UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

AGAS
Arbeitsgruppe Aktives Sehen

Implementieren des FastSLAM Algorithmus zur Kartenerstellung in Echtzeit

Studienarbeit
im Studiengang Computervisualistik

vorgelegt von

Marco Mengelkoch

Betreuer:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Erstgutachter:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Zweitgutachter:	Dipl.-Inf. Johannes Pellenz, Institut für Computervisualistik, Fachbereich Informatik

Koblenz, im Januar 2007

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den ..... Unterschrift

Inhaltsverzeichnis

UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

AGAS
Arbeitsgruppe Aktives Sehen

Implementieren des FastSLAM Algorithmus zur Kartenerstellung in Echtzeit

Studienarbeit
im Studiengang Computervisualistik

vorgelegt von

Marco Mengelkoch

Betreuer:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Erstgutachter:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Zweitgutachter:	Dipl.-Inf. Johannes Pellenz, Institut für Computervisualistik, Fachbereich Informatik

Koblenz, im Januar 2007

Kapitel 1

Einleitung - Das SLAM Problem

SLAM (simultaneous localization and mapping) ist die Problemstellung, dass ein Roboter sich ohne genaue Karte nicht lokalisieren kann, aber ohne genaue Lokalisation keine genaue Karte erstellen kann. Ziel der Lösung des SLAM Problems ist es, einem Roboter zu ermöglichen, sich in unbekanntem Gebiet zu orientieren.

Sowohl das Mapping, als auch die Lokalisation unterliegen einem Fehler, der sich aus der Ungenauigkeit der Sensoren ergibt. Diese Fehler addieren sich inkremtell, falls kein Filter verwendet wird. Der Roboter kennt die Umgebung nicht und muss erst eine Karte erstellen, um sich zu lokalisieren. Nach der Lokalisation wird die Karte mit neuen Sensorinformationen erweitert. Ohne eine geeignete Filterung würde sich die erstellte Karte mit jedem inkrementellen Schritt verschlechtern, da die Ungenauigkeiten der Karte zu Fehlern in der Lokalisation führen.

SLAM ist somit kein spezieller Algorithmus, sondern eine Problemstellung, die eine Methode verlangt, um die Fehler in der Lokalisation und der Kartenerstellung gleichermaßen zu minimieren.

1.1 Lösungsansatz FastSLAM

Der Ursprung des Namens FastSLAM [TBF05] kommt daher, dass FastSLAM ein schneller Algorithmus ist, der das SLAM Problem inkrementell zu lösen versucht. Die Besonderheit bei FastSLAM ist, dass Fehler, die bei der Kartenerstellung verursacht werden, durch die Erstellung mehrerer Karten minimiert werden. Eine falsch erweiterte Karte kann also in einem späteren Schritt durch eine andere, bessere Karte ersetzt werden, auch wenn diese anfangs unwahrscheinlicher ist. FastSlam ist eine Kombination aus Partikel- und Kalman-Filtern. Erst werden Features selektiert, die als Landmarken gespeichert werden und deren Positionen inklusive dessen Genauigkeit mit Extended Kalman-Filtern geschätzt werden. Die Roboterposition wird mittels Partikelfilter ermittelt. Jedes Partikel repräsentiert eine Position und eine Karte. Dadurch ist es möglich, mehrere Karten zu erstellen und über einen gewissen Zeitraum zu verfolgen.

1.2 Ziel dieser Studienarbeit

Ziel dieser Arbeit ist, den FastSLAM Algorithmus so zu implementieren, dass er auf dem Robbie System lauffähig ist. Grundlage der Arbeit bildet das Buch Propabilistic Robotics [TBF05], sowie das Buch FastSLAM [MT07] in denen der Algorithmus beschrieben ist. Da der Algorithmus eigentlich für ein autonomes Fahrzeug entwickelt worden ist, welches sich outdoor fortbewegt und hauptsächlich Bäume und Sträucher als Landmarken benutzt, muss ebenfalls ein eigener Ansatz zum Selektieren von Features entwickelt werden, der sich für die Lokalisation innerhalb von Gebäuden verwenden lässt.

Kapitel 2

Grundlagen für den FastSLAM Algorithmus

Notationstabelle

x_t: Position des Roboters zum Zeitpunkt t

$\hat{x}_t$: vorhergesagte Position des Roboters zum Zeitpunkt t

P_t: Rauschen der Roboterposition zum Zeitpunkt t

z_t: Messung eines Sensors zum Zeitpunkt t

R: Messrauschen

u_t: Kontrolleingaben (Bewegung des Roboters) zum Zeitpunkt t

Q: Systemrauschen (Bewegungsrauschen)

H_x: Jacobi Matrix, abgeleitet nach der Roboterosition

H_μ: Jacobi Matrix, abgeleitet nach der Landmarke

V: Jacobi Matrix, abgeleitet nach dem Fehler der Landmarke

μ : Position der Landmarke

Σ : Rauschen der Landmarke

$\hat{bel}(x_t)$: A priori Wahrscheinlichkeitsfunktion (Belief vor der Messung)

$bel(x_t)$: A posteriori Wahrscheinlichkeitsfunktion (Belief nach der Messung)

2.1 SLAM - ein Bayes Problem

SLAM kann, genauso wie die reine Lokalisation als Bayes Problem angesehen werden.

FOR all x_t DO	
	$\hat{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
	$bel(x_t) = \eta p(z_t \mid x_t) \hat{bel}(x_t)$

Bild 2.1: Bayes Filter Algorithmus

Das heißt, es kann mit einem Bayes Filter gelöst werden [TBF05]. Ein Bayes Filter besteht aus zwei Schritten, einem Vorhersage- und einem Updateschritt. Er errechnet die Wahrscheinlichkeit sich an der Position x_t zu befinden und erstellt dafür eine Wahrscheinlichkeitstabelle über alle möglichen Positionen.

Dafür wird erst die Wahrscheinlichkeit für die Vorhersage erstellt. Die Vorhersage wird auf Grundlage des alten Zustands (der Position) x_{t-1} und zusätzlicher Kontrolleingaben (Bewegung) u_t gemacht. Dies ergibt $\hat{bel}(x_t)$, also die a priori Wahrscheinlichkeit (Belief), sich nach der Kontrolleingabe an Position x_t zu befinden. Die Messung wird in diesem Schritt noch nicht berücksichtigt.

Dieses Belief $\hat{bel}(x_t)$ wird durch das Integral von zwei Wahrscheinlichksfunktionen errechnet - der Wahrscheinlichkeit sich im vorherigen Schritt an der Position x_{t-1} befunden zu haben (dem alten Belief $bel(x_{t-1})$) und der Wahrscheinlichkeit, sich durch die Bewegung u_t an der Stelle x_t zu befinden.

Der Updateschritt geschieht durch die Messung z_t. Hier wird die Wahrscheinlichkeit, die Position x_t durch die Messung z_t zu messen, mit der Wahrscheinlichkeit $\hat{bel}(x_t)$ aus dem ersten Schritt multipliziert. Zur Normalisierung muss noch mit η multipliziert werden.

Das Bayes Filter funktioniert nur bei sehr einfachen Problemen mit endlichen Zustandsmengen, da man nicht die Wahrscheinlichkeit von unendlich vielen Zuständen errechnen

kann. Ist die Zustandsmenge unendlich, so muss eine andere Methode verwendet werden, die nur die Wahrscheinlichkeit von wenigen - aber eher wahrscheinlichen Zuständen errechnet. Solche Filter sind z.B. Kalman- oder Partikelfilter.

2.2 Kalman-Filter

2.2.1 Funktionsweise Kalman-Filter

Ein Kalman-Filter ist ein stochastischer Zustandsschätzer für dynamische Systeme [May79]. Basis des Kalman-Filters ist die Schätzung eines Zustands x_t durch eine oder mehrere verrauschte Messungen z_t. Der Zustand x_t selber besitzt ebenfalls ein Rauschen als Kovarianzmatrix P, um Aussagen machen zu können, wie genau der geschätzte Zustand ist.

Es wird erst eine Messung $\hat{z}_t$vorhergesagt und diese dann mit der realen Messung z_t verglichen. Die daraus resultierende Differenz wird nun mit der Varianz R der Messung gewichtet um daraus eine neue Schätzung des Zustands zu erhalten.

Es basiert also auf 3 Schritten:

1. Vorhersage der Messung
2. Messen
3. Aktualisierung des Zustandes

2.2.2 Einfaches eindimensionales Kalman-Filter (KF)

Gegeben sei der alte Zustand x_{t-1} mit mit der Varianz P_{t-1}, eine Bewegung u_t mit der Varianz Q, und eine Messung z_t welche den Zustand mit der Varianz R misst. Die Varianzen repräsentiert das Rauschen der Sensoren, bzw. die geschätzte Ungenauigkeit des Kalman-Filters. [WB04]

Auch wenn es sich hier nicht um Matrizen handelt, werden beim eindimensionalen Beispiel für die Varianzen Großbuchstaben verwendet. In späteren Kapiteln werden die Varianzen durch Kovarianzmatrizen ersetzt.

Als Erstes wird der neue Zustand vorhergesagt

$$\hat{x}_t = x_{t-1} + u_t \tag{2.1}$$
$$\hat{P} = P_{t-1} + Q \tag{2.2}$$

Um die Differenz einer Vorhersage mit der realen Messung zu gewichten wird das Kalman Gain K benutzt. Es gewichtet, wie stark die Differenz Einfluss auf den neuen Zustand hat und wie stark sich das Rauschen des neu errechneten Zustands verkleinert. Je größer das Zustandsrauschen $\hat{P}$ und je kleiner das Messrauschen R ist, desto größer ist K. Dadurch, dass Varianzen immer positiv definiert sind, gilt $0 \leq K \leq 1$.

$$K = \hat{P}/(\hat{P} + R) \tag{2.3}$$
$$x_t = \hat{x}_t + K(z - \hat{x}_t) \tag{2.4}$$
$$P_t = (1 - K)\hat{P} \tag{2.5}$$

Beispiel: Ein Roboter ist ca. $3,3$m von einer Wand entfernt, mit einer Varianz von $0,15$m Genauigkeit. Er bewegt sich ca. $1,2$m von der Wand weg (Varianz ist $0,3$m). Man errechnet somit eine Entfernung von $4,5$m. Ein Ultraschall-Enfernungsmesser (mit einer Varianz von $0,1$ misst aber eine Entfernung von $4,83$m.

$x_{t-1} = 3,3$; $u_t = 1,2$; $P_{t-1} = 0,15$; $Q = 0,3$; $R = 0,1$; $z = 4,83$

$$\begin{aligned}
\hat{x}_t &= x_{t-1} + u_t = && 3,3 + 1,2 = 4,5 \\
\hat{P} &= P_{t-1} + Q = && 0,15 + 0,3 = 0,45 \\
K &= \hat{P}/(\hat{P} + R) = && 0,45/(0,45 + 0,1) = 0,45/0,55 = 0,8181 \\
x_t &= \hat{x}_t + K(z - \hat{x}_t) = && 4,5 + 0,8181(4,83 - 4,5) = 4,5 + 0,27 = 4,77 \\
P_t &= (1 - K)\hat{P} = && (1 - 0,8181)0,45 = 0,08181
\end{aligned}$$

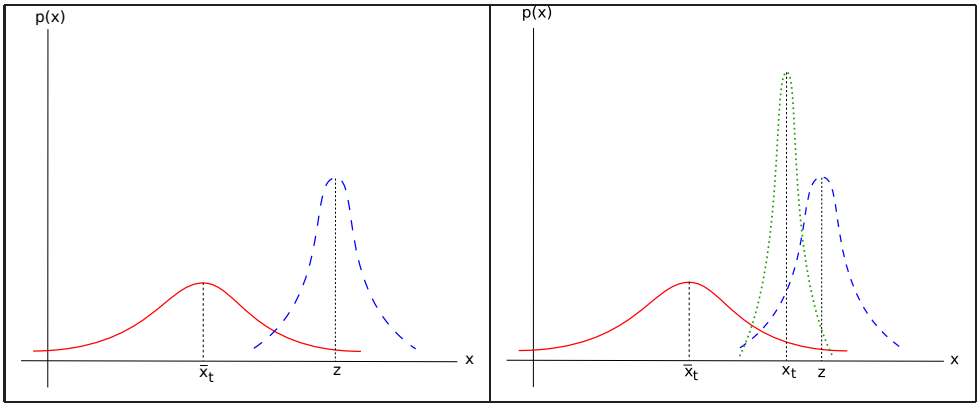

Bild 2.2: Vorhersage rot/durchgehend, Messung blau/gestrichelt, Kalman-Ergebnis grün/gepunktet

Die mittels KF errechnete Position x_t wäre also $4,77$m von der Wand entfernt, die Varianz dieses Ergebnisses wäre $0,08181$ ($\sigma_{x_t} = 0,28604$). Siehe Abbildung 2.2.

2.2.3 Repräsentation von Ungenauigkeiten durch Kovarianzmatrizen

Eine Kovarianzmatrix P kann die Abweichung von einem Vector (einer Position) darstellen. Die Diagonalmatrix in $P(x,x)$, $P(y,y)$ und $P(z,z)$ stellt die Varianz des Vectors dar während die restlichen Kovarianzen die Ausrichtung des Rauschens bestimmen. Die Kovarianzmatrix eines Vektors $a = (x,y,z)^T$ bildet sich folgendermaßen:

$$P = Cov(a) = \begin{pmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{pmatrix}$$

Kovarianzmatrizen besitzen also auf ihrer Diagonalen die Varianzen eines Vektors. Um eine Kovarianzmatrix P zu transformieren (z.B. Rotation oder Transformation von 2×2 auf 3×3 Matrix) wird sie folgendermaßen mit einer Transformationsmatrix A multipliziert

$$\hat{P} = APA^T \tag{2.6}$$

2.2.4 Mehrdimensionales Kalman Filter mit Matrizen

Kalman Filter eignen sich vor allem für mehrdimensionale Systeme [WB04]. Das Kalman Filter auf mehrdimensionale Systeme benutzt zur Darstellung des Rauschens Kovarianzmatrizen.

Daraus ergibt sich für die Vorhersage

$$\hat{x}_t = Ax_{t-1} + Bu \tag{2.7}$$
$$\hat{P} = AP_{t-1}A^T + Q \tag{2.8}$$

und für das Update

$$K = \hat{P}H^T(H\hat{P}H^T + R)^{-1} \tag{2.9}$$
$$x_t = \hat{x}_t + K(z - H\hat{x}_t) \tag{2.10}$$
$$P_t = (I - KH)\hat{P} \tag{2.11}$$

Die Matrix H stellt hier die Transformation vom Zustand zur Messung dar - also die Vorhersage der Messung. Die Dimension von Zustand x_t, Messung z_t und Kontrolle u kann also unterschiedlich sein. I ist hier die Einheitsmatrix mit der Dimension von K.

2.2.5 Extendet Kalman Filter (EKF)

Das Problem des Kalman-Filters ist, dass es sich nur auf lineare Systeme anwenden lässt. Viele Anwendungsfälle sind allerdings nicht linear. Fährt ein Fahrzeug beispielsweise eine Kreisbahn und hat einen winkelabhängigen Sensor, so ist die Bewegung nicht mehr mit einem Kalman-Filter zu schätzen.

Die Matrix H, die Vorhersage der Messung, wäre nicht mehr linear, da sie von der Ausrichtung des Fahrzeugs abhängig ist.

Beim EKF werden die Vorhersagen des Zustands und die Vorhersagen der Messung durch nichtlineare Funktionen $g()$ und $h()$ berechnet. Die Transformationen des Rauschens wird

durch linearisierte Matrizen errechnet. Die Linearisierung geschieht per Taylor Approximation, also einer partiellen Ableitung der nichtlinearen Funktionen. Die daraus errechneten linearisierten Matrizen sind somit Jacobimatrizen [WB04].

Vorhersage:

$$\hat{x}_t = g(x_{t-1}, u) \tag{2.12}$$
$$\hat{P} = GP_{t-1}G^T + Q \tag{2.13}$$

Update:

$$K = \hat{P}H_t^T(H_t\hat{P}H_t^T + R)^{-1} \tag{2.14}$$
$$x_t = \hat{x}_t + K(z - h(\hat{x}_t)) \tag{2.15}$$
$$P_t = (I - KH_t)\hat{P} \tag{2.16}$$

2.2.6 Beispielergebnisse einer Simulation

Grafik 2.3 zeigt den Plot eines zweidimensionalen Zustands, der mit einem EKF berechnet wurde. Der Zustand ergibt sich aus der Entfernung zum Objekt und der eigenen Geschwindigkeit, also die Geschwindigkeit eines Fahrzeugs, welches von einem Startpunkt aus in eine Richtung fährt. Gemessen wird die Entfernung zu einem großen leicht erkennbaren Objekt (z.B. einem Haus), indem eine montierte Kamera die Größe des bekannten Objektes misst. Odometrie Daten gibt es nicht, die Entfernung und Geschwindigkeit wird also alleine durch die Kamera bestimmt. Die bisherige Geschwindigkeit, die im vorigen Schritt aus der zurückgelegten Distanz errechnet wurde, wird als aktuelle Geschwindigkeit genommen.

Das Ergebnis zeigt die ungefilterte und mit einem Kalman-Filter geschätzte Entfernung zu dem Objekt.

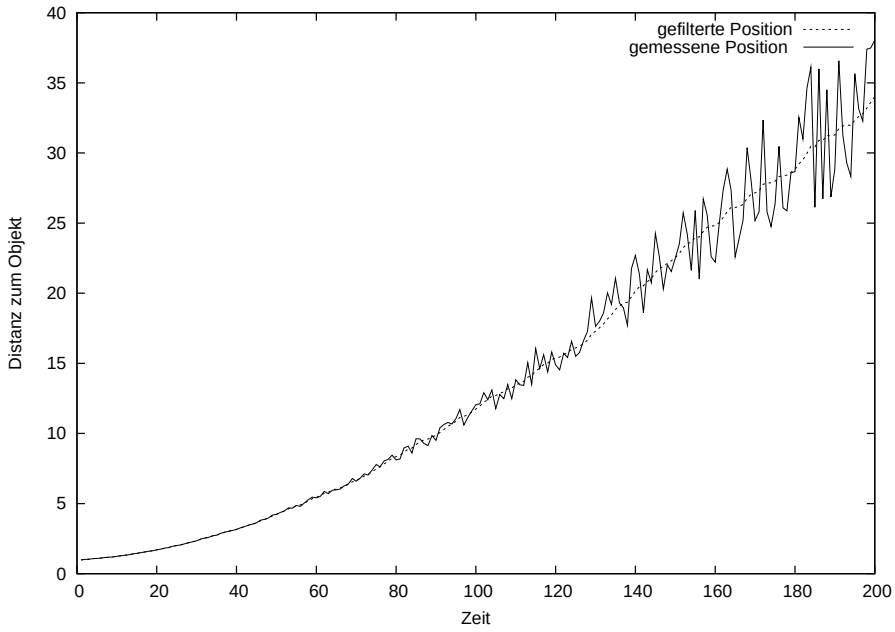

Bild 2.3: Simuliertes Ergebnis eines EKF.

2.3 Partikelfilter

2.3.1 Funktionsweise eines Partikelfilters

Partikelfilter [IB98] bauen ebenfalls auf ein ähnliches Prinzip auf wie Kalman-Filter. Sie sind ebenfalls Zustandsschätzer, allerdings können mit Hilfe von Partikelfiltern mehrere mögliche Zustände betrachtet werden. Es gibt also eine Vielzahl von Zuständen, die nun Partikel genannt werden.

Partikelfilter bestehen aus vier Schritten:

1. Fortschreiten (Bewegung)
2. Addieren eines Fehlers auf die Zustände
3. Messen und gewichten der Partikel
4. Resampeln

Der erste Schritt ist das Fortschreiten der Partikel. Dies entspricht der Vorhersage des Zustands vom Kalman-Filter in Gleichung 2.12, welche für jedes Partikel durchgeführt wird. Der zweite Schritt ist das Addieren eines zufällig gewählten Fehlers auf jeden Zustand. Jeder Zustand wird somit künstlich mit unterschiedlichen Fehlern verrauscht. Dies wird durchgeführt, um ein mögliches Systemrauschen, welches durch eine ungenaues Fortschreiten verursacht wird zu korrigieren. Schritt 1 und 2 werden oft zusammengefasst, um eine verrauschte Bewegung besser zu simulieren.

Im dritten Schritt wird verglichen, wie gut die Partikel in eine reale Messung hinein passen. Passen sie gut in die Messung, bekommen sie ein hohes Gewicht. Passen sie weniger gut, bekommen sie ein niedriges Gewicht.

Der vierte Schritt ist das Resampling. Hier geschieht eine Selektion von Partikeln. Partikel mit hohem Gewicht werden beibehalten, Partikel mit geringem Gewicht, welche nicht gut in die Messung passen werden verworfen. Die Auswahl geschieht zufällig unter Berücksichtigung der Gewichtung der Partikel.

Es werden also zufällig eine Vielzahl von unterschiedlichen Zuständen erzeugt. Die Messung des Sensors entscheidet, welche Zustände weiter erhalten bleiben, und welche Verworfen werden.

2.3.2 Lokalisation durch Partikelfilter

Ein sehr einfaches Beispiel für einen Partikelfilter ist die Lokalisation eines Roboters in einer bereits bekannten Umgebung [FTBD01]. Zuerst werden zufällig Partikel in einer

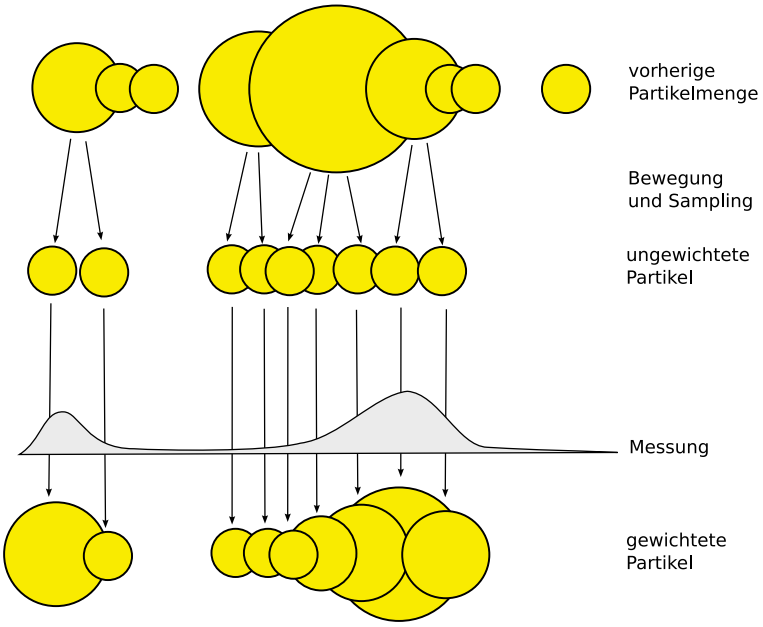

Bild 2.4: Funktionsweise des Partikelfilters. Resampling, Sampling, Measure [IB98]

bereits vorhandenen Karte gestreut. Danach wird verglichen, ob die Messungen des Roboters mit der Karte übereinstimmen. Je mehr diese übereinstimmen, desto höher werden die Partikel gewichtet, und desto mehr Partikel werden beim nächsten Schritt von diesem Partikel aus gestreut, während Partikel, bei denen die Messung nicht in die Karte passt verworfen werden.

So kann man bereits nach wenigen Schritten und trotz sehr verrauschten Messungen eine relativ genaue Positionsbestimmung eines Roboters machen.

2.3.3 Partikelfilter SLAM (Stephan Wirth)

Das Partikelfilter SLAM ist eine einfache aber recht effiziente Methode zu Kartenerstellung. Er ergibt sich aus einer Lokalisation mittels Partikelfilter und der Erstellung einer einzigen Karte durch eine Occupancy Map. Es wird also durch den ersten Scan eine initiale Karte erstellt, in der sich lokalisiert wird und über dessen wahrscheinlichste Position

resample - Neuverteilung der Partikel nach Gewicht
FOR each particle
drift and sample - Fortschreiten und Verrauschen der Position
measure - Vergleichen des Scans mit der Occupancy Map
compute weight - Gewichten der Partikel
normalize and sort - Normalisierung der Gewichte und Sortierung nach Gewicht
extend map - Karte mit dem Laserscan an Position des besten Partikels erweitern

Bild 2.5: Partikelefilter SLAM (Stephan Wirth)

die Karte erweitert wird.

Grundlage für Berechnung der Occupancy Map bildet ein Laser-Range-Scanner, der horizontale Scans auf einer bestimmten Höhe macht, welche dann in eine Karte eingetragen werden. Das Grid der Occupancy Map muss dem Rauschen des montierten Laser-Scanners angepasst werden. Im Idealfall entspricht die Auflösung des Grids der maximalen Standardabweichung des Laser-Range-Scanners. Wird das Grid größer gewählt, ist die Eintragung des Scans in der Karte und somit auch die Lokalisation ungenauer. Wird das Grid kleiner gewählt, so werden eingetrage Punkte oft nicht mehr wiedererkannt, weil die Scanns nicht mehr genau den eingetragenen Punkt in der Karte treffen. Die Occupancy Map dient somit als Subsampling Filter für eingehende Laser Messungen und als Grundlage für den gesamten Mapping Vorgang.

Kapitel 3

Funktionsweise FastSLAM

3.1 FastSLAM 1.0

Das Problem vom Partikelfilter SLAM ist, dass eine falsche Position direkt einen Fehler in der Karte erzeugt, der nicht mehr korrigiert werden kann. Die Lösung von FastSLAM [TBF05] ist, dass nicht nur über die unterschiedliche Positionen gestreut wird, sondern auch über mehrere Karten. Somit können Fehler in der Karte, die durch eine falsche Position verursacht werden abgefangen werden, da auch Karten von weniger wahrscheinlichen Positionen erstellt werden, die sich erst zu einem späteren Zeitpunkt als besser herausstellen.

Ein Partikel besteht aus einer Position $x_t = (x, y, \theta)^T$ und einer Karte. Eine Karte besteht aus mehreren Landmarken, wobei jede eine Position μ und eine Kovarianz Matrix Σ besitzt. Die Kovarianz Matrix Σ repräsentiert die Ungenauigkeit der Landmarke.

Position und Rauschen der Landmarken werden durch ein Kalman-Filter errechnet.

Im ersten Iterationsschritt wird die Karte erstellt. Die Landmarken werden mittels Kalman-Filter initialisiert und gespeichert. Jede Landmarke bekommt ein eigenes Kalman-Filter. Danach werden folgende Schritte durchgeführt. Schritt 1 bis 4 werden für jedes Partikel einzeln berechnet.

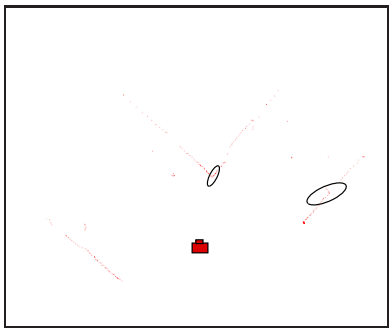

Bild 3.1: Das initiale Speichern von Landmarken.

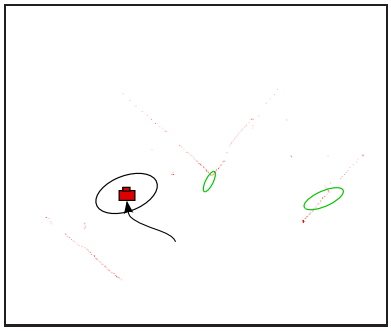

Bild 3.2: Die Bewegung des Roboters und die Vorhersage der Roboterposition und dessen Rauschen.

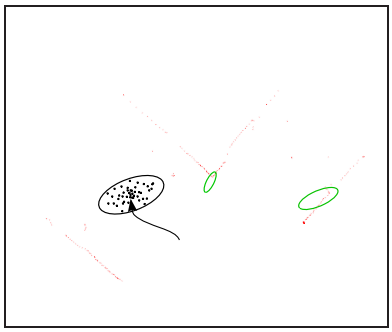

Bild 3.3: Streuung von Partikeln im Bereich des geschätzten Rauschens.

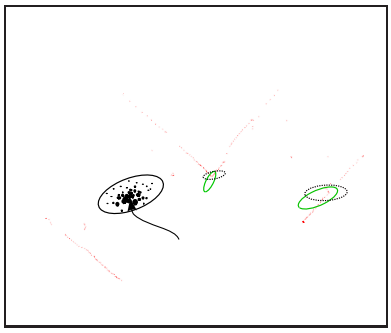

Bild 3.4: Gewichtung der Partikel. Je besser die Messung auf die gespeicherten Landmarken passt, desto höher das Gewicht (und desto größer der Radius in der Abbildung).

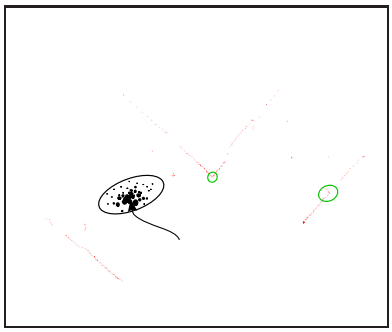

Bild 3.5: Aktualisieren der Landmaren und dessen Rauschen mittels Kalman-Filter.

1. **Fortschreiten:** Zuerst wird die Roboterosition x_t und dessen Rauschen P_t der aktuellen Bewegung u_t angepasst. (Abbildung 3.2)

2. **Streuung:** Im nächsten Schritt werden Partikel im Bereich des Rauschens gestreut. Die Streuung entspricht einer multivariaten Normalverteilung mit dem Mittelpunkt x_t und der Kovarianz P_t. (Abbildung 3.3)

3. **Wiedererkennen:** Die gespeicherten Landmarken werden der wahrscheinlichste Messung $z_{t,i}$ zugeordnet. (Abbildung 3.4)

4. **Aktualisieren der Karte:** Die Landmerken werden mit den Messungen über einen Kalman-Filter aktualisiert, neue Messungen werden als Landmarken gespeichert. (Abbildung 3.5)

5. **Resampling:** Die Partikel werden in Rücksicht auf die errechnete Wahrscheinlichkeit (Gewichtung) neu verteilt. Partikel mit hohem Gewicht erzeugen viele neue Partikel. Partikel mit geringem Gewicht erzeugen keine oder nur wenige Partikel. Jedes Partikel besitzt seine eigene Karte.

3.1.1 Positionsvorhersage

Zur Vorhersage der Roboterposition Position werden die errechneten Odometriedaten $u = (\Delta x, \Delta y, \Delta\theta)^T$ einfach auf die alte Position des Partikels $x_{t-1} = (x, y, \theta)^T$ aufaddiert. $x_t = x_{t-1} + u$.

Zu beachten ist, dass die Odometriedaten für jedes Partikel einzeln berechnet werden müssen, da die $\Delta x, \Delta y$ Koordinaten abhängig von der Ausrichtung des Roboters sind. Die detaillierte Gleichung ist in Kapitel 3.4.1.

Die Vorhersage des Rauschens die sich aus der Bewegung ergibt kann unterschiedlich vorhergesagt werden. Propabilistic robotics [TBF05] geht z.B. für FastSLAM 2.0 hier von einem konstanten Rauschen aus, unabhängig von der Bewegung u_t. Es ist allerdings auch möglich die Bewegungsvorhersage nach dem Fehler ϵ abzuleiten, um Informationen zu erhalten, wie stark sich der Fehler durch die Bewegung verändert. Mit dieser Ableitung V_x

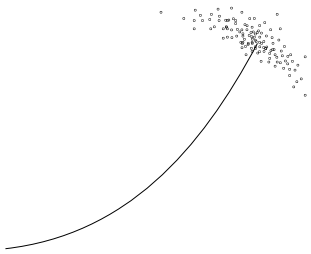

Bild 3.6: Sampeln von möglichen Positionen mittels verrauschter Bewegung [IB98].

kann man dann eine genauere Vorhersage über das Rauschen erstellen, da somit bei einer geringen Bewegung wenig und bei einer starken Bewegung viel gestreut wird. [TBF05].

3.1.2 Sampeln einer Position

Das Rauschen welches durch die Kovarianzmatrix P repräsentiert wird, entspricht einer multivariaten Normalverteilung. Eine Stichprobe daraus lässt sich mittels Cholesky-Zerlegung erzeugen. Es wird die Matrix S mit $P = SS^T$ erzeugt, mit einem gaußverteiltem Zufallsvektor r multipliziert und auf die bisherige vorhergesagte Position aufaddiert.

$$S = chol(P) \tag{3.1}$$

$$x_t = \hat{x}_t + rS \tag{3.2}$$

Zusammengefasste Vorhersage und gleichzeitiges Sampeln der Position

Alternativ zum Erstellen einer Kovarianzmatrix und dem Sampling in Schritt 1 und 2 kann dies bei FastSLAM 1.0 zusammengefasst werden, indem die Bewegung bereits in der Vorhersagefunktion künstlich verrauscht wird. Dies kann das Sampling verbessern, indem die Funktion besser dem realen Bewegungsrauschen angepasst werden kann, als dies eine Kovarianz Matrix darstellen könnte.

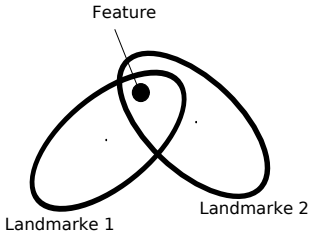

Bild 3.7: Unsicherheit bei der Messung von Landmarken. Die Ellipsen stellen Kovarianzmatrizen dar, welche die Ungenauigkeit der Landmarken repräsentieren.

3.1.3 Unbekannte Datenassoziationen

In einigen Fällen ist es möglich Landmarken mittels eindeutigem Diskriptor zu identifizieren. Dies ist besonders hilfreich, um falsche Datenassoziationen und somit eine falsche Lokalisation zu verhindern.

In vielen Fällen allerdings lassen sich Landmarken nicht eindeutig einander zuordnen [MT03]. In Abbildung 3.7 kann man beispielsweise nicht eindeutig bestimmen, ob das Feature zu der rechten, der linken oder einer neuen noch nicht gemessenen Landmarke zugeordnet werden müsste. Es ist hilfreich, Landmarken mit möglichst großem Abstand zu erzeugen, um solche falschen Datenassoziationen zu minimieren; verhindern kann man sie allerdings kaum. Um die Features eindeutig ihren Landmarken zuzuordnen ist es nötig, die Wahrscheinlichkeit der Assoziation zu errechnen (siehe Kapitel 3.6).

Des weiteren müssen Doppelassoziationen verhindert werden. Für das Errechnen der Wahrscheinlichkeit werden die Vorhersagen der Kalman-Filter durchgeführt um die nötigen Informationen über das aktuelle Rauschen der Landmarken neu zu errechnen. Ist die Wahrscheinlichkeit bekannt, so wird das wahrscheinlichste Feature einer Landmarke zugeordnet. Hierbei muss vorher ein Schwellwert festgelegt werden, der die minimale Wahrscheinlichkeit p_{min} für ein Feature festlegt. Wird dieser Schwellwert unterschritten, so wird das Feature als neue Landmarke eingetragen.

Hieraus ergibt sich wiederum ein Performanceproblem, denn eigentlich müssten für jedes Partikel und jede Landmarke die Wahrscheinlichkeit für alle Messungen per Kalman Filter errechnet werden. Um diesen Aufwand zu minimieren kann man eine Vorauswahl treffen,

wobei nur die Wahrscheinlichkeit vom besten (geometrisch naheliegensten) Feature errechnet wird [MT07].

3.1.4 Speichern einer Messung

Sind die Datenassoziationen bekannt, ist auch bekannt, welche Features bereits als Landmarke gespeichert wurden, und welche neu sind. Bereits beobachtete Features werden benutzt, um mittels Kalman Filter die Position der Landmarke zu verfeinern. Hierfür wird der Update-Schritt der Landmarke benutzt (siehe Kapitel 3.3.3). Je öfter eine Landmarke beobachtet wurde, desto kleiner ist deren Rauschen.

3.1.5 Resampling - Positionsbestimmung mittels Partikelfilter

Um nun das Gewicht eines Partikels zu bestimmen, werden die Wahrscheinlichkeiten aller wiedergefundenen Landmarken miteinander multipliziert. Neu gefundene Landmarken bekommen die Wahrscheinlichkeit vom Mindestschwellwert zum Finden einer Landmarke p_{min}. Um eine möglichst gleichwertige Verteilung zu erhalten werden die Gewichte der Partikel normalisiert. Die Summe der Gewichte aller Partikel wird auf 1 gesetzt.

Nun können neue Partikel k erzeugt werden. Ein Partikel erzeugt in etwa $p(k) \cdot count(particles)$ neue Partikel. $p(k)$ entspricht dem Gewicht des Partikels.

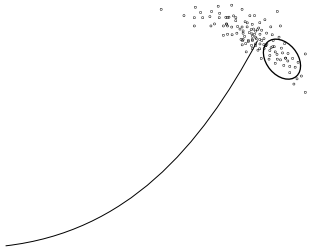

Bild 3.8: Partikel mit hohem Gewicht sind innerhalb der Ellipse. Die Mitte der Ellipse wäre der ideale Ausgangspunkt zum Streuen der Partikel. Partikel außerhalb der Ellipse wurden umsonst gestreut. Die Ellipse lässt sich durch ein Kalman-Filter berechnen. [MT07]

3.2 FastSLAM 2.0

Der Unterschied zwischen FastSLAM 1.0 und FastSLAM 2.0 [TBF05] ist, dass bei FastSLAM 2.0 eine Vorauswahl getroffen wird, um welchen Mittelpunkt und mit welcher Varianz wie weit durch das Partikelfilter gestreut wird. Diese Vorauswahl wird durch ein weiteres Kalman-Filter errechnet. Während FastSLAM 1.0 in Abbildung 3.8 den kompletten Bereich sampelt, würde FastSLAM 2.0 nur im Bereich der Ellipse sampeln.

FastSLAM 2.0 errechnet die Roboterposition also durch ein Kalman-Filter und durch ein Partikelfilter.

Im ersten Iterationsschritt werden immer noch lediglich die Landmarken durch ein Kalman-Filter initialisiert. Schritt 1 bis 6 werden für jedes Partikel einzeln berechnet.

1. **Fortschreiten:** Position x_t und das Positionsrauschen P_t werden der aktuellen Bewegung u_t angepasst.
2. **Wiedererkennen:** Die Landmarken werden der Messung z_t zugeordnet, wie in Schritt 3 bei FastSLAM 1.0.
3. **Positionsschätzung (Roboter):** Nach erfolgreicher Zuordnung der Landmarke zu einer Messung wird die Roboterposition x_t und dessen Rauschen durch ein Kalman-Filter errechnet.
4. **Streuung:** Die Position kann nun um den neuen Mittelpunkt $\hat{x}_t$ und der Kovarianz $\hat{P}_t$ gesampelt werden.
5. **Wiedererkennen:** Da sich die Roboterposition x_t und die daraus errechnete Wahrscheinlichkeit gefundener Landmarken verändert hat muss Schritt 2 wiederholt werden. Den Landmarken wird unter Umständen eine neue Messung zugeordnet.
6. **Aktualisieren der Karte:** Analog zu Schritt 4 in FastSLAM 1.0. Die Landmarken werden durch das Kalman-Filter aktualisiert.
7. **Resampling:** Analog zu Schritt 5 in FastSLAM 1.0. Resampling - Partikel mit hohem Gewicht erzeugen viele neue Partikel, Partikel mit geringem Gewicht werden unter Umständen gelöscht.

3.2.1 Unterschiede zu FastSLAM 1.0

In FastSLAM 2.0 wird die Vorhersage der Roboterposition und dessen Rauschen durch ein Kalman Filter errechnet. Die Messung wird also bereits vor der Streuung berücksichtigt. Somit ergibt sich ein besserer Mittelpunkt und ein kleinerer Streuradius für das Partikelfilter.

Durch die Vorauswahl in FastSLAM 2.0 kann die Anzahl der verwendeten Partikel reduziert werden. Dadurch, dass die Messung bereits bei der Vorhersage berücksichtigt wird, werden weniger falsche Karten errechnet. Allerdings erfordert dies auch eine erhöhte Sorgfältigkeit bei der Wahl der Parameter und ein weiteres Aufstellen von Jacobi Matrizen für das Kalman-Filter die nach der Roboterposition abgeleitet werden.

Schritt 1 und 4 können nicht wie in FastSLAM 1.0 durch eine zusammengefasste Sampelfunktion ersetzt werden, die gleichzeitig die Bewegung und die Streuung durchführt.

3.3 Positionsbestimmung von Landmarken mittels Extended Kalman-Filter

Landmarken werden von einem Sensor gemessen, der einem Rauschen unterliegt. Um dieses Rauschen zu filtern und gleichzeitig Aussagen machen zu können, wie genau die bisherige Schätzung ist, wird die Position von einem Kalman-Filter ermittelt. Dabei kann berücksichtigt werden, wie die Landmarken und deren Rauschen sich aus unterschiedlichen Zuständen verändern. Hierfür wird die Jacobi Matrix H_μ berechnet, indem die Vorhersage Funktion des Zustands der Landmarke $h(x_t, \mu)$ nach μ_t abgeleitet wird. H_μ ist somit eine Linearisierung der nichtlinearen Funktion $h(x_t, \mu)$ an der Stelle μ.

Soll auch noch die Entfernung zur Landmarke beim Errechnen des Rauschens berücksichtigt werden, so errechnet man zusätzlich die Jacobi Matrix V indem man $h(x_t, \mu)$ nach dem Fehler ϵ ableitet, der sich bei unterschiedlichen Entfernungen zur Landmarke unterschiedlich verhält. Verändert sich dieses Rauschen nicht, oder ist es nicht modellierbar, kann man die Matrix V entweder auf die Einheitsmatrix setzen oder ganz aus der Gleichung entfernen.

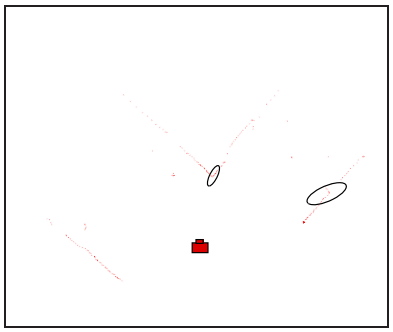

Bild 3.9: Das initiale Speichern von Landmarken.

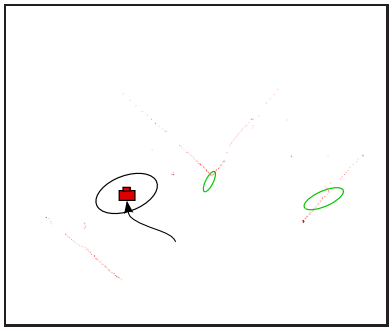

Bild 3.10: Die Bewegung des Roboters und die Vorhersage der Landmarke.

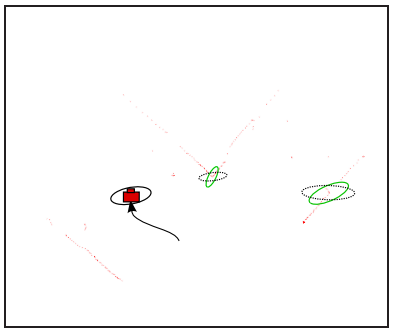

Bild 3.11: Die Roboterposition und dessen Varianz wird durch die Kalman-Filter aktualisiert.

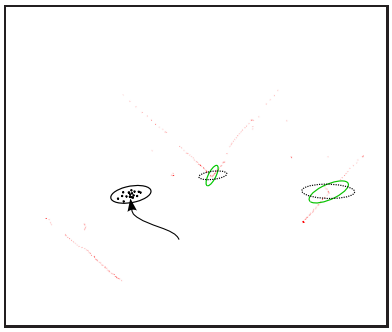

Bild 3.12: Ein Partikelfilter streut über mögliche Positionen.

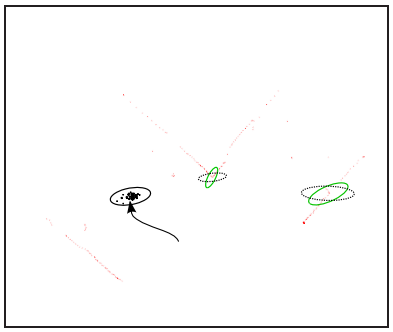

Bild 3.13: Die Partikel werden gewichtet, je nachdem wie weit die Features von den gespeicherten Landmarken abweichen.

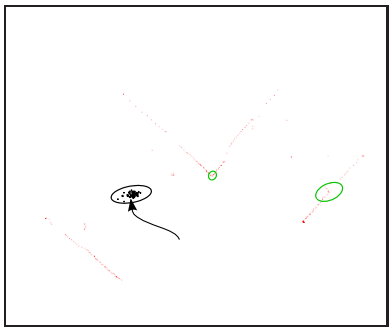

Bild 3.14: Ist die Position für das Partikel bestimmt, werden die Landmarken aktualisiert.

3.3.1 Initialisieren einer Landmarke

Die Landmarke wird nur einmal initialisiert, und zwar bei der ersten Speicherung. Beim Rechnen in kartesischen Koordinaten wird durch die Funktion $^{-1}()$ die Position der Landmarke vom Roboterkoordinatensystem ins Weltkoordinatensystem transformiert. Die hier gespeicherte Position ist noch recht ungenau, was durch die Kovarianzmatrix Σ_t repräsentiert wird. Durch den Kalman-Filter wird diese Matrix bei jeder weiteren Messung aktualisiert, die Genauigkeit der Position wird also durch jede Messung erhöht.

$$H_\mu = \frac{\partial h(x_t, \mu_t)}{\partial \mu} \tag{3.3}$$

$$V = \frac{\partial h(x_t, \mu_t)}{\partial \epsilon} \tag{3.4}$$

$$\mu_t = h^{-1}(x_t, \mu_t) \tag{3.5}$$

$$\Sigma_t = H_\mu^{-1} V R V^T (H_\mu^{-1})^T \tag{3.6}$$

3.3.2 Vorhersagen der Messung einer Landmarke

Die Vorhersage ist die inverse Funktion zur Initialisierung. Während bei der Initialisierung - beim Rechnen in kartesischen Koordinaten - die Position der Landmarke vom Roboterkoordinatensystem ins Weltkoordinatensystem transformiert wird, wird bei der Vorhersage die Landmarke vom Weltkoordinatensystem ins Roboterkoordinatensystem transformiert. Die Matrix H_μ entspricht der partiellen Ableitung dieser Funktion $h()$.

$$H_\mu = \frac{\partial h(x_t, \mu_t)}{\partial \mu} \tag{3.7}$$

$$V = \frac{\partial h(x_t, \mu_t)}{\partial \epsilon} \tag{3.8}$$

$$\hat{z}_t = h(x_t, \mu_t) \tag{3.9}$$

$$\hat{\Sigma}_t = H_\mu \Sigma_{t-1} H_\mu^T \tag{3.10}$$

$$Q_{t,\mu} = \hat{\Sigma}_t + V R V^T \tag{3.11}$$

3.3.3 Update einer Landmarke

Jeder Update Schritt verkleinert die Varianzen der Kovarianzmatrix Σ_t. Die Position wird also - bei richtig zugeordnetem Feature - immer genauer.

$$K = \Sigma_{t-1} H_\mu^T Q_{t,\mu}^{-1} \quad (3.12)$$

$$\mu_t = \mu_{t-1} + K(z_t - \hat{z}_t) \quad (3.13)$$

$$\Sigma_t = (I - K H_\mu)\Sigma_{t-1} \quad (3.14)$$

3.4 Positionsbestimmung des Roboters mittels Extended Kalman-Filter

Die Positionsbestimmung des Roboters bei FastSLAM 2.0 funktioniert ein wenig anders als die der Landmarken. Zum einen wird zu jedem Zeitpunkt t von einer unverrauschten Startposition x_{t-1} ausgegangen; das Systemrauschen wird also nur durch die Bewegung u_t verursacht und nicht zusätzlich mit dem Rauschen aus dem vorherigen Schritt addiert. Zum anderen wird das Rauschen der Landmarken, nach denen sich positioniert wird direkt aus dem Kalman-Filter der Landmarken gewonnen. Für jede erkannte Landmarke wird ein Aktualisierungsschritt des Kalman-Filters für die Positionsbestimmung durchgeführt.

3.4.1 Vorhersage der Roboterposition

Zur Vorhersage der Position wird erst die v, die seit dem letzten Schritt zurückgelegte Strecke errechnet, welche um den Winkel θ rotiert wird und auf die alte Position aufaddiert wird. Die Vorhersage des Rauschens $\hat{P}_t$ wird der Odometrie angepasst. Die Ungenauigkeit der Odometriedaten wird durch die folgenden Parameter α bestimmt.

u:Position nach Odometriedaten v:Translation (velocity)
r: Ausrichtungsänderung

α_v: Geschwindigkeits- bzw. Translationsfehler
α_w: Rotationsfehler
α_{vw}: Translationsfehler bei Rotation
α_{wv}: Rotationsfehler bei Translation
α_p: zusätzlicher unabhängiger Positionsfehler.
α_r: zusätzlicher unabhängiger Rotationsfehler.

$$v = \sqrt{\Delta u_{t,x}^2 + \Delta u_{t,y}^2} \tag{3.15}$$

$$\theta = x_{t-1,\theta} + atan2(u_{t,y}, u_{t,x}) - u_{t-1,\theta} \tag{3.16}$$

$$M = \begin{pmatrix} \alpha_v v^2 + \alpha_{vw} \Delta u_{t,\theta}^2 \\ \alpha_w \Delta u_{t,\theta}^2 + \alpha_{wv} v^2 \end{pmatrix} \tag{3.17}$$

$$W = \begin{pmatrix} cos(\theta), sin(\theta)v \\ sin(\theta), cos(\theta)v \\ 0, 1 \end{pmatrix} \tag{3.18}$$

$$R_x = \begin{pmatrix} \alpha_p, 0, 0 \\ 0, \alpha_p, 0 \\ 0, 0, \alpha_r \end{pmatrix} \tag{3.19}$$

$$\hat{x}_t = \hat{x}_{t-1} + \begin{pmatrix} cos(\theta)v + x_{t-1,x} \\ sin(\theta)v + x_{t-1,y} \\ \Delta u_{t,\theta} x_{t-1,\theta} \end{pmatrix} \tag{3.20}$$

$$\hat{P}_t = WMW^T + R_x \tag{3.21}$$

Um das Bewegungsrauschen zu modellieren, wird die Bewegungsfunktion in Gleichung reflab:predictPose nach dem Fehler α abgeleitet, wodurch sich W ergibt.

Durch FastSLAM 2.0 wird das Positionsrauschen des Roboters durch jede Messung verkleinert. α_p und α_r simulieren zusätzliches Systemrauschen (das z.B. durch Rutschen entsteht). Diese Werte sollte man nicht auf 0 setzen, da dadurch auch die Kovarianzmatrix P_t bei Stillstand des Roboters Werte bei 0 annehmen würde und es somit bei der Positionskorrektur und beim Sampeln der Position zu Singularitäten kommen würde. Jede gemessene

Landmarke verkleinert bei FastSLAM 2.0 das Rauschen $\hat{P}_t$, so dass auch bei hohem Positionsrauschen (durch die Odometrie) weniger durch den Partikelfilter gestreut wird, als bei FastSLAM 1.0.

3.4.2 Aktualisieren der Roboterposition durch ein Kalman-Filter

Das Aktualisieren der Roboterposition wird für jede Messung einzeln durchgeführt. Bei jedem Schritt verkleinert sich die Kovarianzmatrix des Positionsrauschens $\hat{P}_t$, und dadurch verkleinert sich auch das Kalman Gain K_x. Je kleiner K_x wird, desto weniger werden die aktuellen Messungen berücksichtigt.

$$H_x = \frac{\partial h(\hat{x}_t, \mu_t)}{\partial x} \tag{3.22}$$

$$\hat{P}_t = (H_x^T Q_{t,\mu}^{-1} H_x + \hat{P}_t^{-1})^{-1} \tag{3.23}$$

$$K_x = \hat{P}_t H_x^T Q_{t,\mu}^{-1} \tag{3.24}$$

$$\hat{x}_t = \hat{x}_t + K_x(z_t - \hat{z}_{t,\hat{\mu}}) \tag{3.25}$$

3.5 Zusammenarbeit der Kalman-Filter bei FastSLAM 2.0

Um die Position des Roboters zu bestimmen ist es erforderlich, alle Vorhersage-Schritte der Kalman-Filter für alle wiedergefundenen Landmarken zu errechnen um die nötigen Informationen über das Rauschen der Landmarken zu erhalten. Um den Update-Schritt der Landmarken zu errechnen ist es ebenfalls nötig, die endgültige Roboterposition zu wissen, um die Landmarken an der richtigen Position zu speichern. Es handelt sich also eigentlich um ein einziges abgeschlossenes System um die Position zu errechnen, welches zusätzlich auch die bisherigen Informationen über die Landmarken speichert.

3.6 Bestimmung der Wahrscheinlichkeit für die richtige Zuordnung einer Landmarke

Die Wahrscheinlichkeit w eine Landmarke einer Messung richtig zugeordnet zu haben, kann errechnet werden, wenn die Kovarianzmatrix des Rauschens L bekannt ist. Die dafür erforderlichen Jacobi- und Kovarianzmatrizen können aus den vorherigen Schritten des Kalman-Filter entnommen werden.

$$L = H_x P_t H_x^T + H_\mu \Sigma_{t-1} H_\mu^T + V R V^T \tag{3.26}$$

$$w = det(2\pi L)^{-\frac{1}{2}} exp\{-\frac{1}{2}(z_t - \hat{z}_{t,\mu})^T L_t^{-1}(z_t - \hat{z}_{t,\mu})\} \tag{3.27}$$

Ist das Rauschen der Position P_t nicht bekannt, bzw. für alle Partikel gleich, wie in FastSLAM 1.0, so kann man diesen Wert vernachlässigen und $H_x P_t H_x^T$ auf 0 setzen.

Kapitel 4

Implementierung

4.1 Selektieren von Landmarken

FastSLAM setzt eine Selektion von Landmarken voraus. Die Landmarken sollten möglichst weit auseinander liegen und in allen Scans vorkommen. Sind in einem Scan keine Landmarken vorhanden, so ist es für den Roboter nicht möglich sich zu lokalisieren. Da in Gebäuden sehr unterschiedliche Arten von Landmarken existieren, muss eine Auswahl von Möglichen Landmarken getroffen werden.

4.1.1 Ecken

Ecken kann man mit mehreren Methoden extrahieren. Eine recht schnelle und einfach Methode ist, aus den mittels Laserscanner detektierten Punkten zwei Punktepaare aus je 3 Punkten zu selektieren, aus denen man zwei Linien bildet und deren Schnittwinkel misst. Ist der Winkel groß genug, kann man den mittleren Punkt als Ecke definieren. Siehe Abbildung 4.1.

Um gute Werte zu erhalten sollte man die Laserdaten vorher per Filter (Mittelwertfilter) glätten und danach nur das lokale Maximum der Winkel aus einem begrenzten Bereich wählen.

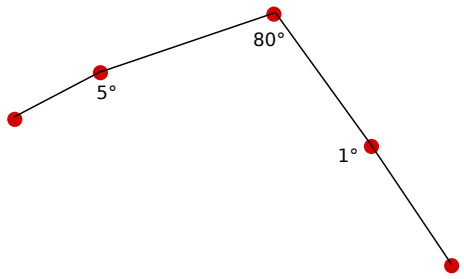

Bild 4.1: einfaches Extrahieren von Ecken

4.1.2 Linien

Zum Detektieren von Linien ist das Incremental Line Fitting Algorithmus ein recht schnelles und brauchbares Verfahren. Punkte werden mittels Least Square Algorithmus zu einer Linie so lange zusammengefügt, bis ein gewisser Schwellwert erreicht ist. Danach wird die nächste Linie begonnen. Andere Verfahren werden im Paper von Nguyen vorgestellt [NMTS05].

Aus Performancegründen können die Daten mittels Clustering oder Mittelwertfilter gefiltert werden. Beim Clustering wird die Anzahl der Punkte reduziert, indem Punkte, die in einem bestimmten Radius zueinander liegen, zusammengefügt werden. Somit werden große Häufungen im Nahbereich des Laserscanners minimiert.

4.1.3 Linien (Wände) als Landmarken benutzen

Leider lassen sich Linien nicht direkt als Landmarken verwenden. Fährt der Roboter z.B. durch einen geraden Gang ohne Türen und Fenster mit zur Fahrtrichtung parallelen Wänden, so werden nur zwei Linien rechts und links vom Roboter erkannt. Fährt man einen Meter nach vorne, haben die Linien immer noch die selber Länge und aus Robotersicht immer noch die gleiche Position da die Ecken der erfassten Linien begrenzt sind, durch die Reichweite des Laser-Range-Scanners. Die erwartete Position wäre einen Meter vorraus gewesen. Ein Kalman-Filter würde somit nun annehmen, da sich die Linien nicht bewegt haben, dass sich der Roboter ebenfalls nicht bewegt hat, obwohl dieser einen Meter nach

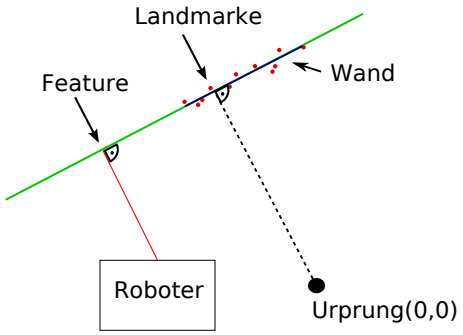

Bild 4.2: Extrahieren von Linien

vorne gefahren ist.

Um dieses Problem zu umgehen, nimmt man die Normalengleichung der Geraden, die durch die Linie aufgespannt wird, also das Lot vom Roboter auf die Gerade der Linie. Gespeichert wird die Normale zum Ursprung des Weltkoordinatensystems. Die vorhergesagte Normale würde nun der Gemessenen entsprechen. Leider gehen dadurch einige Positionsinformationen der Line verloren, da Geraden unendlich lang sind.

4.1.4 Ecken durch Linien extrahieren

Das Extrahieren von Ecken aus Linien ist recht trivial. Man vergleicht alle Linien miteinander und deklariert den Schnittpunkt der Linie als Ecke, falls die Linien in einem bestimmten Winkel aufeinander liegen. Da die Linien durch den Laser-Range-Scanner bereits vin links nach rechts sortiert sind, reicht es aus, nur benachbarte Linien miteinander zu vergleichen. Vergleicht man auch nicht benachbarte Linien, so muss man beachten, dass eventuell ähnliche Linien die gleiche Ecke mehrmals darstellen können, da die Linien unendlich lang sind.

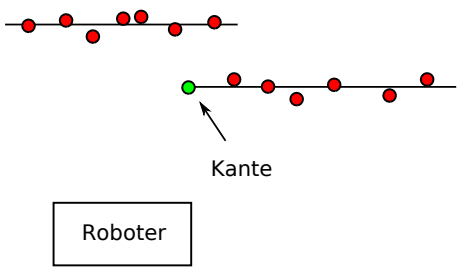

Bild 4.3: Extrahieren von Kanten

4.1.5 Kanten mittels Gradienten feststellen

Eine Kante wird hier als Stelle definiert, an der die Ableitung der vom Laser-Range-Scanner gemessenen Entfernungen (Punkte) ein lokales Maximum bildet.

Es wird die Entfernung vom Laser-Range-Scanner zu einem gemessenen Punkt n mit dem nächsten gemessenen Punkt $n+1$ verglichen. Die Differenz dieser Entfernungen entspricht der ersten Ableitung. Ist der Betrag dieser Differenz größer als ein vorher festgelegter Wert, so wählt man den Punkt, welcher näher am Roboter ist und nimmt ihn als Feature.

Da bei diesem Verfahren oft viele Landmarken nah beieinander liegen, sollte man nur lokale Maxima in einem bestimmten vorher festgelegtem Radius wählen, um später falsche Datenassotiationen zu vermeiden.

4.1.6 Vor- und Nachteile

Jede der vorgestellten Landmarken hat seine Vor- und Nachteile. Während Ecken nur selten vorkommen, z.B. gar nicht bei geraden Gängen ohne Türen, fehlt bei Linien die genaue Position. Es kann also nicht vorhergesagt werden, ob eine vorher gespeicherte Linie bei der nächsten Messung in Reichweite des Laserscanners ist, und somit sind Loops nur schlecht zu schließen. Ebenfalls sind Linien bei unruhigen nicht komplett geraden Wänden oder Strukturputz sehr schlecht zu detektieren und können somit schnell zu Ausrichtungsfehlern führen. Kanten unterliegen sehr hohen Schwankungen. In einigen Räumen können gar keine gemessen werden, und in anderen in denen viele Tisch- und Stuhlbeine auf Scanner

FOR each Partikel do
Fortschritt und Rauschen schätzen(1)
FOR each Form von Landmarken do
Assotiationen von Features und Landmarken finden (2)
Roboterposition und Systemrauschen durch Messung aktualisieren(3)
Verrauschen der Roboterposition(4)
FOR each Form von Landmarken do
Assotioationen von Features und Landmarken finden (2)
Landmarken speichern (6)
Resample - Hoch gewichtete Partikel vervielfältigen (7)

Bild 4.4: FastSLAM 2.0 mit mehreren Sensoren. Die Zahlen in Klammern verweisen auf die Schritte in Kapitel 3.2.

Höhe stehen, sind Kanten so dicht aneinander, dass die lokalen Maxima manchmal die eine, manchmal die andere Kante als Landmarke repräsentieren.

Somit muss das Verfahren auf die Umgebung angepasst werden.

4.2 Erweitern des Algorithmus auf mehrere verschiedene Landmarken

Da die selektierten Landmarken unterschiedliche Vor- und Nachteile aufweisen, bietet es sich an, mehrere Methoden zu verwenden, damit der Algorithmus in unterschiedlichen Umgebungen funktioniert.

Die einzelnen Schritte bleiben gleich und müssen nur an einigen Stellen öfters aufgerufen werden. Das folgende Beispiel in Abbildung 4.4 zeigt die Verwendung von mehreren Methoden zur Landmarken Erkennung, bzw. die Verwendung von mehreren Sensoren in FastSLAM 2.0.

4.3 Errechnen des Gewichts eines Partikels

Die Wahrscheinlichkeit eines Partikels ergibt sich aus dem Produkt aller Wahrscheinlichkeiten aller Landmarken, welches noch normiert wird, damit die Summe der Gewicht aller Partikel 1 ergeben.

$$\sum^{particles} \prod^{landmarks} p(\mu_i) = 1$$

Ein Problem hierbei ist, dass wenn die Kovarianzmatrizen zu groß werden die Wahrscheinlichkeiten sehr kleine Werte von bis zu 10^{-7} annehmen können und dann das Produkt der Wahrscheinlichkeiten schnell unter MINFLOAT fällt. Dies ist besonders der Fall bei kleinen Maßeinheiten, da hier die Kovarianzmatrizen und die Differenz der Vorhersage und der Messung numerisch sehr groß werden.

Abhilfe schafft ein Umstellen auf eine größere Maßeinheit. Hierfür ist es auch nicht nötig, das gesamte System umzustellen, sondern es reicht, wenn man das Errechnen der Wahrscheinlichkeit auf eine andere Maßeinheit umstellt. Die Differenz zwischen Vorhersage und Messung muss also mit einem entsprechenden Skalierungsfaktor und die Kovarianzmatrix mit dem Quadrat des Skalierungsfaktors multipliziert werden.

4.4 Vorhersagefunktionen

Die Vorhersage der Landmarke $h(\mu, x_t)$ ist hier in allen Fällen eine Transformation vom Weltkoordinaten- ins Roboterkoordinatensystem. Auf Polarkoordinaten wurde hier verzichtet.

Die Inverse der Vorhersage $h^{-1}(z, x_t)$ ist somit die Transformation vom Roboter ins Weltkoordinatensystem. Die Position des Sensors wird hier in allen Fällen als Mitte des Roboters angenommen. Da dies selten der Fall ist, sollten die Koordinaten bereits vor dem Selektieren von Features so verschoben werden, dass immer von der Mitte des Roboters als virtuelle Position auszugehen ist.

4.4.1 Ecken / Kartesische Punkte durch den Laserscanner

Die Vorhersage eines kartesischen Punktes ist erst eine Translation um die (negative) Position des Roboters, und dann eine Rotation um die (negative) Ausrichtung des Roboters.

$z = h(\mu, x_t)$:

$$s = \mu - x_t \tag{4.1}$$

$$z = \begin{pmatrix} s_x cos(-s_\theta) - (s_y sin(-s_\theta)) \\ s_x sin(-s_\theta) + (s_y cos(-s_\theta)) \\ s_\theta \end{pmatrix} \tag{4.2}$$

Die Initialisierungsfunktion (inverse der Vorhersage) entspricht erst einer Rotation um die Roboterausrichtung und dann einer Translation um die Roboterposition.

$\mu = h^{-1}(z, x_t)$:

$$\mu = \begin{pmatrix} z_x cos(x_{t,\theta}) - (z_y sin(x_{t,\theta})) + x_{t,x} \\ z_x sin(x_{t,\theta}) + (z_y cos(x_{t,\theta})) + x_{t,y} \\ z_\theta \end{pmatrix} \tag{4.3}$$

4.4.2 Linien

Bei der Vorhersage der Linien wird aus der Normalen auf die Gerade zum Ursprung (gespeicherte Landmarke) die Normale zur Gerade auf den Roboter gebildet(Messung).

$z = h(\mu, x_t)$:

$$\theta_\mu = atan2(\mu_y, \mu_x) \tag{4.4}$$

$$\theta_z = \theta_\mu - x_{t,\theta} \tag{4.5}$$

$$\theta_{x_t} = \theta_\mu - atan2(x_{t,y}, x_{t,x}) \tag{4.6}$$

$$d_z = \parallel \mu \parallel - \parallel x_t \parallel cos(\theta_{x_t}) \tag{4.7}$$

$$absRange(d_z, \theta_z) \tag{4.8}$$

$$z = \begin{pmatrix} d_z cos(\theta_z) \\ d_z sin(\theta_z) \\ 0 \end{pmatrix} \tag{4.9}$$

$\mu = h^{-1}(z, x_t)$:

$$\theta_z = atan2(z_y, z_x) \tag{4.10}$$
$$\theta_\mu = \theta_z + x_{t,\theta} \tag{4.11}$$
$$\theta_{x_t} = \theta_\mu - atan2(x_{t,y}, x_{t,x}) \tag{4.12}$$
$$d_\mu = \parallel z \parallel - \parallel x_t \parallel cos(\theta_{x_t}) \tag{4.13}$$
$$absRange(d_{\mu,z}, \theta_z) \tag{4.14}$$
$$\mu = \begin{pmatrix} d_\mu cos(\theta_\mu) \\ d_\mu sin(\theta_\mu) \\ 0 \end{pmatrix} \tag{4.15}$$

$absRange(d, \theta)$:

$$if(d < 0) \tag{4.16}$$
$$d = -d \tag{4.17}$$
$$\theta = \theta - \pi \tag{4.18}$$
$$endif \tag{4.19}$$

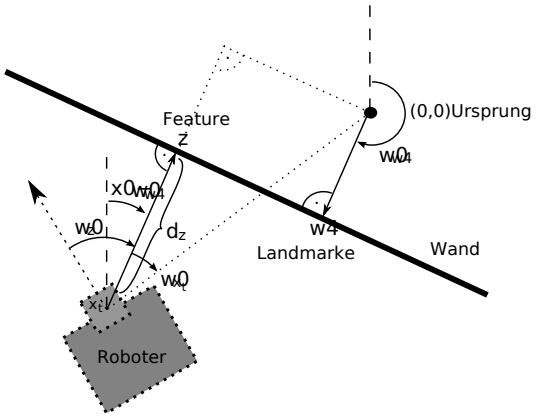

Bild 4.5: Vorhersage der Linien

4.5 Verwendete Jacobi Matrizen

Die Jacobi Matrizen sind die partiellen Ableitungen der Vorhersagefunktionen. Eine Matrix J_f, welche die Funktion f nach x ableitet hat folgende Form:

$$J_f = \frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \tag{4.20}$$

4.5.1 Ecken

Die Ableitung der Vorhersage nach der Position der Landmarke ergibt eine Rotationsmatrix die von der Ausrichtung des Roboters abhängig ist.
$H_\mu = \frac{\partial h(\mu, x_t)}{\partial \mu}$:

$$H_\mu = \begin{pmatrix} cos(-x_\theta) & -sin(-x_\theta) & 0 \\ sin(-x_\theta) & cos(-x_\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.21}$$

Die Ableitung nach der Roboterposition wird benötigt, um die Roboterposition durch eine Veränderung von Messung und Vorhersage der Landmarke zu korrigieren.
$H_{x_t} = \frac{\partial h(\mu, x_t)}{\partial x_t}$:

$$s = \mu - x_t \tag{4.22}$$

$$H_{x_t} = \begin{pmatrix} -cos(-x_\theta) & sin(-x_\theta) & s_x(sin(-\theta)) + s_y cos(-\theta) \\ -sin(-x_\theta) & -cos(-x_\theta) & -s_x(cos(-\theta)) - s_y cos(-\theta) \\ 0 & 0 & 1 \end{pmatrix} \tag{4.23}$$

Die Ableitung der Vorhersage nach dem Fehler der Landmarke ergibt eine Rotationsmatrix, die von der Ausrichtung und Entfernung der Landmarke abhängig ist.

$V_\mu = \frac{\partial h(\mu, x_t)}{\partial \epsilon}$:

$$s = \mu - x_t \tag{4.24}$$

$$d_\epsilon = \sqrt{\| s \|} \tag{4.25}$$

$$\theta = atan2(s_y, s_x) \tag{4.26}$$

$$V_\mu = \begin{pmatrix} d_\epsilon cos(\theta) & -d_\epsilon sin(\theta) & 0 \\ d_\epsilon sin(\theta) & d_\epsilon cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.27}$$

4.5.2 Linien

Die Ableitung nach der Position der Landmarke ist hier nicht von der Roboterposition abähngig.

Die Jacobi-Matrix H_μ ist hier nicht die mathematisch korrekte Ableitung von $h()$. Die mathematisch korrekte Ableitung wäre abhängig von der Postion x_t, was aber heißen würde, dass sich das Rauschen einer Landmarke vergrößern würde, je weiter man vom Startpunkt weg ist. Dies stimmt aber nicht, das Rauschen wird nur von der Ausrichtung der Landmarke selber beeinflußt. Die Ableitung entspricht also einer Rotation um die Richtung der Landmarke, vom Startpunkt aus gesehen.

$H_\mu = \frac{\partial h(\mu, x_t)}{\partial \mu}$:

$$\theta_\mu = atan2(\mu_y, \mu_x) \tag{4.28}$$

$$H_\mu = \begin{pmatrix} cos(\theta_\mu) & -sin(\theta_\mu) & 0 \\ sin(\theta_\mu) & cos(\theta_\mu) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.29}$$

Bei der Ableitung nach der Roboterposition H_{x_t} wird einfach die selbe Ableitung wie bei der Berechnung einer Ecke verwendet. Dies ist zwar mathematisch wiederum nicht korrekt, die Ergebnisse zeigen jedoch, dass es eine gute Approximation ist. Die mathematisch korrekte Herleitung hatte ähnliche unerwartete Ergebnisse geliefert, wie die Ableitung nach der Landmarkenposition. Die neu errechnete Position war nicht brauchbar.

Die Verwendung von der Ableitung aus der Berechnung der Ecke dagegen liefert dagegen absolut korrekte Ergebnisse, falls es sich um einen Translationsfehler handelt. Der Abstand zu einer Ecke, ist genauso zu korrigieren, wie der Abstand zu einer Wand. Bei Rotationsfehlern, bzw. falls die Messung der Wand einen Fehler im Winkel der Normalen hat, so sind die Ergebnisse nicht mehr korrekt. Die Korrekturen der Update-Funktion des Kalman-Filters führen in diesem Fall dazu, dass nicht nur die Ausrichtung des Roboters, sondern auch dessen Position in (x, y) Richtung verändert wird. Dies lässt sich unterdrücken, indem die Kovarianzmatrix des Messrauschens für Linien angepasst wird. Die Varianz in Linienrichtung muss künstlich in Richtung der Normalen etwa verdoppelt, in Wandrichtung mindestens verzehnfacht werden. Somit ist die verwendete Jacobi-Matrix für die Vorhersage der Roboterposition nicht ideal. Wichtig ist das Verwenden von Linien aber vor allem für das Partikelfilter, welches die Partikel aussellektiert, bei denen die Messung der Linien von den gespeicherten Linien abweicht.

$H_{x_t} = \frac{\partial h(\mu, x_t)}{\partial x_t}$:

$$s = \mu - x_t \tag{4.30}$$

$$\theta = atan2(s_y, s_x) \tag{4.31}$$

$$H_{x_t} = \begin{pmatrix} -cos(-x_\theta) & sin(-x_\theta) & s_x(sin(-\theta)) + s_y cos(-\theta) \\ -sin(-x_\theta) & -cos(-x_\theta) & -s_x(cos(-\theta)) - s_y cos(-\theta) \\ 0 & 0 & 1 \end{pmatrix} \tag{4.32}$$

$V_\mu = \frac{\partial h(\mu, x_t)}{\partial \epsilon}$:

$$V_\mu = I \tag{4.33}$$

4.6 Vereinfachtes Klassendiagramm der Implementierung

Abbildung 4.6 zeigt ein vereinfachtes Klassendiagramm der Implementierung. Die $main$ Funktion, von der aus die Klassen aufgerufen werden, enthält drei unterschiedliche Objekte folgender Klassen:

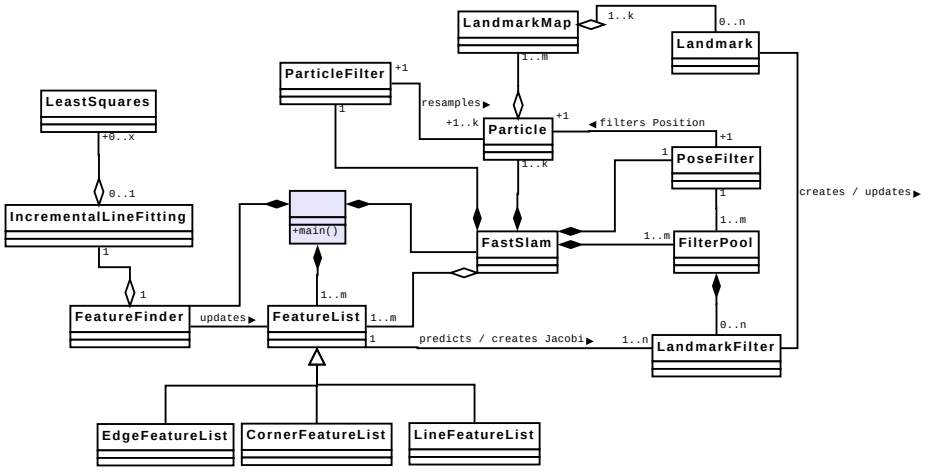

Bild 4.6: Vereinfachtes Klassendiagramm der Implementierung des FastSLAM 2.0 Algorithmus

1. Einen FeatureFinder,
2. eine Klasse FastSlam
3. und mindestens eine Klasse vom Typ FeatureList.

Der FeatureFinder selektiert die Features aus dem Scan. Momentan sind Linien, Kanten und zwei verschiedene Methoden zur Detektierung von Ecken vorhanden. Es kann auch ein eigener FeatureFinder implementiert werden.

Dieser FeatureFinder füllt nun mit den detektierten Features die FeatureList mit $featureList.updateMeasure(z)$. In der FeatureList sind zusätzlich alle featurespezifischen Daten vorhanden, etwa die Kovarianzmatrix der Messwerte sowie die Ableitungen H_x,H_f, V und die Vorhersagefunktionen $h(\mu, x_t)$ und $h^{-1}(z, x_t)$ für die Kalman-Filter. Jede unterschiedlich detektierte Art von Features braucht also eine eigene Klasse, die von FeatureList abgeleitet ist.

Die Klasse FastSlam bekommt nun direkt nach der Initialisierung Referenzen auf die Featurelisten mit $fastSlam.addFeatureList(FeatureList*)$. Dieser Schritt darf nur direkt nach der Initialisierung durchgeführt werden, da das Initialisieren der Feature-Listen sämt-

liche gespeicherten Karten zurücksetzt.

Die Odometriedaten werden mit $fastSlam.updateOdometry(u)$ übergeben. Zum Starten des Filters wird $fastSlam.runFilter()$ durchgeführt, wodurch die Schritte der Kalman- und Partikelfilter durchgeführt werden.

Jeder Schritt von FastSlam erfordert also mindestens 4 Funktionsaufrufe:

$featureFinder.updateScan(z);$

$featureList.updateMeasure(featureFinder.getCorners());$

$fastSlam.updateOdometry(u);$

$fastSlam.runFilter();$

Die Idee bei diesen eher umständlichen Aufrufen ist die, dass beliebig viele Sensoren mit eingebunden werden können, die dann eventuell zur Positionsberechnung beitragen. Es können so also auch Messwerte übersprungen werden. Allerdings ist keine Interpolierung der Messwerte implementiert, die verwendeten Messwerte müssen also alle zum selben Zeitpunkt aufgenommen oder vorher interpoliert werden.

4.7 C++ Bibliothek tvmet - Tiny Vector Matrix library using Expression Templates

Durch das Robbie Projekt war bereits die Programmiersprache C++ und das Betriebssystem Linux vorgegeben. Da SLAM recht zeitaufwändig ist und eine Erhöhung der Partikelzahl nicht nur zu besseren Ergebnissen, sondern auch zu einer Verlangsamung des Algorithmus führt, war eine sorgfältige Auswahl einer mathematischen Bibliotheken in C++ nötig. Die Auswahl fiel auf tvmet, da diese Bibliothek einer sehr gute Performance bei kleinen Matrizen zeigte. Bei zwei- bis dreidimensionalen Matrizen zeigten sich bei der Matrixmultiplikation teilweise Performancevorteile um den Faktor zehn im Vergleich zu lapack++. Die dadurch entstehenden Einschränkungen, wie die Angabe der Dimensionalität bereits zu Kompilierzeit oder das Fehlen von Lösungen für Eigenwertprobleme sind für diesen Anwendungsfall und diese Implementierung zu vernachlässigen.

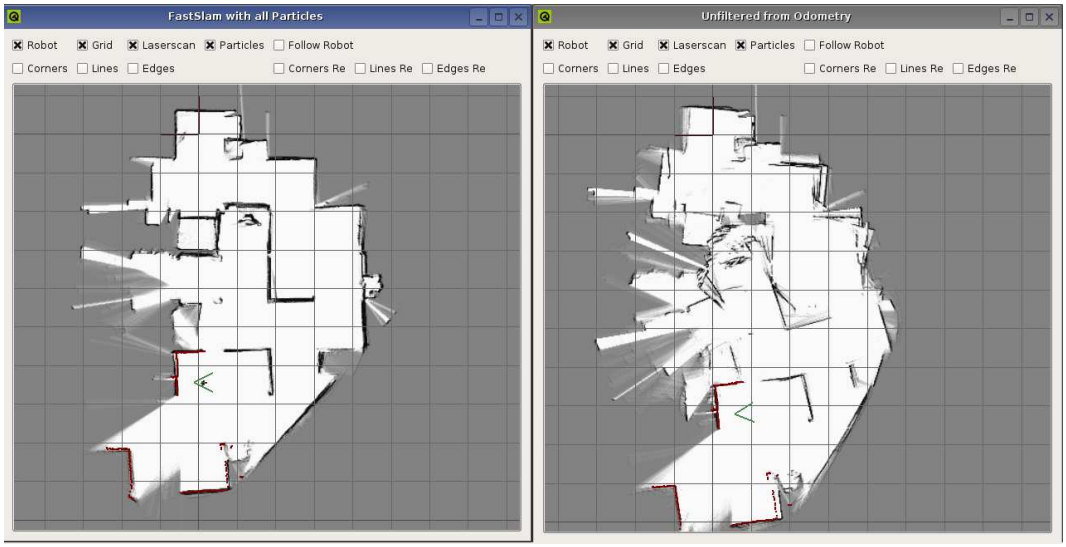

Bild 4.7: RoboCup German Open 2006. Mission 5. Links FastSLAM mit 500 Partikeln, rechts reine Odometrie.

4.8 Ergebnisse

Abbildung 4.7 zeigt FastSLAM (500 Partikel) mit den Daten, die während den RoboCup German Open 2007 in Hannover aufgenommen wurden. Links daneben ist eine Karte, die nur aufgrund der Odometriedaten erstellt wurde.

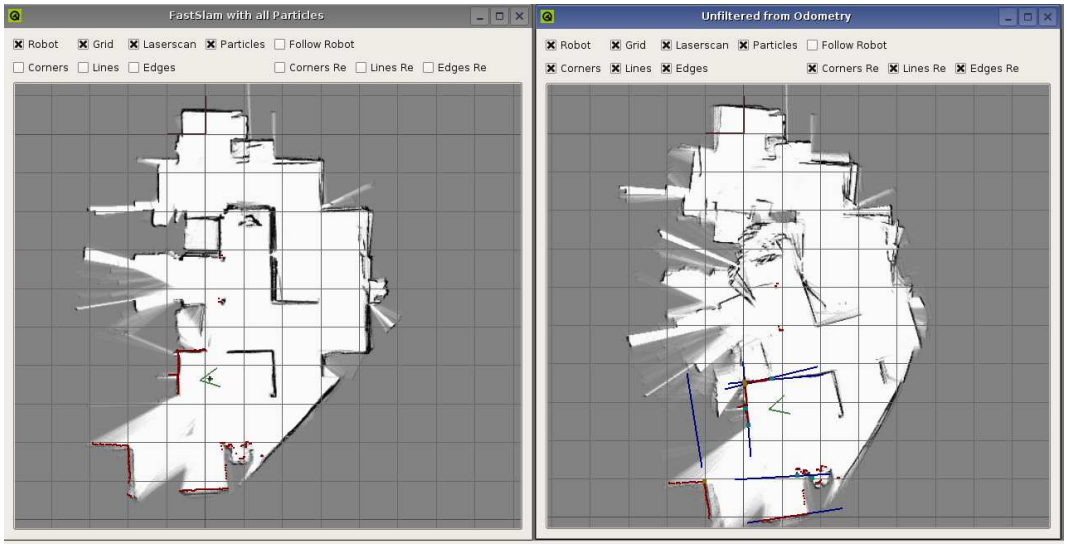

Bild 4.8: RoboCup German Open 2006 Mission 5. Links FastSLAM mit 100 Partikeln, rechts reine Odometrie.

Abbildung 4.8 zeigt den selben Datensatz mit 100 Partikeln. Die Karte sieht fast identisch aus. Lediglich einige Wände sind bereits doppelt eingezeichnet. Die ist allerdings ein gutes Anzeichen, dass die Positionsschätzung nicht ganz korrekt war. Da FastSLAM zu einem großen Teil auf einem Zufallsgenerator aufbaut, ist hier die Wahrscheinlichkeit groß, dass bei nur 100 Partikeln die Positionsschätzung nicht immer stabil läuft.

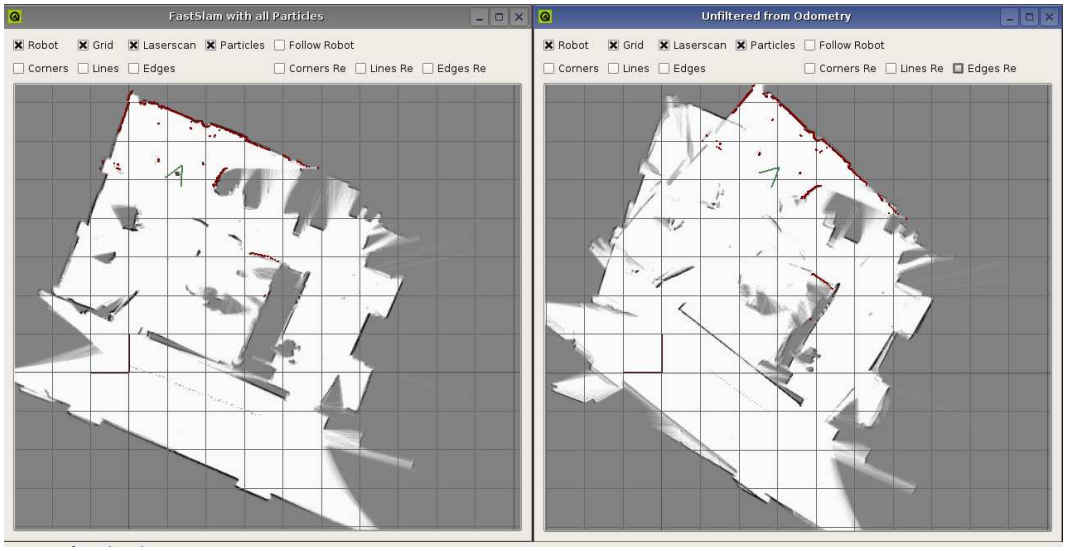

Bild 4.9: Loop Closure mit FastSLAM im Robbie Labor Koblenz. Links FastSLAM mit 500 Partikeln, rechts reine Odometrie.

Abbildung 4.9 zeigt einen Datensatz aus dem Robbie Labor. Hier wurde ein kleiner Loop Closure versucht. Der Roboter sollte also im Kreis fahren, und die Karte korrigieren, falls er eine Position erreicht, an der er schon mal war. Auffällig war hier, dass der Loop bei kleineren Laser-Scanner Intervallen von 100ms der Loop schlechter geschlossen wurde und mehr doppelte Wände verursacht hat als beispielsweise bei 500ms. Das Resampling hat also auch sehr viele Karten verworfen, obwohl diese im Nachhinein besser gewesen wären. Hier wäre es für Loop Closing von Vorteil, wenn das Resampling seltener durchgeführt würde, was dazu geführt hätte, dass die Karten eher erhalten bleiben. Die Anzahl der Partikel dagegen hat das Loop Closing nicht beeinflußt.

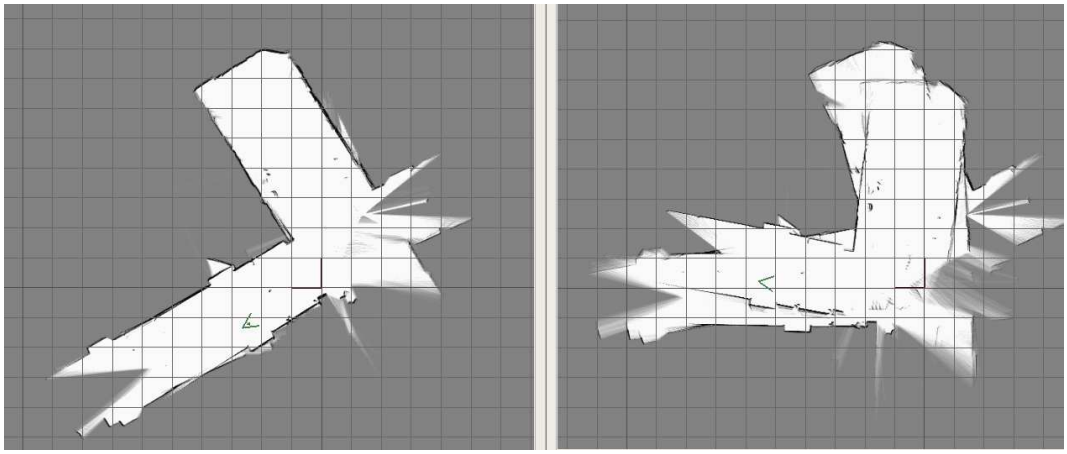

Bild 4.10: FastSLAM 2.0 mit nur einem Partikel (ohne Partikelfilter). Links FastSLAM 2.0 mit 1 Partikel, rechts reine Odometrie.

Abbildung 4.9 zeigt FastSLAM 2.0 mit nur einem Partikel. Das Partikelfilter wurde hier deaktiviert, damit kein zusätzliches Rauschen addiert wird. Die Position wird also nur über das Kalman-Filter geschätzt. Die Karte hat nur minimale Fehler, trotz schlechter Odometrie Daten. Möglich ist dies durch gut messbare Landmarken. Die Umgebung hat viele Ecken, nahezu gerade Wände und in geraden Gängen Kanten, die sich aus Tischbeinen ergeben.

4.9 Zusammenfassung

In dieser Studienarbeit wurde ein Algorithmus vorgestellt, um sich mit einem Roboter in unbekanntem Gebiet zu lokalisieren und gleichzeitig eine Karte von der Umgebung zu erstellen. Die Lokalisation des Roboters geschieht auf 2D Ebene und errechnet die $(x, y, \theta)^T$ Position des Roboters zu jedem Zeitpunt t inkrementell. Der Algorithmus baut auf dem FastSLAM 2.0 Algorithmus [TBF05] auf und wurde abgeändert, um eine möglichst genaue Lokalisation in Gebäuden zu ermöglichen. Hierfür wurden mehrere verschieden Arten von möglichen Landmarken untersucht, verglichen und kombiniert.

Schwerpunkt dieser Studienarbeit war das Einarbeiten in das Extended Kalman-Filter und die Selektion von Landmarken, die für den Einsatz in Gebäuden geeignet sind.

Für das Verständnis des Kalman-Filters wurden mehrere Prototypen in Octave erstellt, etwa ein lineares eindimensionales Kalman-Filtern und ein Nachbau eines 2D Slam Algorithmus in Octave.

Zum Selektieren von Features wurden anfangs nur Ecken verwendet. Da diese Ergebnisse allerdings nicht sehr stabil waren und bei Umgebungen mit wenig oder gar keinen Ecken falsche Positionen geschätzt wurden, mussten weitere Alternativen gesucht werden. Die mit diesen Landmarken gefilterten Karten waren zwar besser als ungefilterte, aber immer noch sehr ungenau. Die Ausrichtung des Roboters wurde oft falsch gefiltert, wodurch sich die Position so verschlechtert hat, dass Landmarken nicht mehr wiedergefunden wurden. Eine Umformung der Modellierung von kartesischen in Polarkoordinaten erbrachte bereits bessere Ergebnisse, da dadurch mitmodelliert wurde, dass sich das Messrauschen auf größere Entfernung ebenfalls vergrößert. Allerdings war dadurch das Erstellen von Jacobi-Matrizen zum Schätzen der Roboterposition in FastSLAM 2.0 deutlich komplizierter und schlechter nachvollziehbar, wodurch sich unerwartete Fehler bildeten. Somit wurden kartesische beibehalten und zusätzlich die in [WB04] vorgestellte Jacobimatrix V eingeführt, die eine Modellierung des Messfehlers zulässt.

Das Benutzen von Wänden als Landmarken brachte ebenfalls keine guten Ergebnisse. Hierfür wurde der Incremental-Line-Fitting implementiert [NMTS05] und auf die Punkte angewendet, die mit dem Laser-Range-Scanners gemessen wurden. Die Modellierung des Kalman-Filters stellte sich hier als problematisch heraus. Die erstellten Ableitungen der

Vorhersage Funktion waren zwar mathematisch korrekt, aber die Kovarianzmatrizen entsprachen nicht der Erwartung. Nach dem Modell müsste sich das Rauschen vergrößern, je weitere die gemessenen Landmarken vom Ursprung entfernt sind, was aber nicht der Realität entspricht. Das Rauschen ist nur von der Ausrichtung und Entfernung zum Roboter abhängig. Somit wurde die Roboterposition durch das Kalman-Filter nicht korrigiert. Eine approximierte Linearisierung der Vorhersagefunktion, die nicht mehr mathematisch korrekt war, brachte bessere Ergebnisse, konnte aber die Position nicht ausreichend korrigieren um dadurch korrekte Karten zu erstellen. Ebenso war von Nachteil, dass nicht vorhergesagt werden konnte, ob eine Landmarke im Bereich des Laser-Range-Scanners liegt, wodurch Loop-Closures nicht möglich waren.

Durch die Kombination von verschiedenen Landmarken ließen sich schließlich stabile Ergebnisse erzielen. Dies hatte allerdings zur Folge, dass die Anzahl einstellbarer Paramter sehr groß wurde. Neben vier Parametern, die die den Fehler in der Roboterbewegung modellieren gibt es für jede Art von Features mindestens eine Kovarianzmatrix mit einem initialen Grundrauschen, welches in der Entfernung größer wird und mindestens 3 weitere Parameter die zum selektieren des Features nötig sind (wie etwa Mindestlänge von Linien, Schwellwert für das Incremental-Line-Fitting und maximaler Abstand der Punkte beim Clustering).

Sind alle Parameter optimiert, so liefert FastSLAM sehr gute und genaue Karten. Bis 500 Partikeln läuft der Algorithmus auf einem 1.4 GHz Intel Centrino ohne Probleme in Echtzeit. Die Qualität der Karte ist hauptsächlich von den eingehenden Sensormessungen abhängig. Unterliegen die Daten einem Gauss-Rauschen und sind immer mindestens fünf gut messbare Features vorhanden, so werden auch sehr gute Karten mit einem 1-Partikel-FastSLAM 2.0 erstellt (siehe Abbildung 4.9), die von einer mit 500 Partikeln erstellten Karte kaum zu unterscheiden wären.

Der benötigte Arbeitsspeicher vergrößert sich mit der Zeit geringfügig, da in den erstellten Karten im Laufe der Zeit immer mehr Landmarken gespeichert werden. Nach einer Laufzeit von 30 Minuten steigt der belegte Speicher um etwa 20 MB.

Damit Landmarken (vor allem von nicht mehr existierenden Karten) gelöscht werden konnten wurden diese als shared_ptr (boost, bzw. tr1) gespeichert. Somit konnten verschiedene Karten auf die selbe Landmarke verweisen, die dann gelöscht wurde falls sie in

keiner Karte mehr vorhanden ist.

Probleme bereitet der Algorithmus bei sehr großen Abweichungen in der Messung der selektierten Features. In diesem Fall werden Features falsch zugeordnet, wodurch sich die durch den Kalman-Filter gemessene Ungenauigkeit der Position verringert, ohne dass Landmarken richtig erkannt wurden.

Dies alles führt dazu, dass FastSLAM 2.0 ein recht komplexer Algorithmus ist, welcher zusätzlich noch durch eine Vielzahl von Parametern eingestellt werden muss. Allerdings bietet er auch Vorteile. Die Mapgröße ist nahezu unbegrenzt, da keine Occupancy Maps benötigt werden. Der Algorithmus ist inkrementeller Natur und läuft in Echtzeit. Ebenfalls sind Loop Closures möglich, abhängig von der Anzahl der Partikel, den Messungen und der Resampling Rate.

Literaturverzeichnis

[FTBD01] D. Fox, S. Thrun, W. Burgard, and F. Dellaert. Particle filters for mobile robot localization, 2001.

[IB98] Michael Isard and Andrew Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[May79] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. 1979.

[MT03] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam, 2003.

[MT07] Michael Montemerlo and Sebastian Thrun. *FastSLAM*. Springer, 2007.

[NMTS05] V. Nguyen, A. Martinelli, N. Tomatis, and R. Siegwart. A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics. *IEEE International Conference on Intelligent Robots and Systems*, 2005.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[WB04] Greg Welch and Gary Bishop. An introduction to the kalman filter, 4 2004.