordinaten für jeden einzelnen Punkt der Kanten, die wir nun mit den tatsächlich von der Kantenextraktion gewonnenen Daten vergleichen können. Jedoch ist die Hypothese in ihrer Sichtweite unbeschränkt und sieht dementsprechend auch weit entfernte Feldlinien. Es wäre somit notwendig, diese weit entfernten Perzepte aus dem Kamerabild zu extrahieren, was sich als sehr aufwendig erweist.

4.1.3 Beschränkung der Sichtweite

Wie im vorherigen Abschnitt bereits angedeutet, ist die Sichtweite ein Problem, da die hypothetische Sicht für ein Partikel erst einmal unbeschränkt ist. Wenn nun etwa der Roboter an einer Ecke des Spielfeldes mit Blick auf das gesamte Feld steht, müsste die Kantendetektion auch den Strafraum auf der anderen Seite (also bis zu 6 Meter entfernt) sauber erkennen, damit die Kanten sinnvoll miteinander verglichen werden können.
Dies ist mit der momentan im Roboter verbauten Hardware nicht möglich, da diese entfernten Linien teilweise nur wenige Pixel breit sind und von Störungen im Bild nicht mehr zu unterscheiden sind. Deshalb wurde hier der entgegengesetzte Ansatz verwendet: Die eigentlich perfekt generierten Hypothesen werden in ihrer Sichtweite künstlich beschränkt, damit sie den gesehenen Kanten ähnlicher werden (siehe Abbildung 16).

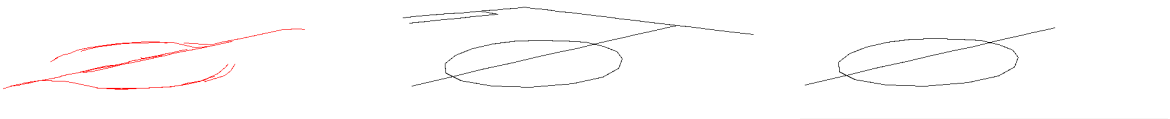


Abbildung 16: Die erkannten Feldlinien (links), eine hypothetische Sicht mit unbegrenzter Sichtweite (Mitte) und mit beschränkter Sichtweite (rechts)

Durch diesen Ansatz kann der Vergleich der Kanten deutlich verbessert werden, da hypothetische Wahrnehmung und tatsächlich gesehene Kanten in Zahl und Orten deutlich ähnlicher wurden.

4.2 Die Qualität des Observation Model

Das Observation Model ist für die Qualität der Selbstlokalisierung von entscheidender Bedeutung, denn es stellt nicht nur das Korrektiv für die Odometrie dar, sondern ist auch die einzige Quelle für den Roboter, um sich am Anfang des Spiels oder nach Verstellen durch den Schiedsrichter zu lokalisieren.

4.2.1 Vergleich von Kanten

Wir haben in den vorherigen Abschnitten gesehen, wie die hypothetische Wahrnehmung für ein Partikel generiert werden kann. Das Problem liegt nun im Vergleich dieser mit den „realen“ Kanten, um daraus eine Bewertung des Partikels möglich zu machen.
Gesucht ist eine spezielle Fehlerfunktion

$$erf : E_{seen} \times E_{hypothesis} \to \mathbb{R}, \tag{16}$$

die eine Hypothese mit den gesehenen Kanten vergleicht und den Fehler als reelle Zahl ausdrückt. Dabei sollte ein größeres Resultat eine eher schlechte Hypothese beschreiben,

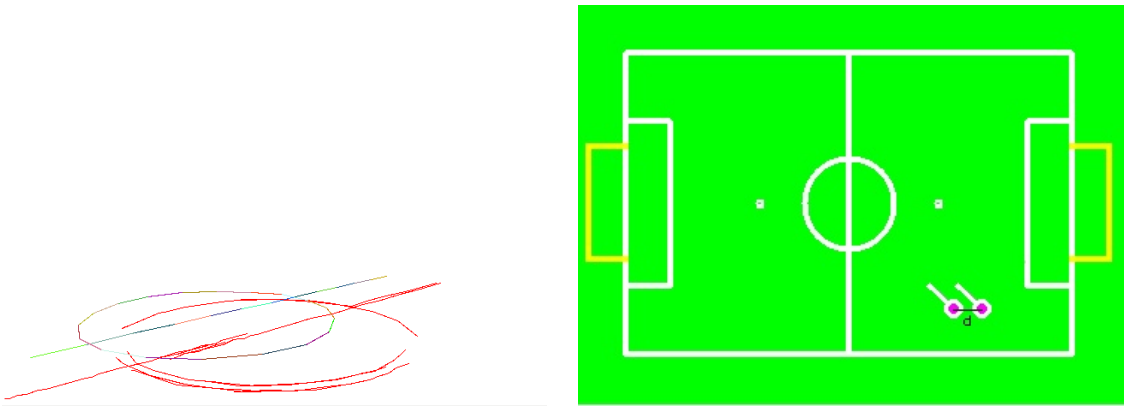


Abbildung 17: Eine perfekte Metrik könnte aus den zwei Bildern den Abstand der Kameras von einander genau berechnen.

während kleinere Werte für bessere Partikel sprechen.
Eine perfekte Metrik würde nun für eine beliebige Hypothese den euklidischen Abstand zwischen realer Position und Hypothese als Resultat haben, wie in Abbildung 17 verdeutlicht. Für eine perfekte Fehlerfunktion erf^* würde gelten: $erf^*(E_{seen}, E_{hypothesis}) = d$, wobei d den euklidischen Abstand zwischen Hypothese und realer Position des Roboters beschreibt, was auch in Abbildung 17 zu sehen ist.

Die Fehlerfunktion soll die Wahrscheinlichkeitsverteilung unserer Zufallsvariable X_t aus Kapitel 2.1.4 möglichst gut widerspiegeln. Die beiden Bilder werden nun verglichen, in-

dem jeder einzelnen Kante ein Partner aus dem „anderen Bild“ zugeordnet wird. Der Vergleich zwischen den Partnern wird dann gemittelt und bestimmt die Fehlerfunktion, also:

$$erf(E_{seen}, E_{hypothesis}) = \frac{1}{|E_{hypothesis}|} \cdot \sum_{e_h \in E_{hypothesis}} \delta(e_h, getCorrespondingPartner(e_h)) \tag{17}$$

Es ist schnell zu sehen, dass sich hier zwei Fragen ergeben:

1. Wie ist es möglich, den besten Partner einer Kante im anderen Bild zu bestimmen, ohne dass dafür alle Kanten untersucht werden müssen?

2. Wie genau soll die Abstandsfunktion für zwei Kanten aussehen?

Das erste Problem werden wir später noch genauer untersuchen. Erst einmal können wir davon ausgehen, dass die Partnersuche für eine Kante aus der erwarteten Sicht alle gesehenen Kanten miteinander vergleicht und diejenige Kante als Partner wählt, für die der Abstand minimal ist:

$$getCorrespondigPartner(e_{hypothesis}) = \min_{e_i \in E_{seen}} \delta(e_i, e_{hypothesis}) \tag{18}$$

Dieser naive Ansatz wird immer $\mathcal{O}(|E_{seen}| \cdot |P|)$ Zeit benötigen und dabei jede gesehene Kante mindestens $|P|$ mal untersuchen. Dieses Problem einer annähernd quadratischen Laufzeit (wenn $|P| = |E|$) wird in Kapitel 4.3.1 noch genauer untersucht. In Kapitel 4.3 wird eine Methode zur Verbesserung der Laufzeit vorgestellt.

4.2.2 Die Abstandsfunktion

Um das zweite Problem - die Definition einer Abstandsfunktion δ - zu lösen, ist eine Metrik vonnöten, welche aus dem Vergleich von hypothetischer Sicht und real erkannten Kanten eine Wahrscheinlichkeit für die Partikel berechnet.

Ein naiver Ansatz wäre beispielsweise der punktweise euklidische Abstand von erwarteter und gesehener Kante. Diese Lösung hätte den Nachteil, dass die Richtung der Kanten nur eine marginale Rolle spielt, allerdings auch den Vorteil der schnellen Berechenbarkeit, was gerade für den Vergleich aller gesehenen Kanten mit der untersuchten hypothetischen Kante wichtig ist. Dennoch existieren viele Spielsituationen, in denen potentiell sehr viele Kanten vorhanden sind, daher kann eine Hypothese sich recht einfach mit dem gesehenen Bild überschneiden.
Deswegen ist der naive Ansatz oftmals ungenau, wie auch Abbildung 18 zeigt. Denn hier würde die Kante e_1 der bessere Partner für e_2 sein, obwohl in einem realistischen Szenario e_3 der viel bessere Kandidat wäre, da e_2 und e_3 parallel zu einander und nicht weit voneinander entfernt sind.

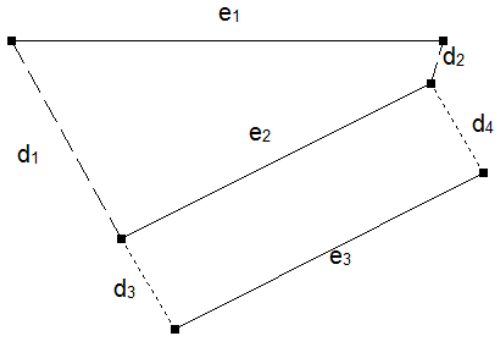


Abbildung 18: Ein naiver Kantenvergleich auf Basis der euklidischen Distanz. Die drei Kanten e_1, e_2 und e_3 werden durch die Abstände vom ersten zum letzten Punkt miteinander verglichen.

Stronger und Stone [25] stellen dagegen einen Vergleich von Kanten vor, der auf all diese Aspekte Rücksicht nimmt.
Für zwei potentielle Partner-Kanten gibt es drei verschiedene Kriterien, nach denen sich die Bewertung richtet:

- *Overlap* bezeichnet die Strecke, auf der sich beide Kanten überlappen. Aufgrund der möglichen unterschiedlichen Richtungen beider Kanten ist diese Beziehung nicht kommutativ, d.h. $overlap(e_i, e_j) \neq overlap(e_j, e_i)$.

- Der *Abstand* bedeutet hier den Mittelwert aus d_1 und d_2, also die Distanz am Anfang und am Ende der Linie.

- Aus Abbildung 19 nicht erkennbar, aber leicht vorstellbar ist auch die *Winkeldifferenz* zwischen den beiden Kanten, also $\Delta(\alpha_i, \alpha_j)$.

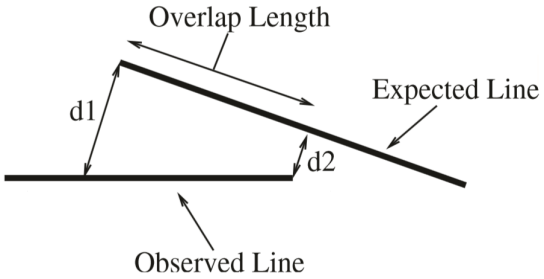


Abbildung 19: Der Vergleich zweier Kanten nach Stronger und Stone [25]

Dieser Vergleich lässt sich verhältnismäßig leicht implementieren und ist gleichzeitig angemessen schnell in der Ausführung. Er eignet sich daher gut für die Rahmenbedingungen. Dadurch ergibt sich insgesamt eine Metrik δ für die Fehlerfunktion, die sich wie folgt

definieren lässt:

$$\delta(e_i, e_j) = r_1 \cdot overlap(e_i, e_j) + r_2 \cdot \frac{d_1 + d_2}{2} + r_3 \cdot \Delta(\alpha_i, \alpha_j) \tag{19}$$

Wir haben also nun eine Metrik, um 17 und 18 zu berechnen und damit alle nötigen Instrumente für die Bewertung der Partikel.

4.2.3 Bewertung eines Partikels

Da das Observation Model eine Wahrscheinlichkeitsverteilung angeben soll, muss jedem Partikel ein Gewicht, oder *belief*, zwischen 0 und 1 zugeordnet werden. Da der errechnete Fehler die normalisierte Summe aller einzelnen Kantenvergleiche beschreibt (vgl. Gleichung 17), sollte eine Übersetzung von durchschnittlichem Fehler in Wahrscheinlichkeit stattfinden. Dies lässt sich durch eine Gauß'sche Funktion wie folgt beschreiben:

$$\omega_p = exp^{-\alpha \cdot error^2} \tag{20}$$

Da die Wahrscheinlichkeitsverteilung des Fehlers nicht bekannt ist, muss sie manuell skaliert werden, hier durch α. Mit diesem Wert wird einerseits die Schiefe der Kurve verändert, wie Abbildung 20 beschreibt. Andererseits kann damit aber auch die Toleranz des Algorithmus für gute, aber dennoch nicht optimale, Gewichte festgelegt.

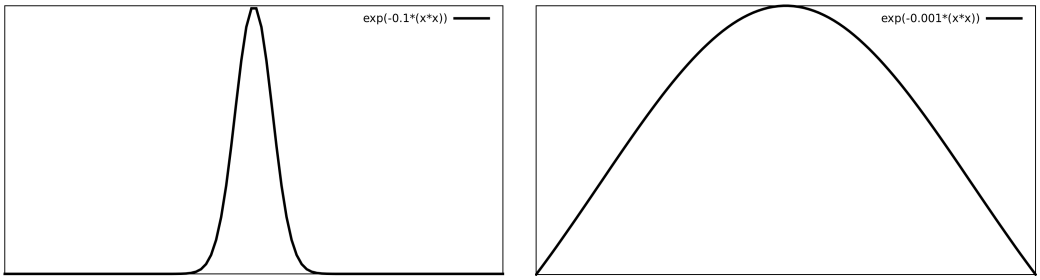


Abbildung 20: Die Wahl von α beeinflusst die Größe des Beliefs

Nun kann zusammenfassend der Algorithmus für das Observation Model festgehalten werden:

Algorithm 4 Observation Model

1: **for all** $\{\vec{s}, \omega\}_p \in P$ **do**
2: $E_{hypothesis} = generateHypothesis(\vec{s}_p)$
3: $error = \frac{1}{|E_{hypothesis}|} \cdot \sum_{e_h \in E_{hypothesis}} \delta(e_h, findPartner(e_h))$
4: $\omega_p = e^{-\alpha \cdot error^2}$
5: **end for**

4.3 Eine Methode zur Verbesserung der Laufzeit

4.3.1 Das Problem der quadratischen Laufzeit

Wie bereits in Gleichung 18 skizziert, muss für die Berechnung des besten Partners für eine Hypothesen-Kante jede einzelne gesehene Kante untersucht werden. Angenommen die beiden Mengen E_{seen} und $E_{hypothesis}$ wären ungefähr gleich groß, dann ergibt sich eine quadratische Laufzeit von $\mathcal{O}(|E_{seen}| \cdot |E_{hypothesis}|)$. Da aber in der Regel die meisten dieser Kanten wahrscheinlich keine guten Partner darstellen, bietet es sich an, hier nach Optimierungen zu suchen.
Zusätzlich dazu sind die gesehenen Kanten immer die Gleichen. Dennoch müssten sie nach Algorithmus 17 für jedes Partikel neu untersucht werden. Indem sie vor Ablauf des Partikelfilters vorbereitet werden, lässt sich das Problem der quadratischen Laufzeit verringern.
Jeder gesehenen Kante wird ein Schlüssel zugewiesen, nach welchem sie dann gespeichert wird, ähnlich wie beim Hashing. Soll nun ein Partner für eine beliebige Hypothesen-Kante gesucht werden, kann der Schlüssel für diese berechnet und anschließend mit denen der vorbereiteten Kanten verglichen werden. Der ähnlichste Schlüssel ist automatisch der beste Partner.
Auf diese Art und Weise wäre es theoretisch möglich, den Partner zu einer Kante in erwarteter Zeit $\mathcal{O}(1)$ zu finden [4], sofern einige realistische Annahmen getroffen werden.

Entscheidend für die Qualität einer solchen Funktion $h : E \to E$ ist, dass für eine beliebige Kante e_x der beste Partner extrahiert wird und sie sich nach Möglichkeit nur in der Laufzeit von einer Lösung ohne Optimierung unterscheidet. Ideal würde gelten:

$$arg \min_{e_i \in E_{seen}} \delta(e_x, e_i) = h(e_x). \tag{21}$$

Dadurch muss die Berechnung des Schlüssels von e_x in einer Relation zu der Abstandsfunktion δ stehen. Da wir aber δ in einer Form definiert haben, die von beiden Kanten abhängt, ist die Suche nach einer guten Form der Berechnung des Schlüssels problematisch.

Leider enthält das Szenario einige Bedingungen, die ein Hashing-Verfahren unmöglich machen. So werden selbst bei einem perfekten Match kaum zwei Kanten komplett gleich sein, sei es aufgrund von Fehlern in der Kantendetektion oder in der Projektion der Punkte (siehe 4.1). Da aber für die Anwendung eines Hashing-Verfahrens die Schlüssel genau gleich sein müssen, scheidet dieses als Datenstruktur aus.

Es sollte vielmehr nach ähnlichen Schlüsseln gesucht werden können, was ein sortiertes Array oder einen Binärbaum sinnvoll erscheinen lässt. Da die gesuchte Datenstruktur in einem ersten Schritt befüllt wird und erst danach nach Elementen gesucht wird und weil es uns möglich ist, diese Trennung zu garantieren, bietet sich ein Array an, welches vor dem ersten Suchvorgang sortiert wird.

Denn im Gegensatz zum Binärbaum bietet ein sortiertes Array die durchaus vorteilhafte Eigenschaft, in konstanter Zeit mehrere benachbarte Elemente extrahieren zu können.

4.3.2 Die Suche nach einem guten Schlüssel

Die Suche nach einer geeigneten Berechnung des Schlüssels stellt sich ebenfalls als schwierig heraus, da die Funktion δ ja immer genau zwei Kanten miteinander vergleicht, die Schlüsselberechnung jedoch unabhängig von potentiellen Partnern sein soll.
Ein guter Schlüssel sollte also zwei Eigenschaften haben, deren Sinn leicht ersichtlich sein sollte:

1. Er sollte für zwei ähnliche Kanten e_i und e_j ähnlich sein

2. Er sollte vermeiden, dass zwei unterschiedliche Kanten den identischen Schlüssel zugewiesen bekommen.

Beispielsweise ließe sich der euklidische Abstand vom Ursprung als Kriterium für den Schlüssel heranziehen. Wird dieser jedoch in einer reellen Zahl zusammengebracht, ergeben sich zwangsläufig Doppeldeutigkeiten. Zum Beispiel führen die beiden Koordinaten $(10, 20)$ und $(20, 10)$ für einen Punkt schon zum gleichen Schlüssel. Wird nun noch zusätzlich der Winkel der Kante in Betracht gezogen, verstärkt sich das Problem, wie Abbildung 21 zeigt.
Gleichzeitig sollte der Schlüssel möglichst viele Informationen über die Kante enthalten, denn wenn beispielsweise ihre Steigung nicht im Schlüssel kodiert wird, kann es auch so zu Problemen bei der Suche für einen Partner kommen. Eine eindeutige Lösung bei der Reduktion mehrerer Dimensionen auf eine Einzige ist dabei nicht möglich.

Nach dem Ausprobieren vieler verschiedener Kriterien für die gute Berechnung der Schlüssel, wurde für diese Arbeit eine Zusammensetzung aus drei Komponenten gewählt:

1. Die Länge des Vektors vom Ursprung zum Mittelpunkt der Kante.

2. Der Winkel dieses Vektors im Vergleich zur x-Achse.

3. Der Gradient der Kante, also ihre Richtung.

Dadurch bilden die ersten beiden Werte zusammen die Polarform des Vektors zum Mittelpunkt der Kante und als drittes Kriterium kommt die Steigung der Kante hinzu. Hierbei kann es durchaus zu unerwünschten Effekten kommen, wie zum Beispiel den oben erwähnten Doppeldeutigkeiten. Dieser Effekt lässt sich jedoch verringern, indem mehrere Kanten aus dem Array entnommen werden.

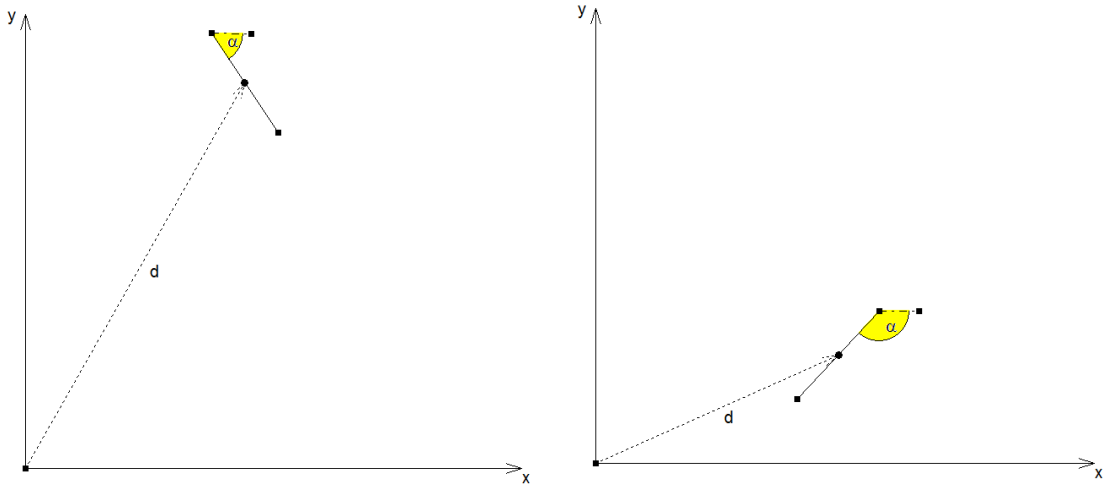


Abbildung 21: Doppeldeutigkeiten bei der Kodierung verschiedener Charakteristika in eine reelle Zahl, den Schlüssel.
Obwohl beide Kanten sehr unterschiedlich sind, könnten größerer Winkel und kleinere Distanz zum selben Schlüssel für komplett unterschiedliche Kanten führen.

Je mehr Kanten also bei der Suche entnommen werden, desto weniger können solche unerwünschten Verhaltensweisen vorkommen. Allerdings verschlechtert sich die Laufzeit entsprechend.

4.3.3 Operationen auf dem Array

Die gesuchte Datenstruktur braucht nur zwei Operationen, das Einfügen und das Suchen. Ob die Liste sortiert werden muss, lässt sich automatisch entscheiden, je nachdem ob vor dem letzten Sortieren eine Einfüge-Operation stattgefunden hat, oder nicht. Eine weitere benötigte Operation, nämlich alle Elemente der Liste zu löschen, ist in der Umsetzung trivial und wird hier nicht weiter behandelt.

Das Einfügen Die gesehenen Kanten E_{seen} werden nun durch iteriert und für jede Kante $e_i \in E_{seen}$ wird der Schlüssel k_i - wie in 4.3.2 beschrieben - berechnet. Das Paar $\{e_i, k_i\}$ wird als Element an das Array angehängt. Dieser Vorgang erzeugt einen Overhead in jedem Frame, da alle gesehenen Kanten einmal in die Liste eingefügt werden müssen, danach reduziert sich jedoch der Aufwand für ein Partikel drastisch, so dass sich das Verfahren lohnt.

Suchen In einem ersten Schritt wird der Schlüssel der Referenz-Kante e_{ref} berechnet. Um dann in der Liste zu suchen, muss festgestellt werden, ob diese schon sortiert ist. Wir machen uns die Eigenschaft zunutze, dass zuerst alle Elemente eingefügt werden und erst danach die Suchanfragen stattfinden. Es wird einfach bei der ersten Suchanfrage geschaut, ob das Array bereits sortiert ist; falls nicht, wird es vor der Suche sortiert.

Ist die Liste schon sortiert, so wird mittels einer Abart der binären Suche in logarithmischer Zeit nach dem ähnlichsten Element zum Suchschlüssel gesucht.
Da bei der normalen binären Suche eine stetige Annäherung an den gesuchten Wert stattfindet, können wir davon ausgehen, dass das ähnlichste Element in genau dem Augenblick gefunden wurde, in dem die Distanz zwischen den Schlüsseln des letzten und des aktuellen Elements wieder größer wird.
Für die Suche nach einer Kante im Array, die e_{ref} am Ähnlichsten ist, lässt sich folgender Algorithmus formulieren:

Algorithm 5 Suche nach dem ähnlichsten Element im Array

1: $key := calculateKey(e_{ref})$
2: $first := 0$
3: $last := array.size$
4: $previousDistance := 0$
5: **while** *true* **do**
6: $\quad mean := first + (\frac{last - first}{2})$
7: $\quad distance := \Delta(array[mean], key)$
8: $\quad$ **if** $distance = 0$ **then**
9: $\quad\quad$ return array[mean]
10: $\quad$ **else if** $|distance| \geq previousDistance$ **then**
11: $\quad\quad$ return array[mean]
12: $\quad$ **else if** $|distance| \geq 0$ **then**
13: $\quad\quad first := mean$
14: $\quad$ **else**
15: $\quad\quad last := mean$
16: $\quad$ **end if**
17: $\quad previousDistance := distance$
18: **end while**

Da es sich aufgrund der möglichen Doppeldeutigkeiten anbietet, mehrere Kanten aus dem Array zu extrahieren, werden nun einfach die benachbarten Elemente um die ähnlichste Kante ebenso zurückgegeben. Diese sind in jedem Fall die nächst-ähnlichen Kanten, da die Schlüssel sortiert sind. Für all diese Kanten wird nun der Partner zur Referenz-Kante e_{ref} berechnet, indem die Funktion *delta* für alle Kandidaten angewandt wird. Das Minimum ist hier der Partner von e_{ref}, also:

$$partner = arg \min_{e_i \in E_{candidates}} \delta(e_i, e_{ref}).$$

4.3.4 Effizienz der Optimierung

Während die klassische Vergleichsfunktion jede gesehene Kante mit jeder erwarteten zu vergleichen hatte und dies auch noch für jedes Partikel, ist die optimierte Version deutlich schneller. Das Einfügen der gesehenen Kanten in die Liste ist vom Zeitaufwand her linear, denn die Berechnung des Schlüssels erfolgt in konstanter Zeit. Dieser Aufwand mit oberer Grenze $\mathcal{O}(n)$ entsteht einmal pro Frame, wobei n hier die Anzahl an gesehenen Kanten beschreibt. Anschließend lassen sich alle erwarteten Kanten in asymptotischer Zeit $\mathcal{O}(\log_2 n)$ vergleichen, denn auch die Abstandsfunktion δ ist in ihrer Laufzeit konstant.

Vergleichen wir also nur die Effizienz des naiven und präziseren Algorithmus mit der hier besprochenen Optimierung, so stellen wir fest, dass der naive Ansatz $\mathcal{O}(n_{seen} \cdot n_{expected} \cdot |P|)$ Aufrufe der Abstandsfunktion δ pro Frame braucht, während es mit der Optimierung noch lediglich $\mathcal{O}(n_{seen} + n_{expected} \cdot |P| \cdot \log_2 n_{seen})$ Vergleiche sind.
Sollte die Bewertung der Partikel mit Optimierung eine ähnliche Qualität aufweisen können wie der naive Algorithmus, so wäre die Verbesserung wünschenswert.

4.4 Effizientes Resampling

Nachdem das Update der Gewichte für jedes Partikel abgeschlossen ist, dementsprechend eine Schätzung über die Wahrscheinlichkeitsdichte der gesuchten Zufallsvariable vorliegt, werden die Partikel in Richtung des Maximums, bzw. der Maxima dieser Dichtefunktion verschoben. Um diese Aufgabe schnell und genau zu erledigen, bedienen wir uns einer Lösung aus dem Bereich der genetischen Algorithmen.
Wenn die einzelnen Partikel als Individuen mit einer Fitness ω gedacht werden, lässt sich das Resampling-Problem umformulieren als Berechnung der nächsten Generation. Bedingung hierfür ist, dass solche Individuen mit großer Fitness auch wahrscheinlich mehr Nachkommen haben werden.

Der in Kapitel 2.2.6 vorgestellte Roulette-Selection-Algorithmus ist in der Lage, eine nachfolgende Generation unter Berücksichtigung der Fitness jedes Individuums zu berechnen. Dabei ist der Algorithmus vergleichsweise schnell, auch wenn zwei Iterationen durch die Menge P nötig sind. Denn wenn eine Zufallszahl gezogen wurde, (in der Analogie des Roulette-Rades: Die Kugel zum Stillstand gekommen ist) muss für diese Zahl das zugehörige Individuum oder Partikel ermittelt werden. Dieser Vorgang kann beim Roulette-Selection-Algorithmus in logarithmischer Zeit mit einer binären Suche erfolgen, weshalb er in dieser Arbeit Verwendung gefunden hat.
Damit ist es möglich, die Nachfolger-Generation in

$$\mathcal{O}(2 \cdot |P| + |P| \cdot \log_2 |P|)$$

Zeit zu berechnen. Obwohl das Resampling nicht in jedem Frame stattfindet, sondern

nur nachdem der Roboter eine bestimmte Distanz zurückgelegt hat, ist es wichtig, dass dieser Vorgang nicht zu viel Zeit benötigt.

4.5 Schätzung der Position des Roboters

Nachdem das Resampling abgeschlossen ist, hat sich der Großteil der Partikel um den Punkt der höchsten Wahrscheinlichkeitsdichte von X_t gesammelt. Um nun wirklich das Maximum der Funktion aus dem Partikelschwarm zu extrahieren, gibt es mehrere Möglichkeiten. Ein naiver Ansatz wäre es, die Position des bestbewerteten Partikels als Resultat heranzuziehen. Allerdings kann dieses Verfahren sehr ungenau sein, denn es kann immer Fehler in den Daten des Observation Models geben und ein solcher Ansatz würde die vielen Informationen verwerfen, die in der Menge der Partikel vorhanden sind.

Eine andere Möglichkeit wäre, anfangs ein Gitter über das Spielfeld zu legen und am Ende einer Iteration von Algorithmus 1 (und damit eines Zeitschritts) zu schauen, in welcher Zelle sich die meisten Partikel gesammelt haben. Dieses *Binning* sorgt dafür, dass möglichst viele Informationen des Filters in das Resultat eingehen und kann die Ergebnisse der Selbstlokalisierung deutlich verbessern.[28] Da in aller Regel der Schwarm gegen eine oder auch zwei Positionen konvergiert, sollte es auch eine Zelle geben, in der sich der Großteil der Partikel akkumuliert. Die finale Position ist dann der Durchschnitt der sich in der Zelle befindlichen Hypothesen. Wenn also in der Zelle N Partikel vorhanden sind, gilt:

$$x_{final} = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad\qquad y_{final} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

Für die Winkel ist der Durchschnitt aufgrund der Wiederholung im Kreis etwas komplizierter, kann aber nach [20] wie folgt berechnet werden:

$$\theta_{final} = atan2(\sum_i sin\theta_i, \sum_i cos\theta_i) \tag{22}$$

Die finale Position $(x_{final}, y_{final}, \theta_{final})$ kann dann als die geschätzte Variable X_t übernommen werden.

Aber auch in der Schätzung der finalen Position lässt sich Expertenwissen nutzen. Denn ein humanoider Roboter kann sich in aller Regel in einem Frame nicht weit bewegen. Wird nun angenommen, dass die letzte Position korrekt war, dann muss für die aktuelle gelten, dass sie nicht weit von der letzten gültigen Position entfernt sein kann.
Konkret bedeutet das, dass die neue Position nicht weiter als ein bestimmter Grenzwert vom letzten Wert entfernt sein darf. Ansonsten wird die Schätzung zwar in die Richtung

der neuen Position bewegt, allerdings nur bis zu diesem Threshold.[28] Mathematisch lässt sich die Weite der Bewegung in Richtung $s_{\vec{final}}$ schreiben als:

$$d = \Delta(s_{\vec{final}}, s_{\vec{old}}).$$

Wenn diese Entfernung den Grenzwert *threshold* überschreitet, wird sie auf diesen Wert, also die maximal erlaubte Weite, beschränkt.

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

Angenommen, dass die Summe aller Gewichte etwas über die „Sicherheit“ der Vorhersage aussagen kann, lässt sich die Reichweite unseres Kandidaten auch dynamisch durch Multiplikation mit dem Mittelwert der Gewichte in der Zelle regeln:

$$p_{new} = p_{old} + \left(\frac{1}{|P_{bestCell}|} \sum_{\{\vec{s},\omega\} \in P_{bestCell}} \omega_i\right) \cdot d_s$$

Diese Annahme, dass höhere Beliefs auch eine bessere Vorhersage bedingen, war auch schon für den Augmented MCL-Algorithmus in Kapitel 2.2.5 wichtig.

In einer Anfangssituation des Spiels, wo vielleicht die vorherige Position p_{old} noch gar nicht gesetzt wurde, ist die Summe der Gewichte in der meist-bevölkerten Zelle eher gering (normalverteilt). Demnach kann sich die geschätzte Position anfangs weiter bewegen, als in einer Situation, in der der Schwarm der Partikel einen „Haufen“ bildet. Dort ist dann eine Bewegung nur sehr eingeschränkt möglich, was auch im Sinne einer akkuraten Selbstlokalisierung liegt.

4.6 Expertenwissen und bekannte Spielsituationen

Der Partikelfilter lässt sich weiter optimieren, auch für das Kidnapped Robot Problem. Wenn beispielsweise klar ist, dass der Roboter eine Strafe (*penalty*) bekommen hat, sind die Punkte bekannt, an denen er wieder ins Spiel gebracht werden darf. Ähnliches gilt auch für den Start des Spiels, denn je nach strategischer Position wie Torwart, Verteidiger oder Angreifer ist die Position auf dem Spielfeld anfangs mehr oder weniger klar.
Ein Vorteil des Partikelfilter-Ansatzes ist es, dass dieses „Expertenwissen“ leicht nutzbar gemacht werden kann. Denn anstatt zu Anfang eine Normalverteilung über dem Spielfeld anzunehmen und

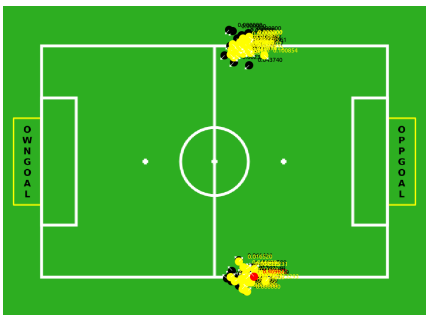


Abbildung 22: Partikelverteilung nach Penalty

nach und nach die Punkte mit einer hohen Wahrscheinlichkeitsdichte zu finden, können die Partikel an Orten erzeugt werden, an denen eine hohe Dichte direkt wahrscheinlich ist. Für einen Roboter mit Torwart-Strategie beispielsweise könnte dies das eigene Tor sein.
Durch die hohen Bewertungen der Partikel, konvergiert der Filter schnell an der richtigen Stelle.

4.7 Das Problem der Symmetrie

Durch die Änderung der Torfarbe beider Tore auf gelb (vgl. [19]) sind beide Spielhälften nun völlig symmetrisch. Das bedeutet, dass es nicht möglich ist, den Roboter „aus dem Nichts“ zu lokalisieren, es braucht immer eine vorherige Position, anhand derer entschieden werden kann, auf welcher Seite des Feldes sich der Roboter befindet.

Hierfür wurde ein neuer Wert eingeführt, der entscheidet, auf welcher Seite des Spielfeldes sich der Roboter befindet. Dieser gibt pro Partikel die Sicherheit, den *belief*, dafür an, sich auf der eigenen Hälfte des Feldes zu befinden. Jedes Partikel, was die Mittellinie überquert, ändert auch seinen belief. Wenn nun die endgültige Position geschätzt wird, so wird einmal der durchschnittliche Symmetrie-belief errechnet. Dieser kann dann mit der vorherigen Position und der Odometrie verglichen werden, um die tatsächliche Sicherheit zu errechnen, sich auf der eigenen Spielfeldhälfte zu befinden.

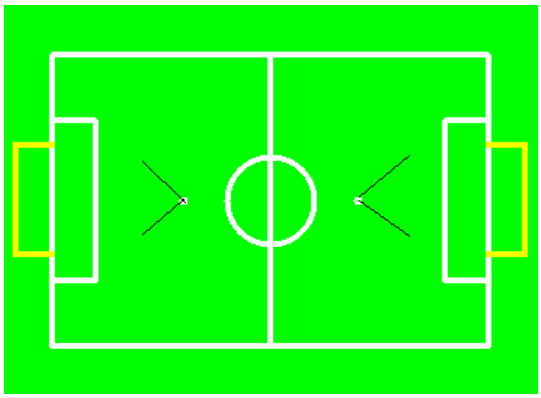


Abbildung 23: Die Symmetrie-Situation auf dem Spielfeld (die Linien stellen jeweils Blickwinkel dar)

Gleichzeitig werden die Partikel nur noch auf einer Spielfeldhälfte modelliert und sobald sie diese verlassen, werden sie so verschoben, dass sie mit invertierten Koordinaten und umgekehrter Rotation wieder in die gleiche Spielfeldhälfte eintreten. Der Symmetrie-belief wird ebenfalls invertiert.
Partikel, die beim Resampling in die Nähe von gut bewerteten anderen Partikeln kommen, erhalten ihren Symmetrie-belief auf Basis der absoluten Position, auf der sie sich befinden. Auf diese Art und Weise lässt sich auch in einer komplett symmetrischen Welt die Frage, auf welcher Spielfeldhälfte sich der Roboter momentan befindet, für das Tracking-Problem oft beantworten.
Dennoch gilt, dass das Kidnapped Robot Problem nicht komplett auflösbar ist, wenn der Roboter in der Nähe der Mittellinie fällt. Denn beim Aufstehen rotiert er oft, ohne dies

mitzubekommen. Dadurch kann es schnell zu einer Umstellung des Symmetrie-beliefs kommen, wodurch die Gefahr eines Eigentors erheblich steigt.
Allerdings könnte auch ein menschlicher Fußballspieler oder eine Fußballspielerin ohne weitere Hinweise nur schwer entscheiden, auf welcher Spielfeldhälfte er oder sie sich befindet, wenn er oder sie alle Erinnerung daran verloren hat, wie er oder sie auf seine momentane Position gekommen ist und keine anderen Spieler_innen auf dem Feld stehen, an denen er oder sie sich orientieren kann.

4.8 Probleme der angewandten Methode

Wir haben in den letzten Kapiteln gesehen, wie der Partikelfilter um viele Punkte erweitert werden kann. Ein Explorations-Faktor lässt das Entkommen aus lokalen Maxima zu [27], durch Optimierungen kann die typischerweise für dieses Verfahren hohe Laufzeit etwas verringert werden und die tatsächlich geschätzte Position des Roboters kann so berechnet werden, dass möglichst viele Informationen in die Schätzung einfließen.
Dennoch ist der gewählte Ansatz der Monte-Carlo-Localization nicht problemlos und gerade durch die Regeländerungen bezüglich der Torfarbe im letzten Jahr steht die Selbstlokalisierung der Roboter allgemein wieder vor neuen Herausforderungen. Von diesen negativen Aspekten des Partikelfilter-Ansatzes allgemein ist in dieser Arbeit vor allem einer aufgetreten, der hier noch kurz diskutiert werden soll.

4.8.1 Einstellung der Parameter

Trotz oder gerade wegen der verschiedenen Optimierungen des klassischen Partikelfilters ist die Anzahl der Parameter des Programms enorm. Dies führt zu Problemen, sobald sich ein Detail im Rest des Programms ändert, wie beispielsweise die Feldlinienerkennung oder die Art der Gewinnung von Odometrie-Daten. Denn für bessere Odometrie-Daten muss der Filter weniger oft den Resampling-Schritt ausführen und die Partikel dabei nicht so weit streuen. Wenn aber die Feldlinien sehr gut erkannt werden, ist es eventuell geschickt, die Partikel weiter auseinander treiben zu lassen und dann anhand des Observation Model zu entscheiden, wo sich der Roboter gerade befindet.
Es ist also oft ein Experte vonnöten, um alle Parameter, die auch teilweise voneinander abhängig sind, einzustellen, sie also zu tunen.

Dieser Effekt ist nicht wünschenswert und ein erheblicher Nachteil des Partikelfilters allgemein, wie schon von Röfer et al. in [3] beschrieben wurde. Eine potentielle Möglichkeit, hier eine globale Optimierung zu erzielen, wird in Kapitel 6 vorgestellt.

5 Resultate und Testergebnisse

Im folgenden Kapitel werden die Resultate und Testergebnisse der Arbeit diskutiert. Zunächst erfolgt eine qualitative Betrachtung des hier präsentierten Ansatzes. Anschließend werden noch die Auswirkungen der optimierten Kantenvergleiche im Vergleich zu den vorhergehenden Resultaten besprochen.
Abschließend wird die Berechnung der Partikel-Gewichte anhand von Kanten mit einer Bewertung anhand von Feldlinien-Punkten verglichen. Feldlinien-Punkte sind die erkannten Punkte im Kamerabild, auf deren Basis die Feldlinien zusammengefügt werden.

5.1 Die Qualität der Lokalisierung

Im Folgenden wurden zwei Testreihen für typische Anwendungen der Lokalisierung im RoboCup durchgeführt.

Da für eine zuverlässige Bestimmung der Position des Roboters auf dem Spielfeld sowohl das Motion Model und das Observation Model gut zusammenspielen müssen, wurden zwei spielnahe Szenarien getestet, die jeweils den Schwerpunkt auf eines der Modelle legen.
In einem ersten Schritt wurden dabei Testreihen im Simulator der FUmanoids durchgeführt. Hierbei sind vor allem die Daten des Motion Model fast perfekt und damit deutlich besser, als in der Realität. Auch die Feldlinien-Extraktion und damit das Observation Model können auf weniger verrauschte Daten zurückgreifen, denn während des Laufens ist die Kamera des Roboters in ständiger Bewegung, was oft zu Verzerrungen im Bild führen kann. Diese wirken sich direkt auf die Qualität der erkannten Feldlinien aus.
Anschließend wurde die Selbstlokalisierung noch auf dem Roboter direkt getestet.

5.1.1 Aufbau der Experimente

In einem ersten Experiment zur initialen Ortung soll der Roboter auf einer zufälligen Position auf dem Spielfeld stehen und sich dort ohne zusätzliche Informationen lokalisieren. Dabei kommt es lediglich auf das Observation Model an.
Dieser Test ist zwar kein vollständiges Kidnapped Robot Szenario, denn es fehlt die Erkennung einer Ortsänderung durch den Roboter. Dennoch kann durch das Experiment eine wichtige Eigenschaft der Selbstlokalisierung getestet werden, die auch im Spiel von Relevanz ist. Dieser Test ist allerdings einfacher operationalisierbar, als das plötzliche Verstellen des Roboters, bei dem die Zeit zur Relokalisierung zu großen Teilen von der Distanz zwischen den beiden Punkten abhängt.
Hier können eventuelle Daten des Motion Model nicht verwendet werden, denn es ist ja kein vorheriger Zustand bekannt, in dem sich der Roboter schon befunden hat.

Daher kommt es zuallererst auf die Qualität der Selbstlokalisierung an, also die Abweichung von der absoluten Position in Zentimetern. Zusätzlich wurde für jeden Test die Zeit bis zum Konvergieren der Partikel an einem Punkt gemessen. Drittes Kriterium war die Anzahl von verarbeiteten Bildern pro Sekunde, also die *frames per second*, oder *fps*.

Die zweite Testreihe behandelt anschließend das Tracking-Problem, also die „Verfolgung" des Roboters über einen gewissen Zeitraum hinweg. Eigentlich steht dem Labor der FUmanoids ein Groundtruth-System zur Verfügung, welches für Testzwecke die tatsächliche Position des Roboters auf dem Spielfeld von außen bestimmen kann.[17] Dieses System hätte sehr gut als Referenz für die Selbstlokalisierung fungieren können. Leider war das Gerät zum Zeitpunkt der Testreihe nicht funktionsfähig; somit konnte diese absolute Validierung der tatsächlichen Roboterposition nicht stattfinden. Stattdessen wurde die absolvierte Strecke jeweils manuell in die Bilder eingetragen, wodurch ein gewisser Fehler nicht auszuschließen ist.
Bei diesem Test wäre vor Allem eine Verfolgung der tatsächlichen, vom Roboter zurückgelegten, Strecke wünschenswert. Dass die vom System geschätzte Position in aller Regel etwas mehr springt, als die real gemessene Strecke, ist hierbei ein oft beobachteter Prozess in der Selbstlokalisierung. [17], [28]

Weiterhin war die Erkennung der Feldlinien zum Zeitpunkt der Tests nach einem längeren Umbau der gesamten „Vision", also der Algorithmen zur Extraktion sämtlicher Perzepte aus dem Kamerabild, noch nicht optimal. Daher konnte die Position des Roboters auf dem Spielfeld zur Lösung des Problems der initialen Ortung nicht komplett frei gewählt werden, denn es gab viele Orte, von denen aus die Kamera keine einzige Feldlinie erkennen konnte. Insbesondere aus diesem Grund ist die Analyse der hier vorgestellten Selbstlokalisierung nicht komplett aussagekräftig. Deutlich bessere Ergebnisse wären zu erwarten, gelänge es, die Feldlinien noch besser zu extrahieren.

5.1.2 Tests im Simulator

Initiale Ortung Hier wurde der Roboter auf eine zufällige Position auf dem Spielfeld gesetzt. Es wurde allerdings darauf geachtet, dass von dieser aus eine ausreichende Anzahl von Linien gesehen wurde, die der Lokalisierung einige Anhaltspunkte geben konnte.
Anschließend wurde das Programm so aufgerufen, dass in jedem Frame das Resampling stattfand. Normalerweise wird dieses nur nach einer gewissen zurückgelegten Strecke ausgeführt. Wie bereits erwähnt, dürfen an die unter Simulatorbedingungen erzielten Ergebnisse wegen der besseren Erkennung der Feldlinien deutlich höhere Erwartungen gestellt werden.

Insgesamt wurden für das Problem der initialen Ortung zehn Testläufe durchgeführt, bei denen der durchschnittliche Fehler mit ca. 11 cm sehr niedrig ist (siehe Abbildung 1).
Die Zeit für die Selbstlokalisierung liegt bei durchschnittlich knapp 6 Sekunden, was vor allem der anfangs etwas trägen Bewegung durch das in Kapitel 4.5 Verfahren bespro-

Tabelle 1: Die Resultate der initialen Ortung im Simulator

Nummer	benötigte Zeit (in s)	Fehler (in cm)
1	2.08	9.2
2	2.27	12.16
3	4.36	5.6
4	7.68	5.8
5	5.61	31.0
6	4.44	7.2
7	16.47	12.3
8	3.2	6.08
9	6.47	14.0
10	5.2	4.12
TOTAL	57.78	107.46
DURCHSCHNITT	5.778	10.746

chene Verfahren geschuldet ist. Denn die maximal erlaubte Änderung in der Position hängt von der Bewertung jedes Partikels ab, welches in die Schätzung einfließt. Da die Unsicherheit über die tatsächliche Position des Roboters anfangs groß ist, darf sich die Schätzung nicht weit bewegen und kommt erst nach einiger Zeit am Schwarm der Partikel an.[28] Tatsächlich sind die Partikel meist nach 1-2 Frames konvergiert (also insgesamt

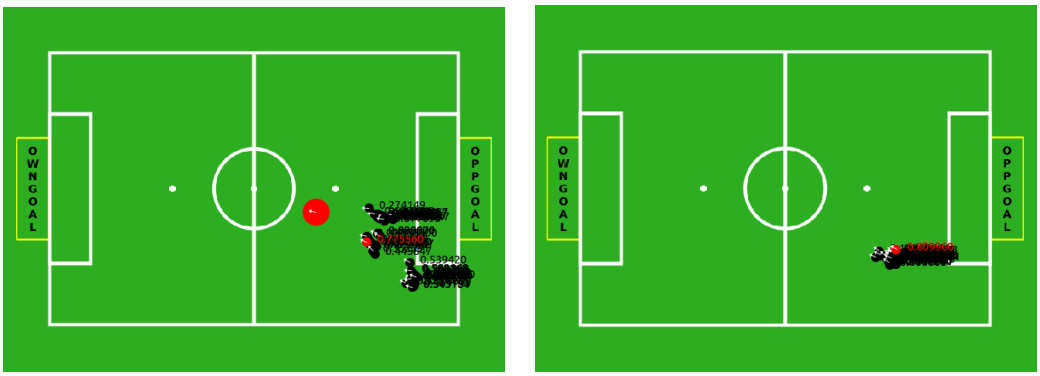


Abbildung 24: Die Bewegung der Partikel nach 1 bzw. 2 Resampling Durchläufen (insgesamt 14 Iterationen des Resampling-Schritts). Die Partikel sind als schwarze Punkte mit ihrer Orientierung visualisiert, während das best-bewertete Partikel als kleiner roter Punkt erscheint.
Die aktuelle Schätzung der eigenen Position durch den Filter wird als großer roter Punkt dargestellt.

14 Aufrufe des Resamplings), wie Abbildung 24 zeigt. Dabei beschreiben die kleinen, schwarzen Punkte die einzelnen Partikel, während der kleine, rote Punkt das momentan best-bewertete Partikel und der große rote Punkt die finale Schätzung symbolisiert. Für

das rechte Bild wurde diese Schätzung weggelassen, da sie sich am gleichen Ort wie der Schwarm der Partikel befand und sonst die Lesbarkeit der Abbildung stark eingeschränkt worden wäre.

Tracking Beim Tracking wurden zum Start des Programms alle Partikel an den Ort gesetzt, an dem sich der Roboter tatsächlich befand. Anschließend ist er eine Zeit lang gelaufen, wobei im Simulator die Groundtruth-Position als Referenz dient.
Dieses System ist im Simulator verfügbar, während es für Tests auf dem Roboter selbst nicht zur Verfügung stand.

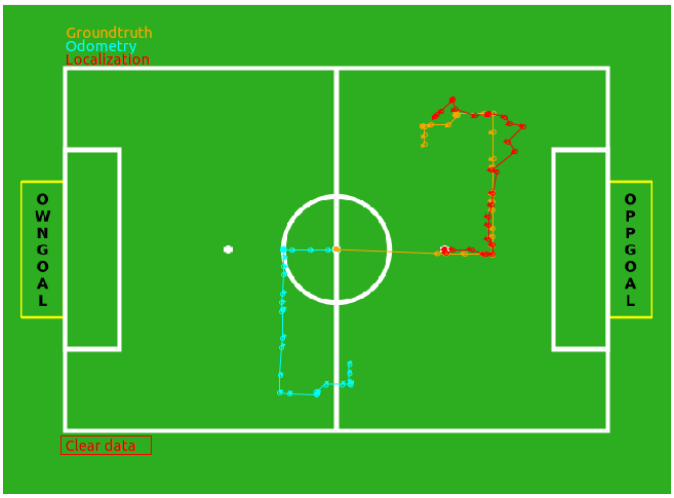


Abbildung 25: Das Tracking im Simulator mit dem zurückgelegten Pfad in orange und der jeweiligen Schätzung in rot.

In der Abbildung 25 ist der vom Roboter zurückgelegte Weg im Vergleich mit dem Groundtruth-System zu erkennen. Der rote Weg symbolisiert die Schätzung des Roboters, während der orangefarbene Pfad den Groundtruth-Daten entspricht, also dem tatsächlichen Pfad. In Blau ist weiterhin die Odometrie zu erkennen, welche aber von einer Ausgangsposition auf dem Mittelpunkt mit Drehung hin zum rechten Tor ausgeht. Dennoch lässt sich hier sehen, dass die Daten der Odometrie im Simulator fast perfekt sind, während dies in der Realität selten der Fall ist.

Da die Partikel sich anfänglich auf der richtigen Position befanden, ist der Fehler auch zunächst im sehr kleinen Bereich und wird erst nach einigen Resampling-Schritten größer. Es ist zu sehen, dass der Partikelfilter in diesem Szenario kaum Sinn ergibt, denn trotz sehr guter Daten ist die Schätzung über einen längeren Zeitraum hinweg nicht immer perfekt.

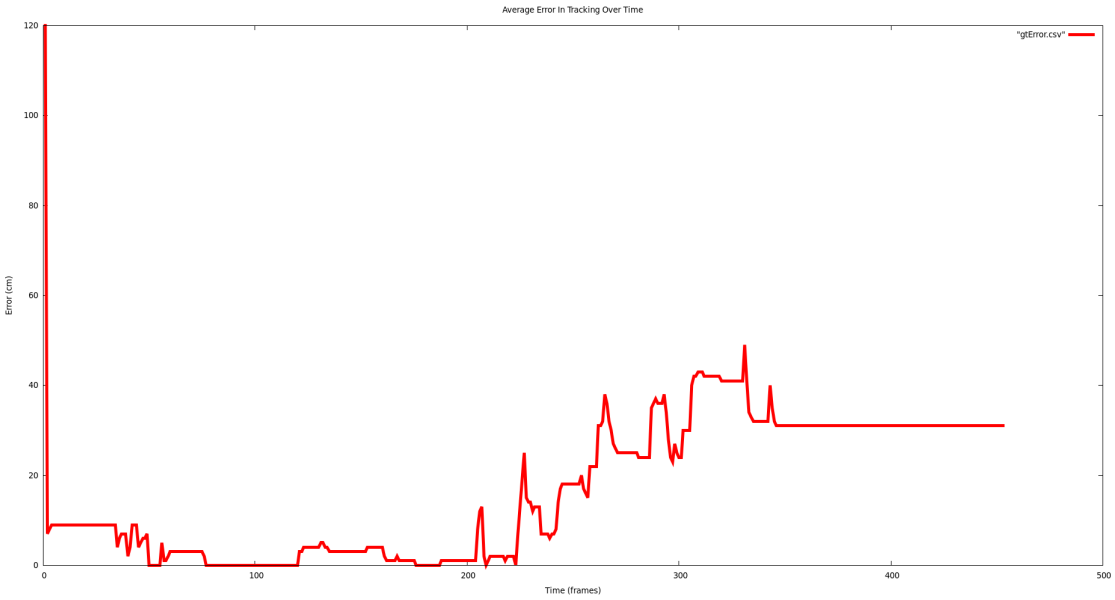


Abbildung 26: Die Abweichung von Groundtruth und tatsächlichem Weg. In der x-Achse sind die Anzahl der Frames zu erkennen, während die y-Achse den Fehler in cm beschreibt.

Noch besser lässt sich dieser Effekt in Abbildung 26 beobachten. Denn hier ist der Fehler über die Zeit zu erkennen, wobei auf der x-Achse die Zeit in Frames und auf der y-Achse der Fehler in Zentimeter zu sehen ist. Es ist offensichtlich, dass das Tracking mit der Zeit bei diesem Testlauf schlechter wird, da sich das System hier nicht genug auf die Odometrie verlässt. Die erste Schätzung ist hier ein Ausreißer und der Initialisierung des Filters geschuldet.Wenn diese gut ist, sollte der Partikelfilter eigentlich nur sie zur Selbstlokalisierung nutzen, da es schwieriger ist, aus den Kamerabildern die Position zu extrahieren.

5.1.3 Tests auf den Robotern

Wie bereits erwähnt, war das Testen auf dem Roboter durch die etwas weniger optimale Feldlinienerkennung mit einigen Problemen behaftet.
Davon war vor allem das Problem der initialen Ortung betroffen, für dessen Lösung die Feldlinien den einzigen Anhaltspunkt darstellen.

Initiale Ortung Wie auch schon im Simulator wurde der Roboter auf eine zufällige Position auf dem Spielfeld gestellt, von der aus er einige markante Feldlinien erkennen konnte. Durch Probleme mit der Feldlinienerkennung wurden diese Orte jedoch noch

Tabelle 2: Die Resultate der initialen Ortung auf dem Roboter

Nr.	benötigte Zeit (in s)	Genauigkeit (in cm)	frames per second (fps)
1	3	14.8	23.9
2	4	89.8	21.6
3	4	23.3	20.9
4	7	49.7	21.3
5	2	13.1	20.0
TOTAL	20	190.7	107.7
DURCHSCHNITT	4	38.14	21.5

weiter eingeschränkt, damit eine Selbstlokalisierung überhaupt möglich war. Im Gegensatz zu der entsprechenden Testreihe im Simulator wurde sich beim Roboter auf fünf Tests beschränkt, da von vielen Orten auf dem Spielfeld als Ausgangspunkt aufgrund mangelnder Erkennung der Feldlinien abgesehen werden musste.

Nach Positionierung des Roboters lief er solange auf der Stelle, bis die Partikel an einem Ort konvergiert waren. Dabei war es aus technischen Gründen nicht möglich, dass die Kamera sich automatisch bewegt (*gaze*), wie es im normalen Spielverlauf der Fall gewesen wäre. Dies wurde durch manuelle Bewegung der Kamera zu bereinigen versucht.

In der Tabelle 2 sind die Resultate des Tests erkennbar. Während der durchschnittliche Fehler mit knapp 40 cm um einiges höher liegt als im Simulator, ist die Zeit bis zum Konvergieren der Partikel fast identisch und wieder der trägen Bewegung der Schätzung am Anfang geschuldet.
Interessant ist weiterhin, dass obwohl sämtliche Optimierungen ausgeschaltet waren und der Algorithmus dadurch eine annähernd quadratische Laufzeit besitzt, die Anzahl der Frames mit durchschnittlich mehr als 21 fps recht hoch liegt. Die Frame-Rate liegt ohne die Lokalisierung bei 30 fps und wird darauf auch begrenzt.

Tracking Für das Tracking-Problem wurde ebenso vorgegangen wie bei den Tests im Simulator. Insgesamt wurde eine Reihe von fünf Tests durchgeführt. Für alle von ihnen wurde der Algorithmus ohne Laufzeit-Optimierung genutzt.
Beispielhaft ist in Abbildung 27 der Vergleich von Schätzung und real zurückgelegtem Weg zu erkennen.

Aufgrund des nicht funktionsfähigen Groundtruth-Systems ist der schwarze Weg, also der reale Weg des Roboters, manuell eingetragen worden. Dementsprechend ist hier ein potentieller Fehler in den Tests vorhanden. In der Grafik ist zu erkennen, dass die Odometrie insbesondere mit Rotationen Probleme hat. Der Schwenk des Roboters um 90 Grad, bei dem er von seitlichem Laufen in Vorwärtslaufen übergegangen ist, wird nur aus den Odometrie-Daten nicht erkennbar. Die Lokalisierung kann das jedoch ausgleichen

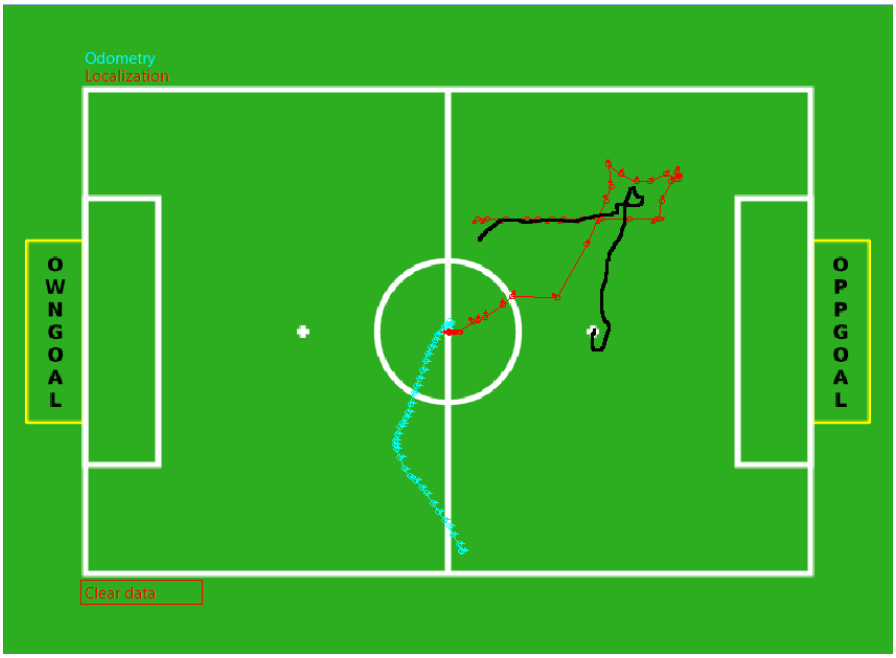


Abbildung 27: Ein Testlauf für das Tracking auf dem Roboter. Rot zeigt die geschätzte Route des Roboters, während schwarz den tatsächlich zurückgelegten Weg markiert. Die Odometrie ist in blau abgebildet.

und gegen Ende eine fast perfekte Ortung des Roboters erreichen.
Da die Schätzung der Position immer auf dem Mittelpunkt anfängt, ist es nicht beunruhigend, dass hier am Anfang sehr schlechte Ergebnisse erzielt wurden. Denn wie schon besprochen, muss erst eine gewisse Sicherheit im System vorhanden sein, damit sich die Schätzung der Position bewegt.

5.2 Qualitätseinbußen

Durch den heuristischen Kantenvergleich kann es zu Abstrichen in der Qualität der Lokalisierung kommen, wie bereits in Kapitel 4.3 beschrieben wurde. Wie genau dieser Verlust von Qualität im Verhältnis zur gewonnenen Ausführungszeit der Selbstlokalisierung steht, wurde in diesem Kapitel versucht, herauszufinden.

Weiterhin werden momentan die Feldlinien extrahiert, indem zunächst Punkte einer Linie im Kamerabild erkannt werden. Diese Punkte müssen anschließend aber mit einem relativ hohen Aufwand zu einer Feldlinie zusammengefügt werden. Es gäbe die Möglichkeit, dass das Observation Model nur mit diesen Punkten arbeitet. Aus diesem Grunde wurde ein Kantenvergleich auf Basis der extrahierten Punkte implementiert, der hier als Referenz zum in dieser Arbeit vorgestellten Algorithmus dienen soll. Diese werden im folgenden

Kapitel untersucht, wobei die Frage im Vordergrund steht, ob sie die Methoden wirklich lohnen.

5.2.1 Auswirkung der heuristischen Partnersuche für Kanten

Die in Kapitel 4.3 vorgestellte Partnersuche zwischen einer gesehenen und einer erwarteten Kante kann theoretisch zu schlechteren Ergebnissen in der Selbstlokalisierung führen. Denn die Partner werden anhand eines Schlüssels überprüft, der nicht eindeutig sein muss. Für die hier durchgeführten Tests auf dem Roboter wurden für eine gesehene Kante sechs potentielle Partner durch die heuristische Suche gefunden. Diese sechs Kanten wurden anschließend mit der gesehenen Feldlinie verglichen. Als Partner wurde diejenige Kante mit dem geringsten Abstand δ ausgewählt.
Aus dem Testlauf in Abbildung 28 wird erkennbar, dass die heuristische Partnersuche

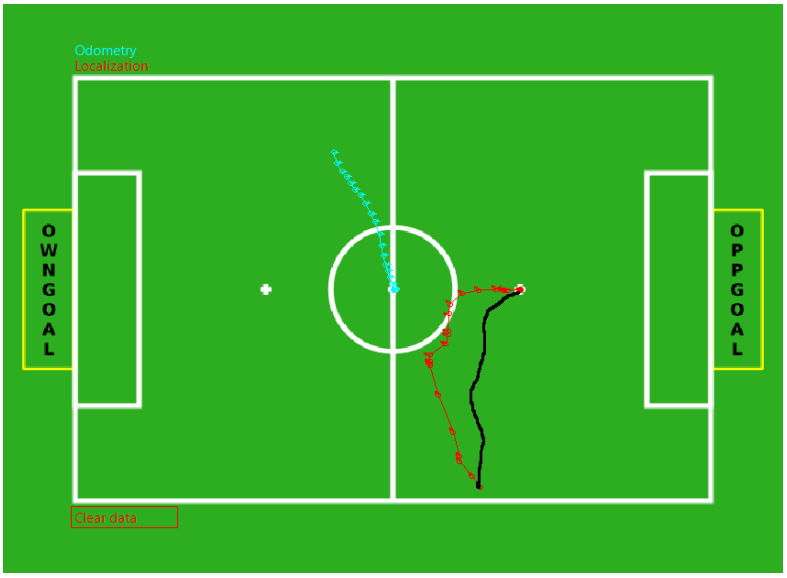


Abbildung 28: Ein Testlauf für das Tracking auf dem Roboter mit Optimierung durch die heuristische Partnersuche für Kanten. Rot symbolisiert die Ergebnisse der Selbstlokalisierung, während der reale Weg in schwarz eingetragen wurde. Odometrie-Daten sind in blau eingezeichnet.

nicht zwangsläufig zu schlechteren Ergebnissen führen muss. Dennoch wird die Position des Roboters zu Anfang eher schlecht approximiert, während gegen Ende die Ergebnisse besser werden.
Im Gegensatz zu den anderen Testläufen wurde hier die Anfangsschätzung bereits auf die tatsächliche Position des Roboters gesetzt, wodurch eigentlich bessere Schätzungen für den Beginn des Tests möglich gewesen wären.

Tabelle 3: Fünf Testläufe mit heuristischer Partnersuche für die Kanten und den Frame-Raten pro Versuch

Nummer	Frames Per Second
1	24.6
2	19.8
3	23.4
4	22.0
5	24.5
TOTAL	114.3
DURCHSCHNITT	22.8

Tabelle 3 beschreibt für fünf verschiedene Testläufe mit einem ähnlichen Szenario wie das oben genannte, mit wie vielen Frames pro Sekunde das Programm lief. Dass hier die Frame-Raten nur marginal höher sind, als ohne eingeschaltete Optimierung liegt vor Allem daran, dass zwischen den beiden Tests die Erkennung der Feldlinien verändert wurde. Inzwischen ist es möglich, mehr Kanten zu sehen, allerdings wurden auch deutlich mehr Feldlinien doppelt erkannt als vorher, was zu höheren Laufzeiten führte.

5.2.2 Punktweise vs. linienbasierte Hypothesenbewertung

In der Literatur wurden die Feldlinien bisher meist als einzelne Punkte mit den Linien der hypothetischen Sicht verglichen.[23] Dieser Vergleich bringt einige Vorteile mit sich, vor allem müssen die Feldlinien nicht erst aufwendig aus den Punkten zusammengefügt werden. Allerdings können ganze Linien oft deutlich mehr Informationen für die Selbstlokalisierung bringen, als einfache Punkte.
Im Folgenden wurde die Bewertung auf Basis der erkannten Punkte als Referenz implementiert und in den selben Szenarien wie in Abschnitt 5.1.1 beschrieben, auf dem Roboter getestet.

Initiale Ortung Wie aus Abbildung 4 erkennbar, sind zunächst die Fehler bei dieser Form der Partikelbewertung deutlich größer, als bei den anderen Tests. Dies liegt zum Einen daran, dass Vieldeutigkeiten leichter entstehen, da die Richtung der Punkte nicht so klar wie bei den Feldlinien ist. Zwar existiert für jeden Punkt eine Steigung, die auch in die Bewertung eingeflossen ist. Dennoch ist bei einer ganzen Feldlinie die Information darüber viel akkurater. Weiterhin ist erkennbar, dass die Frame-Rate scheinbar mit der Qualität der Lokalisierung korrespondiert, wie in Testlauf 4 und 5 erkennbar. Dies erscheint logisch, denn die Frame-Rate hängt stark von der Anzahl der erkannten Punkte ab, was für die Qualität ebenfalls ein entscheidender Faktor ist. Leider bedeutet dies auch, dass wenn die Qualität besser wird, sich die Anzahl der verarbeiteten Bilder pro

Tabelle 4: Fünf Tests der punktweisen Vergleiche mit der hypothetischen Sicht eines Partikels.

Nummer	Zeit (in s)	Fehler (in cm)	FPS
1	4.5	75.6	20.2
2	11.8	53.2	20.7
3	9.8	197.1	20.1
4	4.5	45.0	17.2
5	9.5	20.0	16.9
TOTAL	40.1	390.9	95.1
DURCHSCHNITT	8.02	78.18	19.02

Sekunde verringert.

Tracking Es wurden insgesamt drei Testläufe auf dem Roboter durchgeführt, in denen jeweils 20-30 Sekunden lang gelaufen wurde. Dabei wurde immer vom Penalty-Point auf der eigenen Seite gestartet und in eine Richtung seitwärts gelaufen. Es wurde darauf geachtet, dass die Gaze bewegt wurde und stets viele Feldlinien im Blickfeld der Kamera waren. Das Ergebnis einer der drei Testläufe ist in Abbildung 29 zu erkennen. Anfangs

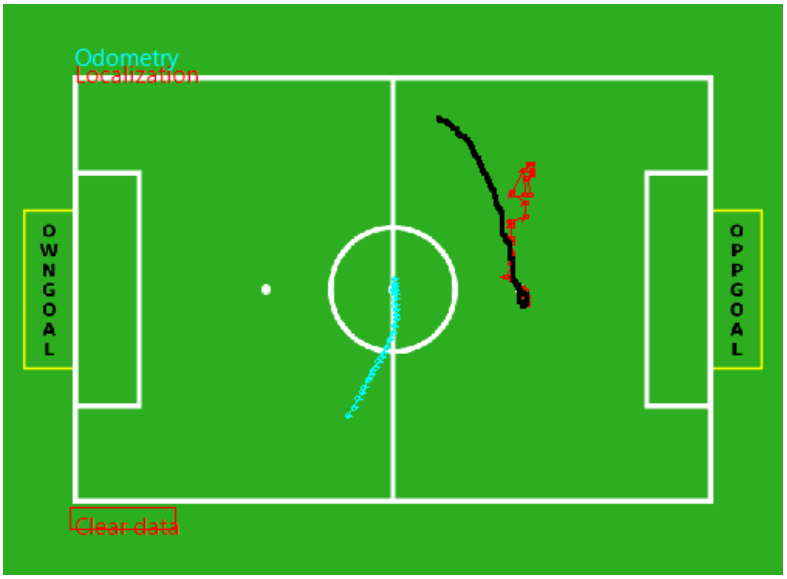


Abbildung 29: Ein Testlauf für das Tracking auf dem Roboter punktweiser Bewertung der Partikel. Rot symbolisiert die Ergebnisse der Selbstlokalisierung, während der reale Weg in schwarz eingetragen wurde. Odometrie-Daten sind in blau eingezeichnet.

waren alle Partikel schon an der richtigen Position und es lässt sich erkennen, dass auch die ersten Resamplings noch keine großen Abweichungen vom realen Weg zeigen. Erst nach einer Weile bewegt sich die Schätzung nur noch minimal, während der Roboter eigentlich weiter lief.
Dies ist typisch für die Bewertung der Partikel auf der Basis von Punkten, denn der Roboter verändert seine Distanz zum Mittelkreis nicht viel und nur die Mittellinie selbst gibt hier Auskunft über die Position auf der Kreisbahn. Dies ist durch die einzelnen Punkte nicht gut zu erkennen, während im Idealfall einer linienbasierten Bewertung dieses Problem nicht entstanden wäre.

6 Fazit und Ausblick

In den vorherigen Kapiteln wurde die erarbeitete Lösung für das Problem der Selbstlokalisierung vorgestellt und anschließend in ihrer Qualität bewertet.
Im Folgenden werden Möglichkeiten und Ansätze vorgestellt, die für weiterführende Untersuchungen interessant erscheinen. Abschließend wird ein Fazit aus dieser Arbeit gezogen.

6.1 Ausblick

Die hier vorgestellte Lösung des Lokalisierungs-Problems stellt zwar eine deutliche Verbesserung der bisherigen Selbstlokalisierung bei den FUmanoids dar, da sie das Potential hat, mit der Symmetrie umzugehen und nur mit den Feldlinien auskommt. Dennoch hat sie noch das Potential für Verbesserungen, um eine stabilere und weniger fehlerbehaftete Ortung des Roboters zu ermöglichen. Einige dieser Ansätze sollen nun vorgestellt werden.

6.1.1 Verbesserung der Daten

Zunächst ist es offensichtlich, dass eine Verbesserung der Ausgangsdaten direkt auch die Qualität der Lokalisierung erhöht. Dies war bei den Tests im Simulator zu beobachten, bei denen das Motion Model perfekte Daten lieferte, aber auch die Erkennung der Feldlinien deutlich besser als auf dem Roboter war. Der Fehler lag dort durchschnittlich bei ca. 10 cm, während er auf dem Roboter knapp 40 cm aufwies.

Insbesondere die Feldlinien könnten in Anzahl und Qualität eventuell noch verbessert werden, während die Odometrie nur schwer bessere Daten über die Bewegung des Roboters generieren kann. Denn hier kommen Faktoren wie Schlupf und Reibung ins Spiel, die bisher bei den FUmanoids nicht modelliert werden.

Insbesondere mit einer eventuell anstehenden Regeländerung zur Spielfeldgröße müsste die Distanz, in der Objekte noch wahrnehmbar sind, verbessert werden. Denn momentan werden kaum Feldlinien erkannt, die weiter als einen bis anderthalb Meter entfernt sind. Sollte das Spielfeld im nächsten Jahr tatsächlich größer werden, so wird die Lokalisierung mit „Blindflügen“ zu kämpfen haben, während derer keine Feldlinie sichtbar ist. Sich in diesem Zeitraum nur auf die Odometrie zu verlassen, erscheint nach bisherigem Erkenntnisstand schwierig, gerade für Spielsituationen, in denen der Roboter genau wissen muss, wo er ist.

6.1.2 Automatische Einstellung der Parameter

Wie bereits in Kapitel 4.8.1 erwähnt worden ist, sind zur korrekten Einstellung des Partikelfilters viele verschiedene Parameter notwendig. Selbst wenn eine sehr gute Einstellung manuell möglich wäre, müsste diese jeweils nach Änderungen von anderen Modulen neu durchgeführt werden. Beispielsweise ist es denkbar, dass die Roboter irgendwann über eine Schrittplanung verfügen. Damit wären deutlich bessere Odometrie-Daten zu erwarten, was wiederum zu einer Neueinstellung des Filters führen müsste.

Eine automatische Einstellung aller Parameter des Partikelfilters könnte zum Beispiel durch den in [3] vorgestellten Ansatz möglich werden. Durch die *particle swarm optimization* werden alle Parameter eines beliebigen Systems optimiert, solange ein Satz von Parametern in seiner Qualität bewertet werden kann.
Im Falle der Selbstlokalisierung ist das kein Problem, denn mit einem vorhandenen Groundtruth-System kann sehr genau bestimmt werden, wie gut eine Lösung ist. Der particle swarm optimizer könnte also eine Einstellung aller Parameter vornehmen, die für die Lokalisierung relevant sind.

6.1.3 Selbstbewertende Lokalisierung

Während die bisher vorgestellten Ansätze vor Allem auf eine Verbesserung der Qualität des Partikelfilters abzielten, wäre auch ein selbstbewertender Ansatz ein interessanter Untersuchungsgegenstand.
Wir hatten ja bereits gesehen, dass durchaus Aussagen über die Sicherheit im Lokalisierungs-System getroffen werden können, um den Explorationsfaktor dynamisch zu regeln. Weiter könnte damit auch bewertet werden, wie sicher sich der Algorithmus bei seiner Schätzung insgesamt ist. Diese Information könnte anschließend von anderen Modulen genutzt werden. Das Team B-Human nutzt beispielsweise einen Kalman-Filter, um die Schätzung nachgehend zu glätten.[21]
Dieser Ansatz ist stark mit einer aktiven Selbstlokalisierung verknüpft, wie in [17] vorgestellt wurde. Denn bei sinkender Sicherheit könnten zunächst auch eigene Maßnahmen ergriffen werden, wie die Kopfsteuerung des Roboters zu regeln. Wenn die Kamera nun Positionen fokussiert, die der Lokalisierung viele Anhaltspunkte über den Aufenthaltsort des Roboters geben, ließe sich die Qualität insgesamt verbessern. Und auch eine Lokalisierung im Team würde dadurch möglich, bei der sich die Roboter gegenseitig Approximationen der Position aller anderen Spieler schicken.

6.1.4 Selbstlokalisierung im Team

Einer der erfolgversprechendsten Ansätze um die Selbstlokalisierung der Roboter zu verbessern, ist wohl die Team-Lokalisierung. Hier würden alle Spieler in regelmäßigen Ab-

ständen die geschätzten Positionen ihrer Mitspieler über WLAN kommunizieren. Dabei hängt natürlich die absolute Position eines anderen Spielers auch von der eigenen Selbstlokalisierung ab, weswegen ein solches System unbedingt eine Bewertung der eigenen Schätzung erforderlich macht.
Wenn der Grad der Sicherheit Sicherheit zusätzlich kommuniziert würde, könnte ein Roboter immer auf dessen Basis entscheiden, ob er dem Vorschlag seiner Mitspieler über seinen Ort folgt, oder ob er sich sicher genug ist, um diese zu ignorieren.

6.2 Fazit

In dieser Arbeit wurde ein Ansatz zur Selbstlokalisierung humanoider Fussball-Roboter für das Team FUmanoids vorgestellt. Mithilfe eines Partikelfilters werden eine Vielzahl von Hypothesen gleichzeitig verwaltet und in jedem Frame wird die wahrscheinlichste Position des Roboters ausfindig gemacht.
Während dieser von mehreren Teams verwendet wird und inzwischen den Standard-Ansatz zur Selbstlokalisierung im RoboCup darstellt, wurde in dieser Arbeit eine Feldlinien-basierte Selbstlokalisierung untersucht. Diese stellen bisher ein eher untypisches Landschaftsmerkmal dar, um die eigene Position auf dem Spielfeld zu bestimmen.
Da die Linien aber in den meisten Spielsituationen sichtbar sind und auch durch ihre Länge und Position im Bild oft eindeutige Rückschlüsse auf den Ort der Kamera erlauben, sind sie ein entscheidendes Landschaftsmerkmal. Auch ihre Unabhängigkeit von eventuellen Regeländerungen in der Zukunft ist gewährleistet, da Feldlinien einen unabkömmlichen Teil des Spielfeldes darstellen.

Jedoch muss für die Erkennung und den Vergleich von ganzen Linien im Feld ein gewisser Mehraufwand betrieben werden, da jede Linie zunächst aus vielen einzelnen Punkten zusammengefügt werden muss. Dieser kann aber durch verschiedene Laufzeit-Optimierungen ausgeglichen werden und fällt somit nicht so stark ins Gewicht.

Inzwischen ist die Selbstlokalisierung einsatzfähig, wie das Kapitel 5 gezeigt hat, auch wenn noch Potential für Verbesserungen vorhanden ist. Sie kann eventuell mit noch weniger Partikeln oder selteneren Gewichts-Updates derselben ausgestattet werden könnte, um die Laufzeit geringer zu halten. Denn die momentan in jedem Frame stattfindenden Updates der Partikelgewichte stellen den teuersten Anteil des hier präsentierten Ansatzes zur Selbstlokalisierung dar.

Abbildungsverzeichnis

Literatur

[1] Bäck, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford University Press, Oxford, UK, 1996.

[2] Behnke, S., Missura, M., and Schulz, H. NimbRo TeenSize 2012 Team Description. Tech. rep., Rheinische Friedrich-Wilhelms-Universität Bonn, 2012.

[3] Burchardt, A., Laue, T., and Röfer, T. Optimizing Particle Filter Parameters for Self-Localization. In *RoboCup 2010: Robot Soccer World Cup XIV* (2011), J. R. del Solar, E. Chown, and P. G. Ploeger, Eds., vol. 6556 of *Lecture Notes in Artificial Intelligence*, Springer; Heidelberg; http://www.springer.de/, pp. 145–156.

[4] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. *Introduction To Algorithms.* MIT Press, 2001.

[5] Dellaert, F., Burgard, W., Fox, D., and Thrun, S. Using the condensation algorithm for robust, vision-based mobile robot localization. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.* (1999), vol. 2, pp. –594 Vol. 2.

[6] Doucet, A., and Johansen, A. M. A tutorial on particle filtering and smoothing: fifteen years later, 2011.

[7] Fischer, B., Heinrich, S., Hohl, G., Lange, F., Langner, T., Mielke, S., Moballegh, H. R., Otte, S., Rojas, R., von Schmude, N., Seifert, D., Steig, D., and Weissgerber, T. FUmanoid Team Description Paper 2010. Tech. rep., Freie Universität Berlin, 2010.

[8] Forsyth, D. A., and Ponce, J. *Computer Vision: A Modern Approach.* Prentice Hall Professional Technical Reference, 2002.

[9] Gamerman, D., and Lopes, H. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference.* Chapman and Hall/CRC Texts in Statistical Science Series. Taylor & Francis, 2006.

[10] Gutmann, J.-S., and Fox, D. An experimental comparison of localization methods continued. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on* (2002), vol. 1, pp. 454–459 vol.1.

[11] Julier, S., and Uhlmann, J. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE 92*, 3 (2004), 401–422.

[12] Kalman, R. E. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME - Journal of Basic Engineering*, 82 (Series D) (1960), 35–45.

[13] Kuhn, J., Kohlbrecher, S., Petersen, K., Scholz, D., Wojtusch, J., and von Stryk, O. Team Description for Humanoid KidSize League of RoboCup 2012 . Tech. rep., Technische Universität Darmstadt, 2012.

[14] Langner, T. Selbstlokalisierung für humanoide Fussballroboter mittels Mono- und Stereovision. Diplomarbeit, Freie Universität Berlin, Fachbereich Mathematik und Informatik, September 2009.

[15] MacKay, D. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[16] Moravec, H. *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press, 1988.

[17] Otte, S. Where Am I? What's going on? - World Modelling using Multi-Hypothesis Kalman Filters for Humanoid Soccer Robots. Master's thesis, Freie Universität Berlin, 2012.

[18] Quinlan, M., and Middleton, R. Multiple Model Kalman Filters: A Localization Technique for RoboCup Soccer. In *RoboCup 2009: Robot Soccer World Cup XIII*, J. Baltes, M. Lagoudakis, T. Naruse, and S. Ghidary, Eds., vol. 5949 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 276–287.

[19] RoboCup Soccer Humanoid League Technical Committee. Robocup soccer humanoid league rules and setup for the 2013 competition in eindhoven (changes marked), 2013.

[20] Röfer, T., and Jüngel, M. Fast and Robust Edge-Based Localization in the Sony Four-Legged Robot League. In *In 7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence* (2004), Springer, pp. 262–273.

[21] Röfer, T., Laue, T., Müller, J., Burchardt, A., Damrose, E., Fabisch, A., Feldpausch, F., Gillmann, K., Graf, C., de Haas, T. J., Härtl, A., Honsel, D., Kastner, P., Kastner, T., Markowsky, B., Mester, M., Peter, J., Riemann, O. J. L., Ring, M., Sauerland, W., Schreck, A., Sieverdingbeck, I., Wenk, F., and Worch, J.-H. B-Human Team Report and Code Release 2010, 2010.

[22] Röfer, T., Laue, T., Müller, J., Graf, C., Böckmann, A., and Münder, T. B-Human Team Description for RoboCup 2012. In *RoboCup 2012: Robot Soccer*

World Cup XV Preproceedings (2012), X. Chen, P. Stone, L. E. Sucar, and T. V. der Zant, Eds., RoboCup Federation.

[23] Röfer, T., Laue, T., and Thomas, D. Particle-filter-based self-localization using landmarks and directed lines. In *RoboCup 2005: Robot Soccer World Cup IX*. Springer, 2006, pp. 608–615.

[24] Rohloff, J., von Geyso, N., and Schulte-Sasse, R. Hypothesen-Partikelfilter - Software-Praktikumsbericht. *Freie Universität Berlin* (2012).

[25] Stronger, D., and Stone, P. Expectation-based vision for self-localization on a legged robot. In *proceedings of the 21st national conference on Artificial intelligence - Volume 2* (2006), AAAI'06, AAAI Press, pp. 1899–1900.

[26] ten Velthuis, D., Verschoor, C., Wiggers, A., van der Molen, H., Blankenvoort, T., Cabot, M., Keune, A., Nugteren, S., van Egmond, H., Rozeboom, R., Becht, I., de Jonge, M., Pronk, R., Kooijman, C., Slaap, R., and Visser, A. Team Description for Robocup 2013. Tech. rep., Universiteit van Amsterdam and TU Delft, November 2012.

[27] Thrun, S., Burgard, W., and Fox, D. *Probabilistic Robotics*. Intelligent robotics and autonomous agents series. Mit Press, 2005.

[28] von Schmude, N. Selbstlokalisierung Humanoider Roboter im RoboCup - Studienarbeit. *Freie Universität Berlin* (September 2011).