FACHHOCHSCHULE BRANDENBURG

Deckblatt zur Semesterarbeit

Bitte ankreuzen!

Seminarschein ☐ ***Projektschein*** ***X***
(Hausarbeit)

Studiengang/Studienrichtung	SG Informatik (Bachelor)
Prüfungsfach	Autonome Mobile Systeme
Name des Prüfers	
Abgabetermin	3. Februar 2010

Vorname	Nils
Name	Petersohn
Matrikelnummer	20022749

Thema: Partikelfilter in ARNL

Bewertung: ☐

Brandenburg, den..... .....
Unterschrift des Prüfers/der Prüferin

(zul. Notenspektrum lt. RPO der FHB: 1,0; 1,3; 1,7; 2,0; 2,3; 2,7; 3,0; 3,3; 3,7; 4,0; 5,0)

Abschlussaufgabe V - Partikelfilter in ARNL

Nils Petersohn

3. Feb. 2010

Inhaltsverzeichnis

Tabellenverzeichnis

1 Aufgabenstellung

Es soll die Monte-Carlo-Lokalisierung in Arnl untersucht werden. Durch Experimente und Recherche soll der genaue Ablauf des Partikelfilters auch als MCL(Monte-Carlo-Lokalisierung) und die Wirkung der Parameter dargestellt werden. Ausserdem

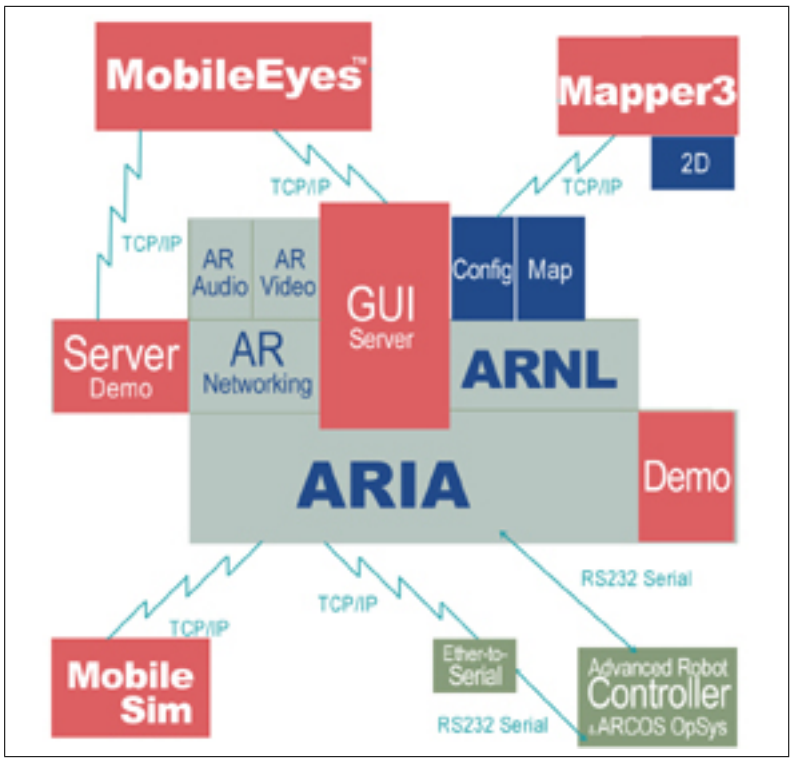


Abbildung 1: ARIA Architektur

soll geklärt werden wie die Lokalisierung in eigenen Programmen genutzt werden kann.

2 Einführung

2.1 ARIA

Die C++ Bibliothek ARIA ist eine API für alle MobileRobots/ActivMedia Plattformen. Mit der Hilfe von ARIA kann u.a. Geschwindigkeit, Richtung oder relative Richtung entweder durch einfache oder komplexe Anweisungen kontrolliert werden. Sie umfasst viele Komponenten wie z.B. ARNL in Abbildung 1

2.2 ARNL

ARNL(Advanced Robotics Navigation and Localization System) ist eine Teilbibliothek in ARIA um Navigations und Lokalisationsschnittstellen zur Verfügung zu stellen. Die Lokalisationskomponente bietet eine Schnittstelle um den angeschlossenen Roboter zu lokalisieren. Sensor und Odometriedaten des Roboters werden in

Verbindung mit einer vorhanden Umgebungskarte verwendet um die wahrscheinlichste Position des Roboters in der Karte zu bestimmen. [Wiki-ARNL] Die ARNL Bibliothek beinhaltet Lokalisierung mit Sonar (SonArnl), Laserscanner (Arnl) und GPS (Mogs).

2.3 ArNetworking

ArNetworking ist eine API in ARIA um eine client-server Architektur zwischen dem Roboter(Server) und dem entfernten Rechner(Client) über das Netzwerk aufzubauen. Der Client berechnet Bewegungsbefehle und sendet Anfragen an den Server welche dann vom Roboter ausgeführt werden. Der Roboter hingegen sendet u.a. seine Odometriedaten.

2.4 MobileEyes

MobileEyes kann sich mit ARIA, ArNetworking und ARNL Server verbinden und dessen Umgebung also interne Karte anzeigen. Unter anderem kann man mit MobileEyes Bewegungskomandos an den Roboter schicken.[whitebrook2010]

3 Theoretischer Hintergrund

Das Lokalisierungsmodul von ARNL benutzt die Monte-Carlo-Lokalisierung. Diese basiert auf der Markov-Annahme welche wiederum auf die Bayes-Regel fundiert.

3.1 Probabilistische Lokalisierung

Um die Position eines Roboters zu berechnen, gibt es verschiedene Ansätze. Sie beruhen alle auf der Fusionierung von Daten, die durch die Odometrie und weitere Sensoren des Roboters geliefert werden. Gebräuchliche Sensoren für autonome Roboter sind Ultraschallsensoren oder Laserscanner. Diese Ansätze verwenden Methoden der Wahrscheinlichkeitsrechnung zur Bestimmung einer Wahrscheinlichkeitsverteilung über alle möglichen Positionen, an denen der Roboter sich befinden kann. Dieser Ansatz ist mittels der Monte-Carlo-Method in ARNL umgesetzt. Der Probanbilistische Ansatz vereint relative und absolute Postionierung als Bewegungsmodellierung und Wahrnehmung der Umwelt.

3.2 Markov-Lokalisierung

Das Ziel der Markov Lokalisierung ist es, Wahrscheinlichkeiten für das Eintreten zukünftiger Ereignisse anzugeben. Die aktuelle Position hängt nur von der vorhergehenden Position und dem Bewegungsbefehl ab und die aktuelle Beobachtung hängt nur von der aktuellen Position ab. Es ist ein zellbasiertes Verfahren bei dem jeder Zelle Wahrescheinlichkeitswert oder Belief zugeordnet wird. [Zweigle] Jede Bewegung des Roboters verschiebt die Wahrscheinlichkeiten und streut sie und jede Messung von Merkmalen der Umgebung ändert die Wahrscheinlichkeiten. Wenn die Position unbekann ist erfolgt eine Gleichverteilung. Das Problem dabei ist, dass alle

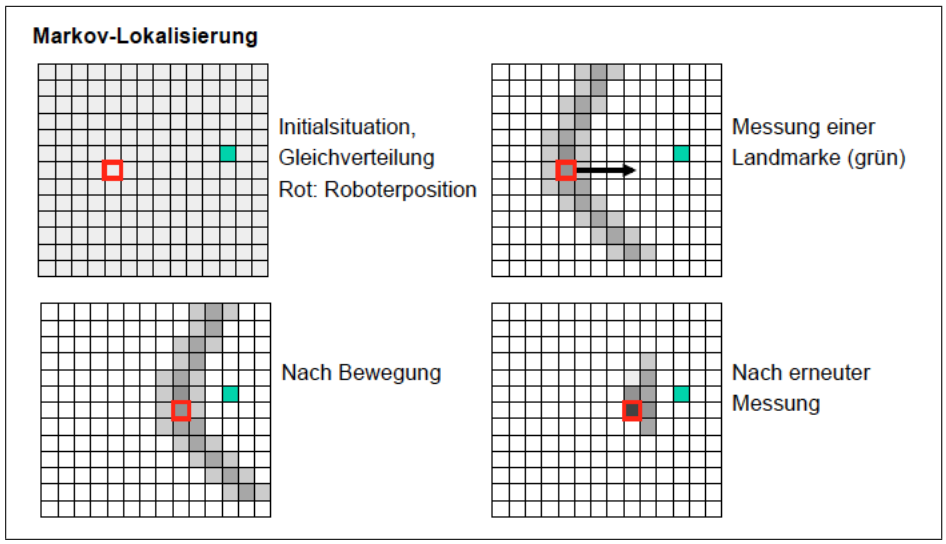


Abbildung 2: Markov-Lokalisierung [Zweigle]

Werte(Zellen) nach jeder Bewegung neu berechnet werden muessen Siehe Abbildung 2. Es gibt nur eine mögliche Position des Roboters und eine unsicherheit wird dabei nicht repräsentiert.

3.3 Partikelfilter

Bei der Lösung mittels Partikel-Filern wird die Pose des Roboters über eine Partikelwolke repräsentiert.[WIKI-Partikelfilter] Jeder Partikel stellt eine mögliche Pose des Roboters dar. ber den Partikelfilter wird jeder Partikel, also jede dadurch repräsentierte Pose, auf ihre Plausibilität überprüft. Die Wahrscheinlichkeit plausibler Partikel wird heraufgesetzt, die Wahrscheinlichkeit wenig plausibler Partikel wird reduziert. Fallen Partikel unter einen bestimmten Wahrscheinlichkeits-Schwellwert, werden sie verworfen.

3.4 Monte-Carlo-Lokalisierung

Der Vorteil des MCL - Ansatzes ist, dass zur Laufzeit die Grösse der Stichprobenmenge variabel sein kann. Je unsicherer die Roboterposition ist, desto gröer fällt die Stichprobenmenge aus. In Analogie zur Gitterbasierenden (grid-based) Methode, müsste bei einer hohen Sicherheit (belief) nur ein Teil des Zustandsraumes aktualisiert werden.[Delipetkos1] Der Grundgedanke bei der Monte Carlo Lokalisierung ist, das Belief (Vertrauen) mittels einer Stichprobenmenge, welche auch als samples oder auch particles bezeichnet wird, darzustellen. MCL ist ein iterativer Bayesscher Filter, welcher als ein Schätzer für die zukünftige Wahrscheinlichkeitsverteilung der Roboterposition verwendet wird. Es werden mehrere Schritte immerzu durchlaufen:

- Prognose-Schritt: jedes Sample erzeugt ein neues Sample entsprechend unscharfem Bewegungsmodell. Resultat ist eine neue Samplemenge bel‘
- Korrektur-Schritt: Jedes Partikel aus bel‘ erhält ein Gewicht nach dem Sensormodell (wie eine Fitness, bereinstimmung mit Wahrnehmung)
- neue Samplemenge bel‘ wird durch zufällige Selektion anhand der Fitness aus bel erzeugt.

In Abbildung 3 ist ein Zustandsdiagramm der Berechnungsschritte des Partikelfilters. Diese Schritte sind in ARNL umgesetzt. Es erfolgt also eine ständige Evolution einer Partikelwolke anhand von Bewegungsbefehlen und Sensorwahrnehmung. Gegenüber der Markov Lokalisierung erlaubt die Monte-Carlo-Lokalisierung eine elegante Fusion der Daten verschiedener Sensorik, sie ist skalierbar (Partikelanzahl) und effizient, bspw. dynamische Anpassung der Partikelanzahl (adaptiv).

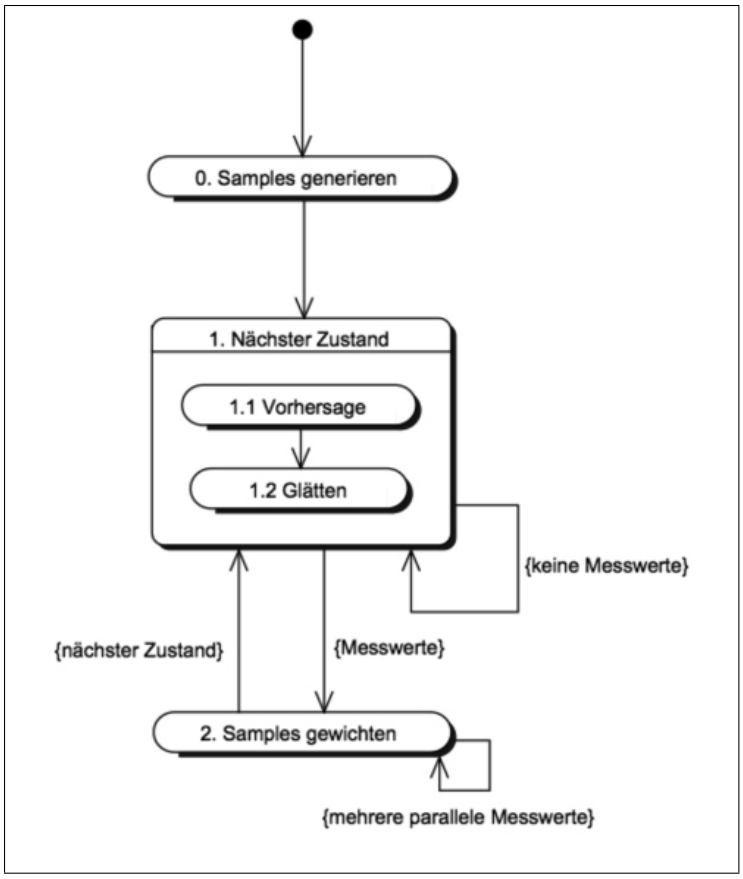


Abbildung 3: Die Initialisierung (0.) erfolgte mit einer Gleichverteilung. Fuer die Propagation des Zustandes (1.) wurde das lineare Modell der Steuerungssoftware uebernommen [Plagge]. Die Berechnung der a-posteriori-Wahrscheinlichkeit (2.) positionsabhängigen Parametern durchgeführt.

4 Experimente

In den folgenden Experimenten werden die zwei Hauptprobleme der Lokalisierung und deren Umsetzung in ARNL untersucht: global localization and position tracking. [FOX1999]

4.1 Positionsverfolgung (Tracking)

In einem Beispielprogramm wird ARNL und ArNetworking verwendet um einen Server zu erstellen. Clients wie MobileEyes können sich mit diesem Server verbinden. MobileEyes sendet anfragen an den Server um den Roboter zu steuern, die aktuelle Karte abzufragen, Lokalisierungsparameter zu setzen, Sensordaten abzufragen und den Roboter manuell wieder neu zu lokalisieren. Die automatische lokalisierung des Roboters erfolgt durch eine Instanz von ArLocalizationTask welcher also Teil des Serverprograms ist. Wie in Abschnitt 5 deutlich gemacht wird, ist das Monte-Carlo Verfahren in der Klasse ArLocalizationTask implementiert. Durch die Verwendung von MobileEyes kann eine Anpassung der Lokalisierungsparameter Einfluss auf den Partikelfilter genommen werden. Die Einstellungen befinden sich in MobileEyes unter Lokalisierungseinstellungen. Eine liste und deren Beschreibung befinden sich in Appendix I. Der Monte Carlo Filter verlässt sich unter anderem auf die Odometriedaten die von den Rädern kommen. Dabei kann bestimmt werden wie weit der Roboter gefahren ist, wenn sich die Räder unterschiedlich bewegen oder wie weit sich der Roboter um seine Achse gedreht hat. Wenn man die genaue Startposition des Roboters kennt und auch die Odometriedaten der Räder, so kann man theoretisch die genaue Position des Roboters zum Zeitpunk t bestimmen. In der Praxis ist dies aber nur schwer zu realisieren. Die Unsicherheit wächst also mit zunehmender Entfernung vom Startpunkt. generell gibt es drei Odometrie Fehlerarten:

- Entfernungsfehler: Fehler beim zurücklegen einer geraden Strecke
- Drehfehler: Fehler beim Drehen des Roboters
- Driftfehler: Orientierungsfehler beim Zurücklegen einer geraden Strecke

Das Bewegungsmodell in ARIA/ARNL bildet genau diese Fehler ab. Um die Bewegungsunschärfe zu testen müssen verschiedene Initialalparameter verwendet werden:

- PassThreshold muss auf 0 gesetzt werden um zu vermeiden, dass der Roboter in den Lost Modus geht
- Um die Partikelwolke besser sichtbar zu machen wird der Wert NumSamples auf 5000 gesetzt
- PeturbX, PeturbY, PeturbTh sind Paramter um neue Samples nach dem resampling prozess hizuzufügen. Diese Streuwerte werden auf null gesetzt um den Rechenanspruch zu verrringern.

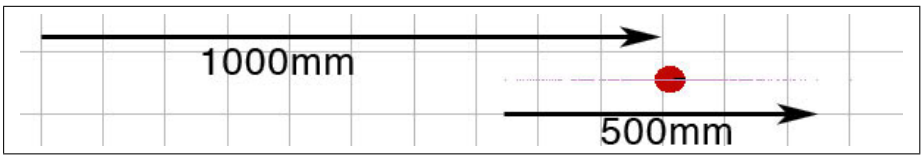


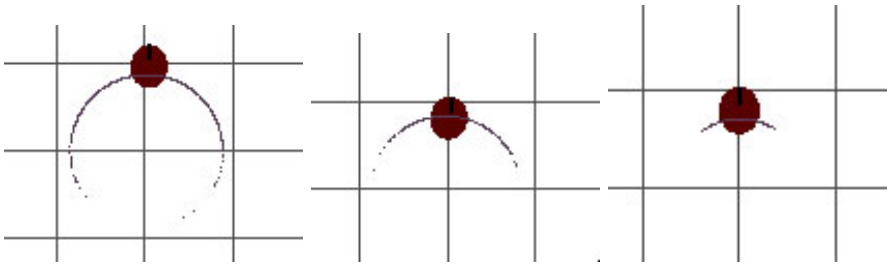
Abbildung 4: KMmPerMm = 0.5

- Per default wird die Neuberechnung der Position erst nach einer Bewegung von 20 cm gemacht. Eine Neuberechnung soll erst nach 2m stattfinden. Somit wird der Paramter Triggerdistance auf 2000 gesetzt.
- Die Neuberechnung nach einer Drehnung soll nicht erst ab 5 Grad erfolgen sondern schon ab 0 Grad. Der Paramter **TriggerAngle** wird also auf 0 gesetzt.

Um den Entfernungsfehlerverhältnissparameter zu testen wird der Drehfehlerparameter und der Driftfehler zuerst ausgeschaltet. Um den Parameter zu untersuchen wird er auf 0.5 gesetzt. Dies soll bewirken, dass die Partikelwolke sich um die Hälfte der zurückgelegten Strecke ausdehnt. Der Roboter wird zuerst an einem Punkt lokalisiert. Mit einem Bewegungsbefehl wird der Roboter nun 1 Meter gerade aus bewegt. Wie angenommen hat die Partikelwolke nun eine Länge von 50 cm.

Dieser Parameter kann dazu verwendet werden um die Lokalisation in bezug auf die Oberflächenbeschaffenheit anzupassen. In einem hügligem Gelände kann der Roboter eine Steigung hochfahren sich aber im eigentlichen Sinne nicht so schnell weiterbewegen wie die Odometriedaten es übermitteln. Desshalb muss dieser Parameter angemessen erhöht werden.

Mit dem Drehfehlerparameter $KDegPerDeg$ kann nach der Dokumentation in Anhang A die Unsicherheit der Drehung beeinflusst werden. Wenn der Roboter sich z.B. auf einem leicht unebenen Untergrund dreht ist die Drehung nicht eindeutig an den Odometriedaten nachvollziehbar. Um diesen zu testen wird der Parameter zuerst auf 1 gesetzt und Entfernungsfehlerverhältnissparameter wieder auf null. In Abbildung 5a wurde der Roboter um 360 Grad gedreht und dann 1cm gerade aus gefahren. Der Parameter wurde in Abbildung 5b und 5c stufenweise halbiert um den Winkel der Partikelstreuung zu erkennen. Dieser Entspricht demnach dem Verhältnis zwischen der Drehung in Grad und der KDegPerDeg Einstellung. Bei einer Drehnung von 360 Grad und einer Einstellung von 1 hat der Partikelfilter eine breite von auch ca. 360° bei 0.5 von ca. 180° und bei 0.25 von ca. 90°. Wenn also der Untergrund hüglig ist, sollte die Unsicherheit also der KDegPerDeg Paramter höher eingestellt sein.



(a) KDegPerDeg = 1 (b) KDegPerDeg = 0.5 (c) KDegPerDeg = 0.25

Abbildung 5: KDegPerDeg Parameter Test

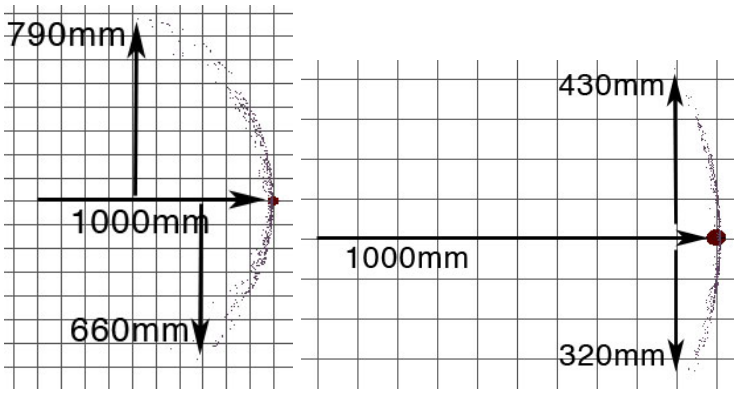
Der Parameter $KDegPerMm$ betrifft die Unsicherheit der Odometriedaten der Räder. Bei einem differenziellem Antrieb können leichte Unterschiede auftreten, wenn der Roboter sich gerade aus bewegt. Wenn der Roboter sich z.B. auf einem unebenen Boden fortbewegt, kann er nach links und rechts abdriften. Die Odometriedaten der Räder geben aber ein stetiges geradeausfahren zurück.

In dem Experiment in Abbildung 6 wird der Parameter $KDegPerMm$ zuerst auf 0.02 (Abbildung 6a) dann auf 0.01 (Abbildung 6b) und dann auf 0.005 (Abbildung 6c) gestellt. Der Roboter wird lokalisiert und dann 1000mm gerade aus gefahren. Um die Korrektheit der Dokumentation zu ueberpruefen wird durch eine einfache Winkelfunktionsrechnung in Abbildung 7 der Winkel errechnet. Zuerst wird der Mittelwert der Abweichung durch das arithmetische Mittel bestimmt siehe Formel 1

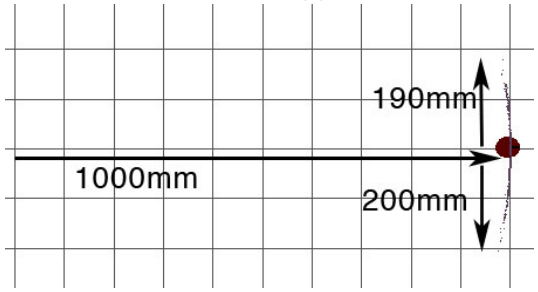
Um die Rechnungen zu bestätigen wird der Wert ausgerechnet welcher zu einem Fehlerwinkel von 180° nötig ist und bilden dann davon das Arithmetische Mittel. Wenn dieses nicht sonderlich abweicht waren die Rechnungen richtig.
Dies ist der Fall in Abbildung 11 denn eine maximale Abweichung von weniger als 0.012 und eine Minimale Abweichung von 0.0009 ist anehmbar.

Die Bestätigung liefert eine Einstellung von KDegPerMm = 0.04 welche eine 180° ausgedehnte Partikelwolke aufspannt siehe Abbildung 12.

Somit lässt sich der Parameter nachvollziehen. Wenn man einen Winkel α erreichen will muss man Rechnen: $KDegPerMm = \alpha \cdot 2.22 \cdot 10^{-4}$ Diese Rechnung ist nicht genau und wurde mit Rundungswerten ermittelt.



(a) KDegPerMm = 0.02 (b) KDegPerMm = 0.01



(c) KDegPerMm = 0.005

Abbildung 6: KDegPerMm Parameter Test

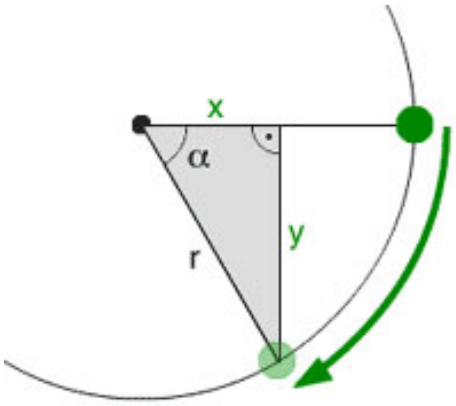


Abbildung 7: Winkelfunktionen im Einheitskreis
`http://www.ullala.at/experiments/movement/images/circ_1.gif`

$$\bar{y} = \frac{790 + 660}{2} = 725mm \tag{1}$$

$$\theta_1 = \sin^{-1}\left(\frac{y}{r}\right) \cdot 2 \tag{2}$$

$$\theta_1 = \sin^{-1}\left(\frac{725}{1000}\right) \cdot 2 \approx 93^\circ \tag{3}$$

Abbildung 8: Berechnung für Abbildung 6a

$$\bar{y} = \frac{430 + 320}{2} = 375mm \tag{4}$$

$$\theta_2 = \sin^{-1}\left(\frac{y}{r}\right) \cdot 2 \tag{5}$$

$$\theta_2 = \sin^{-1}\left(\frac{375}{1000}\right) \cdot 2 \approx 44^\circ \tag{6}$$

Abbildung 9: Berechnung für Abbildung 6b

$$\bar{y} = \frac{190 + 200}{2} = 195mm \tag{7}$$

$$\theta_3 = \sin^{-1}\left(\frac{y}{r}\right) \cdot 2 \tag{8}$$

$$\theta_3 = \sin^{-1}\left(\frac{195}{1000}\right) \cdot 2 \approx 23^\circ \tag{9}$$

Abbildung 10: Berechnung für Abbildung 6c

$$\frac{\theta}{KDegPerMm} = \frac{180^\circ}{x} \tag{10}$$

$$x_0 = \frac{180^\circ \cdot 0.005}{23} = 0.0391 \tag{11}$$

$$x_1 = \frac{180^\circ \cdot 0.01}{44} = 0.0409 \tag{12}$$

$$x_2 = \frac{180^\circ \cdot 0.02}{93} = 0.0387 \tag{13}$$

$$\bar{x} = \frac{x_0 + x_1 + x_2}{3} \approx 0.04 \tag{14}$$

Abbildung 11: Test der Berechnungen in Abbildung 3,6,9

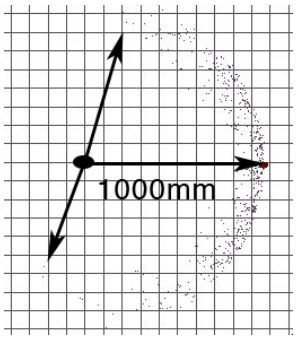


Abbildung 12: KDegPerMm = 0.04

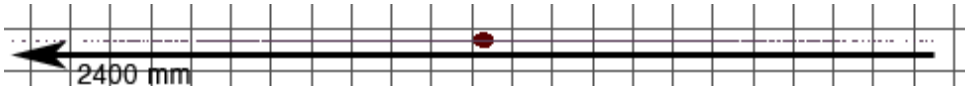


Abbildung 13:
NumSamplesAtInit = 20000,
StdX = 4000,
StdY = 0,
StdTh = 1

4.2 Globale Positionierung

Wenn die Initiale Position des Roboters nicht bekannt ist, dann werden Samples ausgestreut. Die Menge der ausgestreuten Samples und deren Verteilung wird u.a. durch die Parameter NumSamplesAtInit, StdX, StdY und StdTh eingestellt. Um diese Parameter zu testen werden Extremwerte verwendet. Die Parametereinstellung in Abbildung 13,14,15 soll verdeutlichen welche Auswirkungen eine Eindimensionale Ausbreitung auf der X Achse für Konsequenzen mit sich ziehen. Die Streuung der Samples bei einem Wert von StdX = 4000mm ist nicht wie erwartet 4000mm in Richtung der X Achse sondern 2400mm wie in Abbildung 13. Der eigentliche Wirkung von StdX ist also $\approx (StdX/2) + (\frac{StdX}{10})$ mm. Und nicht wie angenommen $StdX$ mm. In Abbildung 14,15 wird dies bestätigt. Eine grosse Ausdehnung bei der initialen Lokalisierung der Partikelwolke erfolgt durch die Einstellungen wie in Abbildung 16

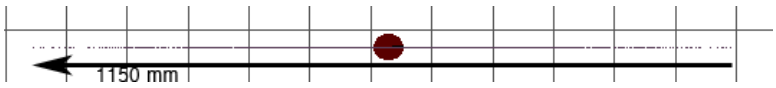


Abbildung 14:
NumSamplesAtInit = 20000,
StdX = 2000,
StdY = 0,
StdTh = 1

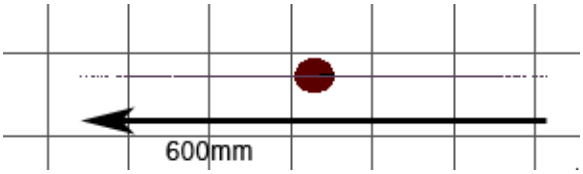


Abbildung 15:
NumSamplesAtInit = 20000,
StdX = 1000,
StdY = 0,
StdTh = 1

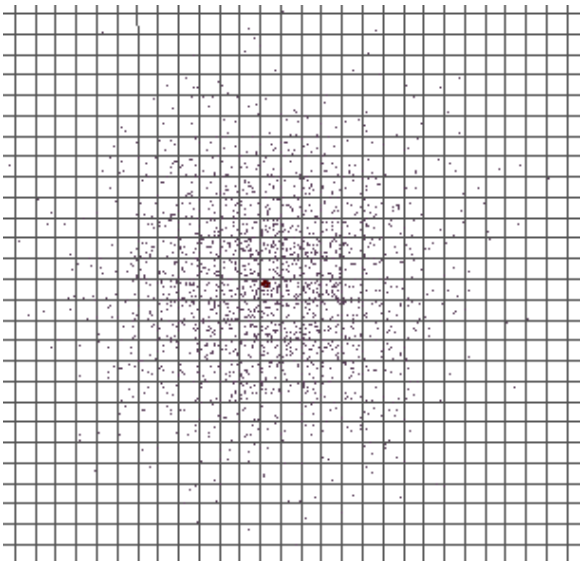


Abbildung 16:
NumSamplesAtInit = 20000,
StdX = 4000,
StdY = 4000,
StdTh = 30

5 Nutzung der Lokalisierung in eigenen Programmen

Die Klasse ArLocalizationTask kann Daten vom Laserscanner (ArSick) auswerten um die Lokalisierung durchzuführen. Abgesehen vom ArLocalizationTask gibt es noch andere Klassen für die Lokalisierung. Die Elternklasse ist ArBaseLocalizationTask in der BaseARNL Bibliothek. [Wiki-ARNL] Um den ArLocalizationTask zu instanzieren müssen Pointer von Instanzen der Typen ArRobot, ArRangeDevice und ArMapInterface (ArMap) ürbgeben werden. Eine Karte kann durch senorDaten erzeugt werden aber auch manuell durch das Program Mapper3. Die Instanz von ArLocalizationTask startet einen eigenen asynchronen Hintergrund-thread um anhand des Lasersensors wiederholend den Roboter zu lokalisieren. Die Position wird anhand der Berechnungen im ArLocalizationTask neu gesetzt.

```
#include "ArLocalizationTask.h"
...
ArRobot robot;
...
ArLaser *firstLaser = robot.findLaser(1);
...
ArMap map(fileDir);
...
ArLocalizationTask locTask(&robot, firstLaser, &map);
```

Um eine initiale Lokalierung durchzuführen muss die Methode localizeRobotAtHomeBlocking ohne Parameter aufgerufen werden. Eine Instanz von ArLocalizationManager erlaubt Ausgabedaten von verschiedenen Lokalisationsthreads zu vereinigen. Durch die Methode addLocalizationTask kann eine neue Instanz von ArBaseLocalizationTask wie z.B. ArSonarLocalizationTask oder ArLocalizationTask hinzugefügt werden. Durch die Methode fuseTwoDistributions kann man die Daten fusionieren.

```
ArLocalizationManager locManager(&robot, &arMap);
ArLocalizationTask locTask (&robot, &sick, &arMap);
locManager.addLocalizationTask(&locTask);
```

Die KonfigurationsDatei arnl.p ist für eine erfolgreiche Lokalisation notwendig. Darin befinden sich u.a. Einstellungen für die Monte-Carlo-Lokalisation. Diese Konfigurationsdatei wird durch einen Parser ausgelesen:

```
ArArgumentParser parser(&argc, argv);
parser.loadDefaultArguments();
...
ArServerHandlerConfig handlerConfig (
        &server, Aria::getConfig (),
         Arnl::getTypicalDefaultParamFileName (),
         Aria::getDirectory ()
);
```

```
// Read in parameter files.
Aria::getConfig()->useArgumentParser(&parser);
Aria::getConfig()->parseFile (Arnl::getTypicalParamFileName() )
```

6 Zusammenfassung

Die Experimente haben gezeigt, dass der Partikelfilter gut in ARNL umgesetzt ist. Viele Einstellungsmöglichkeiten bei MobileEyes machen eine gute Lokalisierung möglich. Durch die Verwendung der Monte-Carlo-Lokalisierung sinkt die Hardwareanforderung drastisch und es werden schnelle und gute Ergebnisse erziehlt. Eine genaue Analyse der Funktion ist nur duch reverse Engeneering möglich, denn wichtige Klassen stehen nicht unter der Open Source Lizenz. Auch ist die Dokumentation unzureichend und das verwendete Englisch ist nicht nativ was den Analyseprozess zusätzlich erschwert hat. Dennoch kann mit dem umgesetzten Lokalisationsmodul in ARIA gut gearbeitet werden. In ARIA sind die Methodennamen sind verständlich und aussagekräftig. Daher ist eine intuitive Benutzung gegeben.

A Appendix I

Parametername Standartwert Minimumwert, Erklärung

Map Karte der Umgebung um zu Navigieren.

NumSamples 2000 minumum 0, Die (maximale) Anzahl der Partikel wird hier festgelegt. Je höher dieser Wert ist desto mehr Rechenleistung wird benötigt. In dieser Arbeit wurde in einem Bereich von 5000-20000 Partikel gearbeitet.

NumSamplesAtInit Wenn der Roboter in der Karte neu initialisiert wird, dann bedeutet ein Wert von 0, dass $NumSamples$ Partikel benutzt werden. Bei den Experimenten wurde ein Wert von ca. 20000 eingestellt um die Initiale Partikelanzahl vorzugeben. Bei der Initialisierung wird somit eine Gleichverteilung von 20000 Samples ausgeführt

GridRes 100 minimum 10, Die Auflösung der Gitter in MobileEyes. Der Standartwert wurde bei den Experimenten verwendet.

PassThreshold 0.2 range [0, 1], Wenn die LLS unter diesen Wert fällt, dann befindet sich der Roboter im 'Lost' Zustand. Bei den meisten Experimenten wurde dies mit einem Wert von 0 vermieden. LLS = Laser Localization Score = das aktuelle maximale Gewicht, d.h. wie gut stimmt das beste Sample mit der Sensorwahrnehmung überein, Maximum 1.0

KMmPerMm 0.05 minimum 0, Entfernungsfehler in mm per mm. Wurde in Abschnitt 4.1 erklärt.

KDegPerDeg 0.05 minimum 0, Drehfehler in Grad per DrehungsGrad.Wurde in Abschnitt 4.1 erklärt.

KDegPerMm 0.0025 minimum 0, Driftfehler in Grad per mm. Wurde in Abschnitt 4.1 erklärt.

TriggerDistance 200 minimum 0, Lokalisierung wird erst nach einer gefahrenen Strecke von $TriggerDistance$ durchgefuehrt. Der Standartwert wurde bei den Experimenten verwendet.

TriggerAngle 5 minimum 0, Lokalisierung wird erst nach einer Drehung von $TriggerAngle$ durchgefuehrt. Gemessen in Grad.

TriggerTimeEnabled false

Die Lokalisierung soll nicht nach einer abgefahrenen Strecke oder einer bestimmten Drehung erfolgen sondern nach einem Zeitintervall $TriggerTime$ Diese Einstellung wurde auch getestet. Jedoch wurde der Roboter immer wieder in den 'Lost' Zustand gebracht und neu Lokalisiert. Daher war diese Einstellung war für die Experimente nicht verwendbar.

TriggerTime 10000 minimum 1500, Zeitinterval in msec für den Lokalisierungstrigger.

IdleTimeTriggerX 200 minimum 0, Streuung der Samples auf der X Achse in mm wenn der Lokalisierungstrigger eingeschaltet ist.

IdleTimeTriggerY 200 minimum 0, Streuung der Samples auf der Y Achse in mm wenn der Lokalisierungstrigger eingeschaltet ist.

IdleTimeTriggerTh 15 minimum 0, Ausrichtung der Samples (θ) in Grad.

RecoverOnFail false Bei einer positiven Einstellung (true) wird der Roboter ,wenn er in den 'Lost' Zustand uebergeht durch eine statische Reinitialisierung unter Verwendung der letzten bekannten Position neu lokalisiert. Bei den Experimenten war diese Einstellung jederzeit negativ eingestellt.

FailedX 300 minimum 0, Streuung der Samples auf der X Achse bei einer statische Reinitialisierung durch den 'Lost' Zustand in mm.

FailedY 300 minimum 0, Streuung der Samples auf der Y Achse bei einer statische Reinitialisierung durch den 'Lost' Zustand in mm.

FailedTh 45 minimum 0, Streuung der Samples in einem bestimmten Bereich bei einer statische Reinitialisierung durch den 'Lost' Zustand in Grad.

PeturbX 10 minimum 0, Nachdem die Sensorwerte normalisiert wurden und der Resamplingprozess stattgefunden hat, kann durch diese Einstellung eine grössere Streuung auf der X Achse durch neue Samples vorgenommen werden. Gemessen in mm.

PeturbY 10 minimum 0, Nachdem die Sensorwerte normalisiert wurden und der Resamplingprozess stattgefunden hat, kann durch diese Einstellung eine grössere Streuung auf der Y Achse durch neue Samples vorgenommen werden. Gemessen in mm.

PeturbTh 1 minimum 0, Nachdem die Sensorwerte normalisiert wurden und der Resamplingprozess stattgefunden hat, kann durch diese Einstellung eine grössere Streuung in einem bestimmten Winkel vorgenommen werden.

PeakStdX,PeakStdY,PeakStdTh 10 minimum 0, Eine weiter einstellung um die grösse der Streuung (grösse der Ellipse) zu beeinflussen

PeakFactor 1e-06 range [0, 1], Wenn manche Partikel eine Wahrscheinlichkeit ungleich null haben dann ist diese Einstellung der Schwellwert fuer eine valide Hypothese ansonsten ist die Hypothese nicht valide und wird verworfen. Der Schwellwert bezieht sich auf die maximale Wahrscheinlichkeit LLS

StdX 400 minimum 0, Die gausssche Wahrscheinlichkeitsverteilung bei dem Beginn der Lokalisierung (σ) in Richtung der X-Achse. Diese Parameter wurde in Abschnitt 4.2 behandelt.

StdY 400 minimum 0, Die gausssche Wahrscheinlichkeitsverteilung bei dem Beginn der Lokalisierung (σ) in Richtung der Y-Achse. Diese Parameter wurde in Abschnitt 4.2 behandelt.

StdTh 30 minimum 0, Die gausssche Wahrscheinlichkeitsverteilung bei dem Beginn der Lokalisierung (σ) in Grad. Diese Parameter wurde in Abschnitt 4.2 behandelt..

SensorBelief 0.9 range [0, 1], Die Laserdaten sind zu einem gerringem Mass nicht richtig. Mit diesem Parameter kann die Glaubwürdikeit der Laserdaten einstellen.

Kalman Filter Weitere Parameter betreffen den Kalman Filter welcher nicht Bestandteil dieser Arbeit ist.

Literatur

[Wiki-ARNL] MobileRobots Inc, *ARNL, SONARNL and MOGS*, wiki (9 September 2009), available at `http://robots.mobilerobots.com/wiki/ARNL,_SONARNL_and_MOGS`.

[Delipetkos1] Fraunhofer Gesellschaft AIS.ARC, Particle Filter Ein probabilistischer Ansatz zur Lokalisierung mobiler Roboter available at S. 12 `http://www.ais.fraunhofer.de/~delipetk`

[FOX1999] Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. Dieter Fox,Wolfram Burgardy, Frank Dellaert, Sebastian Thrun School of Computer Science Carnegie Mellon University Pittsburgh, PA 1999

[ARNL-Ref] MobileRobots Inc, *ARNL-Reference*, (1.7.0) available at `http://vigir.missouri.edu/~gdesouza/Research/MobileRobotics/Software/ARNL-SONARNL/Arnl-1.7.0+gcc41/docs/ARNL-Reference/index.html`.

[whitebrook2010] Programming Mobile Robots with Aria and Player Dr. Amanda Whitbrook University of Nottingham School of Computer Science, Springer-Verlag London Limited 2010

[Zweigle] Vorlesung 5 Lokalisierung und Mapping - Universität Stuttgart Institut für Parallele und Verteilte Systeme (IPVS) `http://www.ipvs.uni-stuttgart.de/abteilungen/bv/lehre/lehrveranstaltungen/vorlesungen/SS09/Robotik%20II_termine/dateien/robotik_8_lokalisierung_2.pdf`

[Borens 1996] Borenstein, J., Everett, H.R., Feng, L., April 1996, Where am I? Sensors and Methods for Mobile Robot Positioning. Technical Report, The University of Michigan. Seite 131 Chapter 5: Dead-Reckoning

[WIKI-Partikelfilter] Lokalisierung (Robotik) Abschnitt Partikel-Filter. Datum 2.2.2010 `http://de.wikipedia.org/w/index.php?title=Lokalisierung_(Robotik)#Partikel-Filter`

[Borens 1996] Borenstein, J., Everett, H.R., Feng, L., April 1996, Where am I? Sensors and Methods for Mobile Robot Positioning. Technical Report, The University of Michigan. Seite 131 Chapter 5: Dead-Reckoning

[Plagge] M. Plagge. Personal communication: Pioneer baseserver, 2000.

[Neumann2002] Kalman-Filter und Partikelfilter zur Selbstlokalisation Ein Vergleich Dirk Neumann, 2002 `http://www.allpsych.uni-giessen.de/dirk/projects/particle.pdf`

Abbildungsverzeichnis