RWTH AACHEN

PRAKTIKUM

Long Term Simultaneous Localization and Mapping

Autor:
Johnson LOH
Matr.-Nr.: 318755

Betreuer:
Stefan DÖRR

Dieser Praktikumsbericht wurde erstellt für die berufspraktische Tätigkeit für den Masterstudiengang Elektrotechnik, Informationstechnik und Technische Informatik

in dem Betrieb

Fraunhofer Institut für Produktionstechnik und Automatisierung IPA
Nobelstr. 12
70569 Stuttgart

15. Oktober 2017

Inhaltsverzeichnis

Abkürzungsverzeichnis

ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
RBPF	Rao-Blackwellized Particle Filter
HMM	Hidden Markov Model
NDT	Normal Distribution Transform
GPS	Global Positioning System
amcl	Adaptive Monte Carlo Localization
KLD	Kullback Leibler Divergenz
MLE	Maximum Likelihood Estimation
RFID	Radio-Frequency Identification
FTF	Fahrerloses Transportfahrzeug
STL	Standard Template Library

Kapitel 1

Einleitung

1.1 Fraunhofer-Gesellschaft

Die Fraunhofer-Gesellschaft ist mit 67 Instituten und Forschungseinrichtungen die größte Forschungseinrichtung in ganz Europa. Mit über vierundzwanzig tausend Mitarbeitern und Mitarbeiterinnen ist die Fraunhofer Gesellschaft hauptsächlich in ingenieurs- und naturwissenschaftlichen Bereichen der Forschung tätig. Große Themen der Forschung sind dabei Sicherheit, Kommunikation, Mobilität, Gesundheit, Umwelt und Energie. Der jährliche Etat der Fraunhofer Gesellschaft erstreckt sich auf über 2,1 Milliarden Euro, wobei der größte Teil des Etats sich durch Aufträge aus der Wirtschaft, sowie öffentliche Forschungsprojekte zusammensetzt.

Gegründet wurde die Fraunhofer Gesellschaft im Jahre 1949. Der Name Fraunhofer stammt vom gleichnamigen Forscher und Entwickler Joseph von Fraunhofer.

1.2 Fraunhofer-Institut für Produktionstechnik und Automatisierung (IPA)

Das IPA ist eines der größten Institute der Fraunhofer Gesellschaft. Mit knapp 1000 Mitarbeiter und Mitarbeiterinnen und einem Jahresbudget von knapp 60 Millionen forscht das Institut in den Bereichen von Automotive, Maschinen- und Anlagebau, Elektronik- und Mikrosystemtechnik, Energie, Medizin- und Biotechnik sowie Prozessindustrie. Dabei setzt die Finanzierung aus einem drittel Industrieprojekten und zwei drittel staatlich geförderten Mitteln zusammen.

1.2.1 Struktur des Instituts

Das IPA unterteilt sich in 14 verschiedene Fachbereiche, welche zu sechs Geschäftsbereichen zugeordnet werden. Dabei beschäftigt das IPA bis zu annähernd 1000 Mitarbeiter. Eine Visualisierung der Unternehmensstruktur ist in Abb. 1.1 dargestellt.

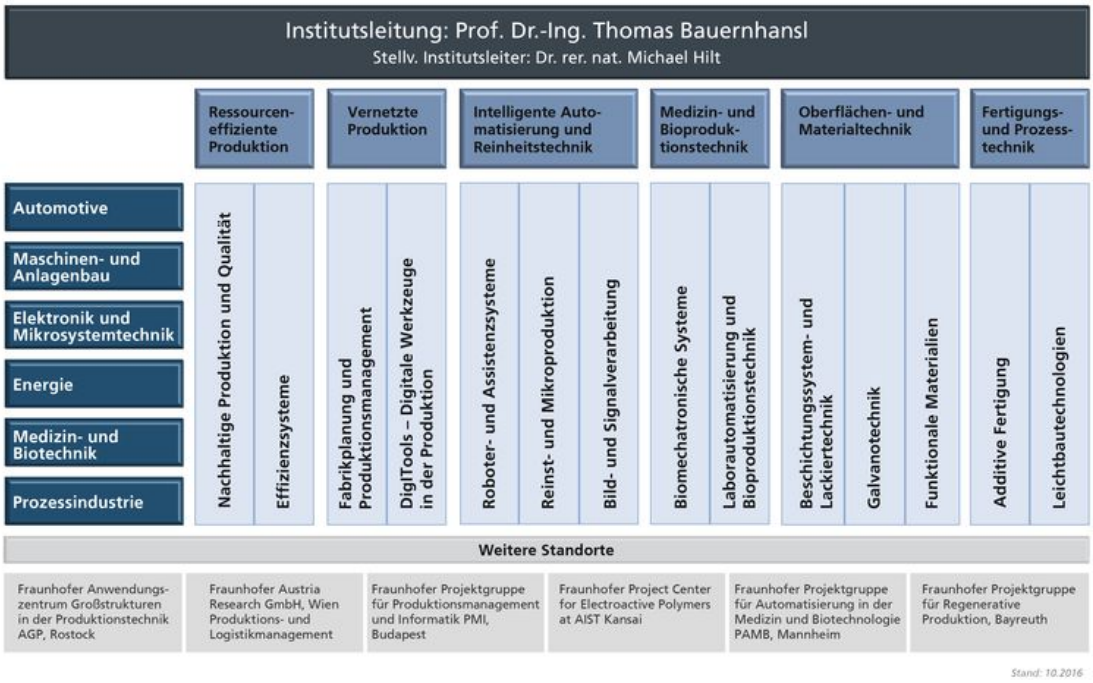


ABBILDUNG 1.1: Fachbereiche des IPA [5]

1.2.2 Roboter- und Assistenzsysteme

Der Fachbereich für Roboter- und Assistenzsysteme ist verantwortlich für die Entwicklung von Automatisierungslösungen und mobile Servicerobotik. Im Zeitalter der Automatisierung sind autonome Systeme die Schlüsseltechnologie sowohl in der Industrie (Stichwort: Industrie 4.0) als auch im Dienstleistungsbereich. Damit diese Systeme sich in einem realem Umfeld orientieren und navigieren können, sind fortgeschrittene Methodiken notwendig, die auch an diesem Insitut entwickelt und angewendet werden.

1.3 Aufgabenbereich

Die Weiterentwicklung von Orientierungsalgorithmen für die mobile Robotik und deswegen auch mein Aufgabenbereich befindet sich daher in der Abteilung für Roboter- und Assistenzsysteme des Instituts.

Dabei wurde mir zunächst zur Einarbeitung ein interdisziplinärer Aufgabenbereich im Bereich der Navigation zugeteilt. Der Fokus liegt darin sich mit ROS (Robot Operating System) sowie dem modularen Aufbau der Navigationskomponenten bekannt zu machen. Nach der Einarbeitung besteht meine Hauptaufgabe den vorhandenen Long Term SLAM (Simultaneous Localization and Mapping) Algorithmus, um diverse Funktionalitäten zu erweitern und zu warten, welche in den folgenden Kapiteln näher erläutert werden.

Kapitel 2

Grundlagen und Stand der Technik

Im ersten Teil dieses Kapitels werden die grundlegenden Themengebiete und Tools zusammengefasst erläutert, die als Vorbereitung meiner Arbeit am Long-Term SLAM notwendig waren. Dabei gehe ich nur auf die Gebiete ein, die von der Thematik her das Fundament des Projektes bilden. Natürlich wurden nebenbei im Praktikum andere Basisskills erworben wie zum Beispiel Versionsverwaltungsysteme wie Git und die Weiterführung vom klassischen im Studium erlerntem C++ (C++11, boost, Eigen etc.). Auf diese wird aber nicht näher eingegangen.

Der zweite Teil behandelt den Stand des Projektes vom Anfang des Praktikums.

2.1 ROS

Das Robot Operating System (ROS) ist eine Middleware[1], die dazu gedacht ist wiederkehrende Softwarefragmente in einem Robotersystem besser verwalten zu können. Somit sollen bereits implementierte Funktionalitäten wiederverwendet werden können, um somit das „Neuerfinden vom Rad“ zu vermeiden. Der Kern von ROS besteht aus folgenden Modulen:

- Publisher/Subscriber System
- Aufnahme und Wiedergabe von Kommunikationseinheiten
- Remote Procedure Calls
- Verteiltes Parameter System

[1] anwendungsneutrale Programme, die zusätzlich zu den Funktionen des Betriebssystems Softwareapplikationen implementieren. Diese vereinfachen überlichweise die Kommunikation zwischen Prozessen.

Nachrichten (in ROS: Topics) von einem Modul (Node) werden anonym und asynchron versendet (Publish) und von anderen Modulen empfangen (Subscribe). Abbildung 2.1 zeigt die Kommunikationsstruktur von ROS. Dabei wird der Datenaustausch über einen zentralen Prozess, dem *ROS Master* verwaltet. Verschiedene Module können auch Funktionen bereitstellen, die wiederum andere Module aufrufen und verwenden können. Dadurch wird Kommunikationen zwischen Sensoren, Servern, Klienten etc. einfach modelliert und verwaltet. ROS bietet auch Anwendungsprogramme an, mit denen die Daten und der Datenaustausch der Module untereinander visualisiert, aufgenommen und wieder abgespielt sowie simuliert werden können (mit z.B. rviz, rosbag, rqt-graph, gazebo etc.) . Auf die innere Infrastruktur und Funktionsweise von ROS wird hier nicht näher eingegangen. Für detailliertere Informationen darüber kann über die offizielle Webseite von ROS ([10]) danach recherchiert werden.

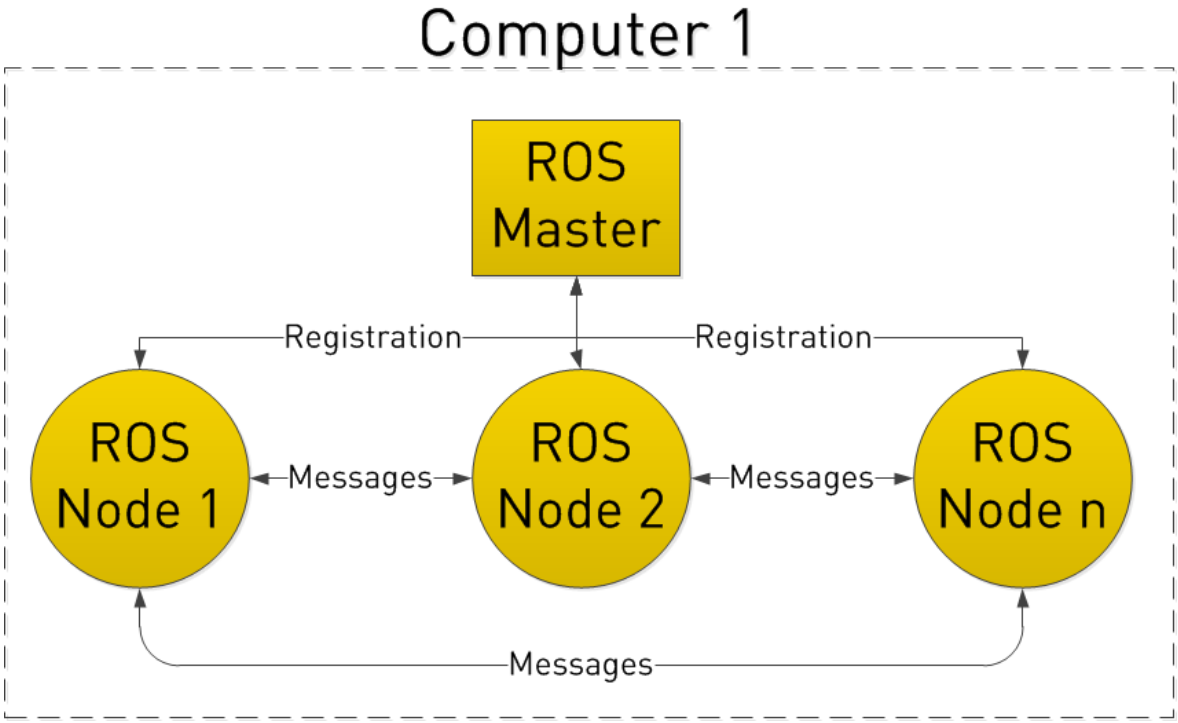


ABBILDUNG 2.1: Grobe Veranschaulichung der Funktionsweise von ROS [9]

Abbildung 2.2 zeigt die Visualisierungstools *rviz* und die Simulationsumgebung *gazebo*. Zum Testen der implementierten Prototypen wird im Praktikum meistens auf die Simulation zurückgegriffen, da sie eine sichere und weniger umständliche Alternative zur realen Umsetzung bietet. Am IPA wird zunächst mit der Version ROS Indigo gearbeitet während im Verlauf des Praktikums auf ROS Kinetic umgestiegen wurde.

2.2 Simultaneous Localization and Mapping (SLAM)

Der Basisalgorithmus auf dem meine Arbeit aufbaut ist der SLAM Algorithmus. Dabei werden eingehende Sensordaten dazu verwendet sowohl eine Karte von der Umgebung zu generieren, als auch sich in dieser Karte zu lokalisieren. Ein klassisches Beispiel wäre ein mobiler Roboter mit LiDAR-Daten. Die Laserscans liefern basierend auf der momentanen Position Längenmessungen und Winkelpositionen, welche in eine inneren Repräsentation der Welt (z.B.: Gridmapkarten oder Landmarken) umgewandelt und gespeichert werden. Mit Hilfe dieser Umgebungsinformationen, die auch vorprogrammiert sein kann, lokalisiert sich der Roboter, d.h. er vergleicht die Sensordaten mit seiner eigenen Karte und ermittelt seine Position neu, sodass die Daten am besten zu den vorhergehenden Informationen passen. Die gleichzeitige Ausführung vom Kartieren und Lokalisieren wird als SLAM bezeichnet.

Aufgrund von nicht vorhersehbaren Einflüssen auf die Umgebung und den Roboter, sowie Bauungenauigkeiten, inkonsistente Fahrflächen etc. ist das Systems nicht deterministisch, deswegen wird die Welt probabilistisch modelliert. D.h. mathematisch gesehen ist die Position von Objekten und vom Objekten nicht direkt bekannt, sondern hat an bestimmten Punkten im Raum eine Wahrscheinlichkeit sich dort aufzuhalten. Unter der Annahme, dass die Welt mithilfe der Wahrscheinlichkeitstheorie dargestellt wird, kann man aus dem Zustandsraum einen guten Wert schätzen. Dazu werden Filteralgorithmen (z.B. Bayes Filter, Extended Kalman Filter, Partikelfilter) verwendet. Für eine genauere Beschreibung dieser Algorithmen verweise ich auf [11].

2.3 Stand des IPA SLAM

Das bestehende Projekt zu Beginn des Praktikums war die Implementierung eines SLAM Algorithmus basierend auf einem Rao-Blackwellized Partikelfilter (RBPF) für die Lokalisierung und einer Hidden Markov Occupancy Grid Map für die Kartierung der Umgebung. Dessen Grundlage setzt die Diplomarbeit von *Barsch, P.* [1].

Der Partikelfilter ermittelt die Position und Umgebung des Roboters in einer Vielzahl von Hypothesen (Partikel), die unterschiedlich gewichtet sind. Diese Gewichtung wird aus der Sensormessung und der Karte berechnet und ist ein Maß dafür wie sehr sie übereinstimmen. Der Algorithmus des Partikelfilters kann in eine Abfolge von folgenden Schritten (nicht notwendigerweise in dieser Reihenfolge) unterteilt werden:

Sampling: In diesem Schritt wird basierend auf dem letzten Zustand des Partikels und der Odometriemessung ein neuer Zustand berechnet. Diese Berechnung nimmt Rücksicht auf die vorherige Wahrscheinlichkeitsverteilung sowie das Bewegungsmodell, dass man für sein Robotersystem annimmt (*motion model*).

Weighting: Im Gewichtungsschritt wird eine Sensormessung mit der Karte verglichen und je nach Übereinstimmung ein entsprechendes Gewicht fürs Partikel kalkuliert.

Mapping: Im Mappingschritt wird die Sensormessung mit in die Karte übernommen.

Resampling: In diesem Prozess werden analog zur natürlichen Selektion die Partikel auf Grund ihrer Gewichtung vervielfacht bzw. vernichtet. Dadurch kommt eine neue räumliche Verteilung zustande

Das RBPF unterscheidet sich vom einfachen theoretischen Partikelfilter dadurch, dass der neue Zustand im Samplingschritt aus einem Unterraum der Wahrscheinlichkeitsverteilung des Zustands ermittelt wird (hier: die Pose vom Roboter ohne die momentane Karte). Der mathematische Hintergrund sowie die präzisere Beschreibung des verwendeten Algorithmus kann aus [7] und [6] entnommen werden. Die IPA Implementierung enthält im Gewichtungsschritt zusätzlich eine Posenoptimierung des Partikels. Durch kleine diskrete Verschiebungen um die eigentliche Pose herum, wird „ausprobiert“, ob sich der Score verbessert oder nicht. Wenn ja, dann wird die entsprechend verschobene Pose weiterverwendet.

Die Karte des Algorithmus ist ein Occupancy Grid Map. Dabei wird die Welt in diskrete Zellen eingeteilt (in diesem Projekt wird eine planare Umgebung angenommen auf der sich der Roboter bewegt, daher 2D), die zwei Zustände annehmen kann: belegt oder frei. Aufgrund von dynamischen Objekten und besseren Konvergenzeigenschaften wurde die Belegtheit einer Zelle mit einem Hidden Markov Modell (HMM), wie bei [12], modelliert. Die zwei Zustände der Zelle besitzen somit Transitionswahrscheinlichkeiten von ihrem alten Zustand zum Neuen, die von Sensormessungen im Mappingschritt geupdatet werden.

Das Projekt implementierte zu jenem Zeitpunkt auch eine kooperative Infrastruktur zum Austausch und Synchronisieren von Informationen mehrerer Roboter oder sensorischen Systeme [2]. Die individuell erfassten Daten werden einem Upstream Server gesendet, welcher die kollektiv erfassten Daten in einer Karte synchronisiert und die gemeinsame Karte den Untersystemen zurückschickt. In der vorherigen Abb. 2.2 kann man die Interaktion gut erkennen. Dabei ist die linke Karte die Serverkarte und rechts die Wahrnehmung der einzelnen Roboter von der Welt.

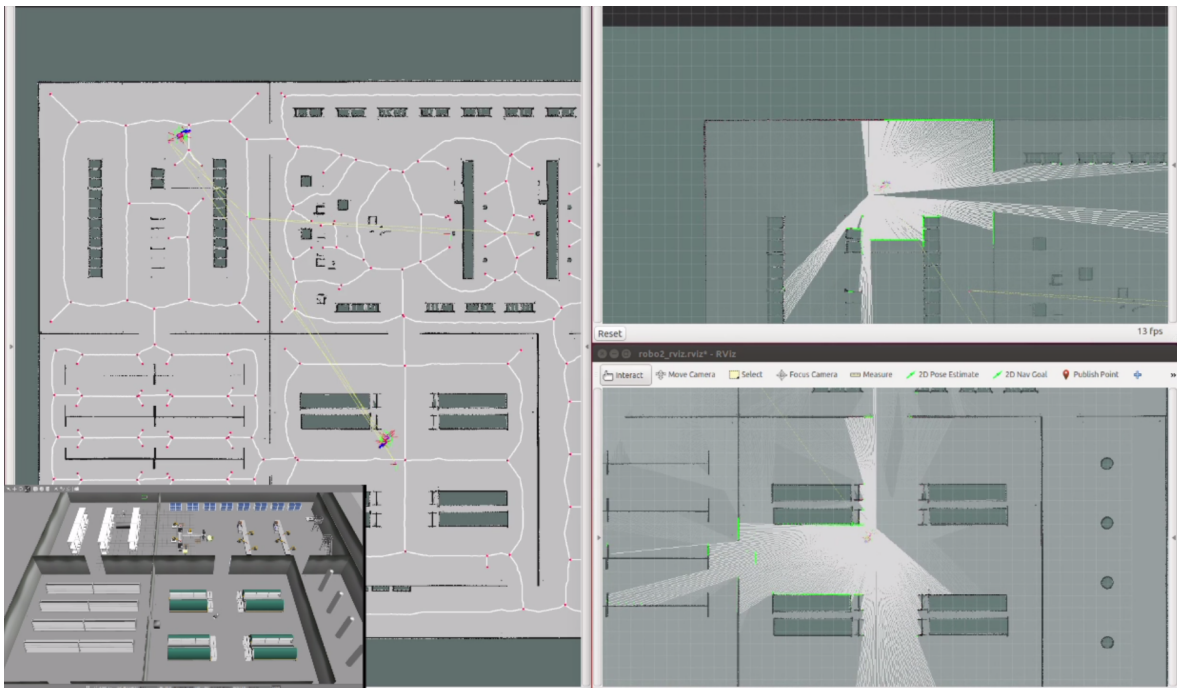


ABBILDUNG 2.2: Simulation des kooperativen Long Term SLAM in Anbindung mit dem Slam Server in der Simulationsumgebung *gazebo*

Dieses SLAM Modul implementiert mit anderen Modulen durch das ROS Framework ein komplettes Softwarepaket (Navigation Stack[2]), mit dem ein Roboter sich eigenständig im Raum bewegen kann. Zu diesen Modulen gehören u.a. die Roboterbeschreibungen, Umgebungskarten, Treiber, Planer, Kollisionsvermeidungsmodule und diverse Filter. Eine grobe Veranschaulichung des Long Term SLAMs sowie ihre Interaktion mit anderen Modulen wird in Abb. 2.3 dargestellt.

[2] wiki.ros.org/navigation

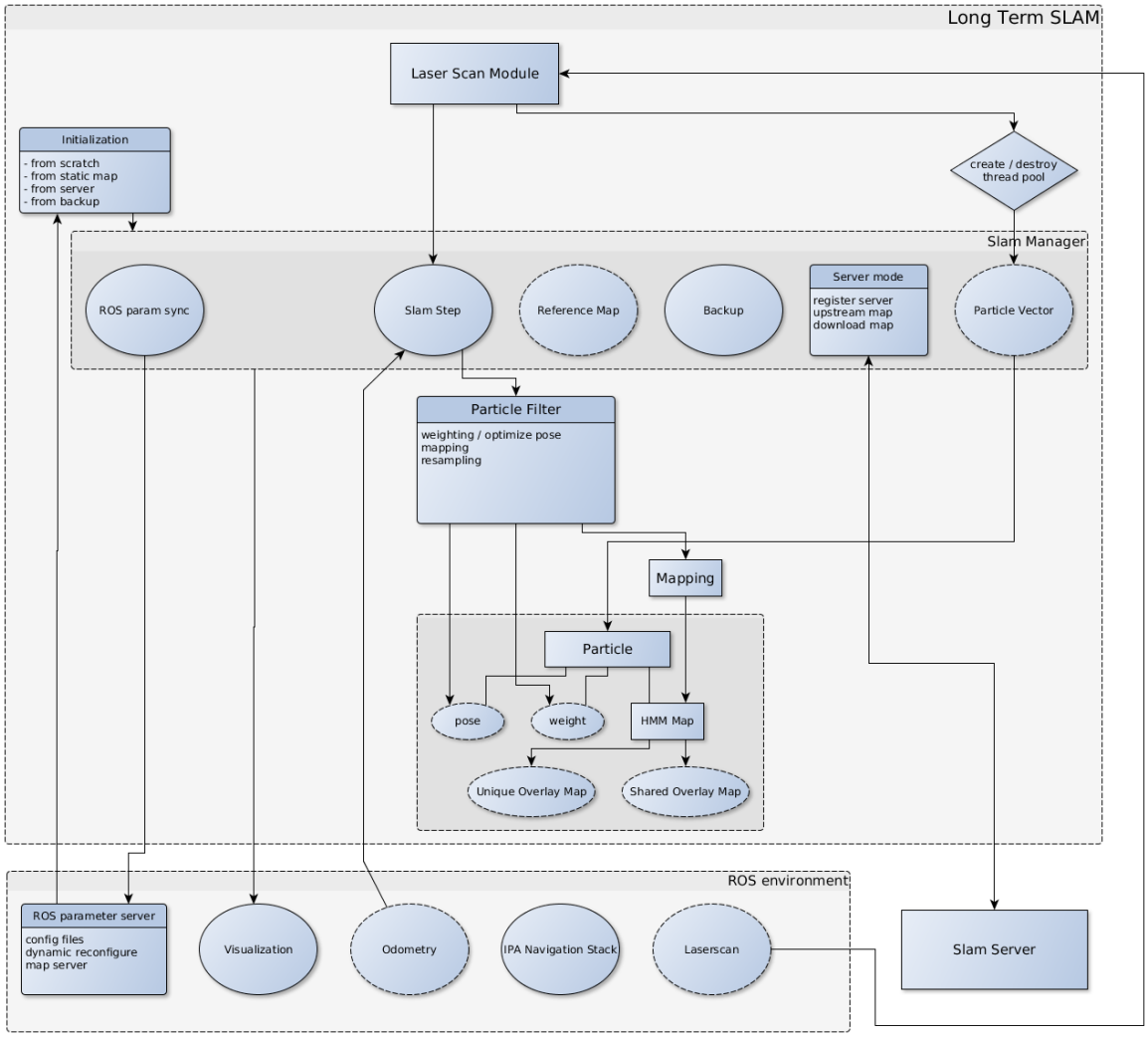


ABBILDUNG 2.3: Struktur des Long Term SLAMs zu Beginn des Praktikums

Kapitel 3

Aufgaben

Dieser Kapitel befasst sich mit den zentralen Aufgaben[1] meines Praktikumaufenthalts. Diese sind im Folgenden thematisch sortiert und näher in ihrer Umsetzung, Funktion und Problematik erläutert.

3.1 Interne NDT Karte und externe Occupancy Gridmap

Während meiner Einarbeitungszeit wurde die Karte von einem anderen Praktikanten erweitert. Die Zellen der Gridmap enthalten in dem neuen Feature sogenannte *Normal Distribution Transforms* (NDT) wie sie in [3] verwendet wurden. Diese Erweiterung ermöglicht eine Repräsentation der durch die Laserscans erfassten Objekte innerhalb einer Zelle, indem die Laserscanpunkte, die in die Zelle einfallen, durch eine Gaußverteilung approximiert werden. Das Matching der Laserscans mit der NDT Karte erfolgt probabilistisch durch ein entsprechendes Sensormodell. Insgesamt mussten für die Einführung dieses neuen *observation models* die Interfaces zu bestehenden Modulen neu überdacht und und implementiert werden. Zum Beispiel ist es dadurch möglich eine wesentlich geringere interne Auflösung von der Karte zu behalten mit gleicher Lokalisierungsgenauigkeit. Dies verbraucht auch weniger Rechenresourcen und Speicherplatz.

Daher folgte logischerweise meine Aufgabe: Die Implementierung des internen / externen Karteninterfaces und die Integration der Änderungen in bereits existierende Module. Dabei soll die interne Karte, die die NDT Zellen enthält, eine geringere Auflösung besitzen als die externe Karte, die sowohl zu Initialisierungszwecken importiert werden kann, als auch für diverse andere Module bereitgestellt werden soll (z.B.: Graphengenerierung für die Pfadplanung).

[1] für eine komplette Liste der im Praktikum durchgeführten Arbeiten wenden Sie sich an den Author

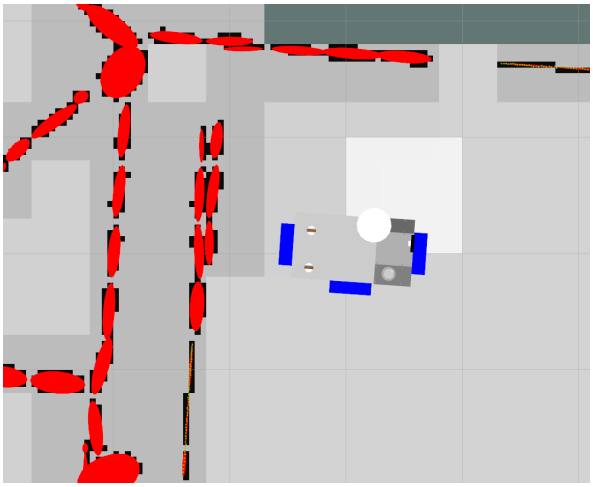


ABBILDUNG 3.1: *rviz* Darstellung der internen und externen Karte

Man kann in Abb. 3.1 das Resultat meiner Aufgabe erkennen. Die Normalverteilungen der größeren Zellen sind als rote Ellipsen dargestellt und darunter ist die gesampelte Rasterkarte in höherer Auflösung zu erkennen.

3.1.1 Initialisierung der internen Karte

Bei der Initialisierung des Long-Term SLAMs über eine statische Karte soll eine hochauflösende *occupancy gridmap*, dessen Zellen entweder belegt, frei oder unerforscht sind, in die interne NDT Karte überführt werden. Zur Vereinfachung werden nur Zellbreiten zugelassen die von den Größen her ein ganzzahliges Vielfaches n der Zellbreite von der Importkarte sind. Dadurch wird eine Zelle genau aus einem Zellcluster mit n^2 Unterzellen generiert. Wichtig ist dabei welchen Zustand die Zelle dabei einnimmt in verschiedenen Situationen.

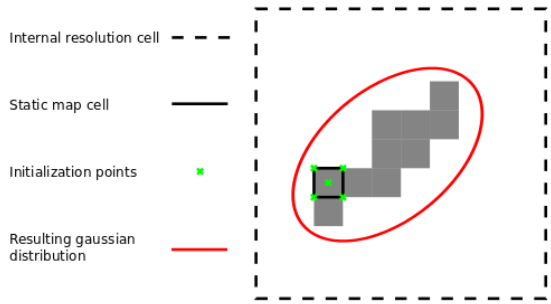


ABBILDUNG 3.2: Initialisierung der Gaußverteilung einer NDT Zelle mit einer externen Karte

Es wurde sich auf die folgende Konvention geeinigt: Falls keine der Unterzellen erforscht ist, soll die innere Zelle ebenfalls unerforscht bleiben. Wenn jedoch mindestens eine Unterzelle als frei markiert wurde und keine belegten Zellen enthält, dann ist die ganze innere Zelle frei. Bei mindestens einer belegten Zelle ist der Zustand der inneren Zelle belegt und wird mit Hilfe der Unterzellen initialisiert. Dabei werden pro Unterzelle fünf Punkte verwendet, um eine Gaußverteilung innerhalb der Zelle zu bilden (siehe Abb. 3.2). Dies hilft Spezialfälle (wie die Gaußverteilungen aus einem Sample) zu vermeiden. Anzumerken ist, dass bei der momentanen Wahl der Punkte (Mittelpunkt und Eckpunkte der Unterzelle) keine komplett gleichmäßige Punktwolke für die Initialisierung verwendet wird, da bei benachbarten Unterzellen Punkte doppelt bzw. vierfach verwendet werden. Trotzdem ist diese Methode hinreichend genau als Approximation.

3.1.2 Möglichkeiten zur Umwandlung von NDT Zellen in binäre Rasterzellen

Die größte Problematik dieses Teilprojekts war die Konvertierung der internen NDT Karte in die hochauflösendere Gridmap zum Exportieren. Dies musste sowohl die Normalverteilungen möglichst genau darstellen, als auch recheneffizient sein, da die Datenmengen, die verarbeitet werden, sich entsprechend der Partikelanzahl und der exportierten Kartenauflösung skalieren. Im optimalen Fall wird die Gaußglocke der Normalverteilung durch zweidimensionale Rechteckfunktionen in Form einer Ellipse dargestellt, dessen Grenzen bei ihrer Standardabweichung (oder einer ihrer Vielfachen) liegen, sodass nach einer Diskretisierung die Ausgangskarte die binären Zustände belegt in und frei außerhalb der Ellipse hat. Da die Verteilung in der Zelle durch die zwei Parameter Mittelpunkt und Kovarianzmatrix dargestellt werden, müssen die belegten Zellen explizit berechnet werden. Dazu wurden mehrere Ansätze theoretisch ausgearbeitet, wobei nur letztere beiden implementiert wurden:

- **Discrete ellipse edge sampling:**
  Bei diesem Ansatz berechnet man direkt aus der achsenparallelen Ellipsengleichung, diskrete Randpunkte einer zentrierten ungedrehten Ellipse. Diese transformiert man mit einer Rotationsmatrix $\mathbf{R}$ in Richtung des ersten Hauptvektors der Kovarianzmatrix und einem Translationsvektor $\mathbf{t}$ in die Zielkoordinaten. Das Problem hierbei ist zum einen eine geeignete Wahl und Implementierung der diskretisierten Punkte, sodass sie sowohl äquidistant auf dem Ellipsenrand liegen, als auch alle Randzellen nach der Transformation im

Zielgitter belegen. Zum anderen ist der Rechenaufwand zu groß, da für jeden einzelnen Punkt die Ellipsengleichung und eine Transformation evaluiert werden muss.

- **Bresenham circle transformation:**
  Aus der Computergrafik ist der Bresenham-algorithmus als einer der effizientesten Algorithmen zum Zeichnen von Geraden auf Rastenanzeigen bekannt. Für diesen Anwendungsfall kann man die Kreisvariante des Algorithmus ensprechend abgeändert werden, d.h. Skalierung vom Kreis in eine achsenparallele Ellipse und dann durch Scherung drehen (siehe [8]). Obwohl diese Variante sehr genau und womöglich eine der effizientesten Methoden ist, wurde sie aufgrund ihrer Implementierungskomplexität nicht verwendet und steht für zukünftige Optimierungen aus.

- **Rhombus edge approximation:**
  Die einfachste aller vorgestellten Methoden ist die Ellipse mit einer Raute anzunähern. Dadurch hat man nur noch vier zu berechnende Eckzellen, die mit vier Kanten verbunden werden. Die Kanten werden mit Bresenham ermittelt und die Eckzellen durch einfache Vektoraddition des Mittelpunktes mit den Eigenvektoren der Kovarianzmatrix. Dazu wurde die *IPA Toolbox* mit einem weiteren Bresenham-algorithmus ergänzt, der die Spezialfälle (wie vertikale Linien) abdeckt, um konsistente Kanten zu erhalten, die zusammen eine geschlossene Raute bilden. Obwohl die Rechenzeit sehr gering ist, muss evaluiert werden, ob die Raute im allgemeinen Fall präzise genug ist um das Objekt darzustellen. Vorallem im Kantenbereich ist der Fehler[2] besonders bei kreisförmigen Ellipsen sehr groß.

- **Wavefront algorithm:**
  Dieser iterative Algorithmus ist die momentan angewandte Methode und lehnt an diversen Algorithmen in der Pfadgenerierung und Pfadplanung aber auch Suchalgorithmen in der Graphentheorie an. Das Prinzip in diesem Fall ist simpel: Man fängt mit einen Punkt an, der auf jeden Fall innerhalb der Ellipse liegt (z.B. der Mittelpunkt), und markiert diesen als belegt. Danach untersucht man die Nachbarzellen, ob sie sich in der ellipsoiden Grenze befinden. Ist dies der Fall wird er markiert und dessen Nachbarn werden ebenfalls untersucht.

[2] Der maximale Fehler zwischen Ellipse und Raute kann in Abhängigkeit ihrer Achsenlängen (in unserem Fall den Eigenwerten) mathematisch hergeleitet werden. Dies überschreitet aber den Rahmen dieser Arbeit.

Natürlich sollte die Suchfront nur nach außen zur Verteilungsgrenze hin propagieren und bei dieser terminieren. Die Praxis hingegen weist einige Schwierigkeiten auf, auf die im nächsten Unterkapitel 3.1.3 eingegangen wird.

3.1.3 Implementierung des Wavefront Algorithmus

Die Fragen, die bei der Implementierung der letzten in Unterkapitel 3.1.2 skizzierten Methode enstehen, lassen sich auf folgende Punkte zusammenfassen:

- Entscheidungsgrenze
- Korrekte Rasterdarstellung der Ellipse
- Umsetzbarkeit
- Effizienz

Ersteres erscheint auf den ersten Blick trivial, da es sich eindeutig, um die Ellipsengleichung handelt. Aber da mit dem zweiten Punkt die Diskretisierung folgt, ist eine geeignete Konvention erforderlich. Idealerweise sind alle Rasterzellen belegt, die von der kontinuierlichen Ellipse „überlappt “ werden, und die restlichen frei. Dies ist aber in der Umsetzung nicht einfach. Außerdem besteht eine Schwierigkeit darin wie man Zellindizes in der Ellipsengleichung. Darüberhinaus ist die Rotation der Ellipse im Raster ein weiteres Problem.

Unter Berücksichtigung dieser Problematiken wurde eine Methode gefunden, die einen Kompromiss aus Genauigkeit und Effizient bildet: Man sampelt diskrete Punkte im kontinuierlichen Raum aus dem verschobenen und rotierten Koordinatensystem der Ellipse. Diese Punkte werden, wenn sie sich innerhalb der Grenzen befinden, in die entsprechende Unterzelle umgewandelt und markiert. Dadurch vereinfacht sich die Abfrage der Abtastpunkte auf die Ellipsengleichung

$$\left(\frac{\hat{x}}{\sqrt{\lambda_0}}\right)^2 + \left(\frac{\hat{y}}{\sqrt{\lambda_1}}\right)^2 \leq 1 \tag{3.1}$$

, wobei $\hat{\mathbf{x}} = (\hat{x}, \hat{y})^T \in \mathbb{R}^2$ die Koordinate der abgetasteten Punkte im transformierten Koordinatensystem $\mathbb{G}_{Ellipse_m}$ der Ellipse m ist und $\lambda_0, \lambda_1 \in \mathbb{R}$ die Eigenwerte der Kovarianzmatrix der Normalverteilung sind. Im Raum $\mathbb{G}_{Ellipse_m}$ ist die Ellipse zentriert und achsenparallel, weswegen Gleichung 3.1 gilt. In Gleichung 3.1 wurde die Ellipsengröße in Abhängigkeit der NDT Zellgröße nicht berücksichtigt. Setzt dies nun nicht mehr voraus und nimmt noch hinzu, dass die Ellipse, die die Normalverteilung visualisiert, ihre Grenze nicht nur bei der Standardabweichung besitzt,

sondern ein Vielfaches c von ihr sein kann, dann ergibt sich Gleichung 3.2, wobei $c \in \mathbb{R}$ ein Skalierungsfaktor ist und ξ unsere neue Grenzbedingung

$$\left(\frac{\hat{x}}{\sqrt{\lambda_0}}\right)^2 + \left(\frac{\hat{y}}{\sqrt{\lambda_1}}\right)^2 \leq c \cdot d_{intern}^2 =: \xi. \tag{3.2}$$

Diese Formel kann in wenigen Schritten hergeleitet werden, indem man die Gleichung mit der korrekten Skalierung der Standardabweichungen $\sqrt{\lambda_0}$ und $\sqrt{\lambda_1}$ umformt. Die Koordinaten, die Gleichung 3.2 erfüllen, werden zunächst danach transformiert

$$\mathbf{x} = \mathbf{R}\hat{\mathbf{x}} + \mathbf{t}. \tag{3.3}$$

$\mathbf{x} \in \mathbb{R}$ ist dabei die Koordinate im Weltkoordinatensystem $\mathbb{G}_{Welt}$. Dabei setzt sich die Rotationsmatrix $\mathbf{R} = (\mathbf{v_0}|\mathbf{v_1})$ aus den Eigenvektoren der Kovarianzmatrix $v_0, v_1 \in \mathbb{R}^2$ zusammen. Der Translationsvektor $\mathbf{t} \in \mathbb{R}^2$ ist hier der Mittelwert der Gaußverteilung. Um daraus die Zellindizes zu generieren, wird entsprechend abgerundet.

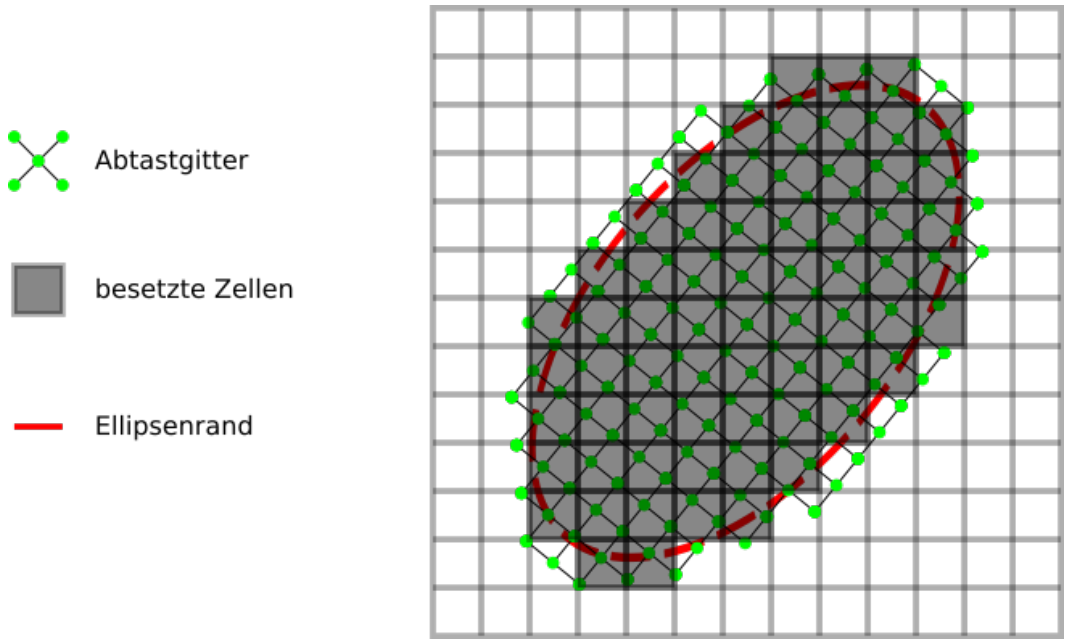


ABBILDUNG 3.3: Beispiel einer Umwandlung von internen NDT Zelle zum externen Raster mit minimaler Abtastgitterabständen

Die Wahl der Abtastschrittweite muss berücksichtigen, dass alle diskreten Zellen mindestens einmal abgetastet werden. Da dies durch die Rotation nicht einfach die Zellbreite der Unterzelle ist, betrachten wir einmal den *worst case*. Der maximale Abstand einer Zelle zu ihren Nachbarn (in der Moore-Nachbarschaft) vom Zellzentrum zu Zellzentrum ist $d_{max} = \sqrt{2} \cdot d$ mit d als Zellbreite. Der minimale Abstand

ist einfach $d_{min} = d$. Im *worst case* ist das Abtastgitter um 45° gegenüber dem Raster der Karte gedreht. Dabei muss minimale Abstand letzteren Rasters größer sein als der maximale Abstand Abtastrasters. Somit muss die Abtastschrittweite $\Delta \leq \frac{d}{\sqrt{2}}$ sein. Da die Anzahl der Samples pro Ellipse für die Performance minimiert werden möchte, sollte Δ möglichst groß gewählt werden. Es wurde sich auf $\Delta = \frac{d}{1.42} \approx \frac{d}{\sqrt{2}}$ geeinigt.

Algorithm 1 Wavefront algorithm

Inputs: Intern cell (Covarianzmatrix Σ, Mean μ, etc.), intern/extern resolution d_{intern}, d_{extern}, sigma factor c

```
1:  procedure GETEXTERNCELLSONELLIPSE( )
2:      calculate ellipse bound ξ = c · d²_intern
3:      add (0, 0) to queue_0
4:      while n < ξ do
5:          while queue_n not empty do
6:              index_cursor ← queue_n.pop()
7:              sample ← transform index_cursor into sampling grid
8:              if sample in nth annulus then
9:                  calculate export grid index from sample
10:                 add grid index and n to extern_cells
11:                 add non seen moore neighbours from index_cursor to queue_n
12:             else
13:                 add non seen moore neighbours from index_cursor to queue_{n+1}
14:             end if
15:         end while
16:         n ← n + 1
17:     end while
18:     return extern_cells
19: end procedure
```

Das Grundprinzip wurde in der echten Implementierung um eine weitere Funktion erweitert wie in Algorithm 1 zu erkennen ist. Diese unterteilt die Abtastpunkte in unterschiedliche Ellipsenringe, welche den gleichen Bereich der Wahrscheinlichkeit in der Gaußverteilung haben. Dadurch hat jede externe Zelle die Eigenschaft $n \in \mathbb{N}$ (siehe Zeile 10). Mit Hilfe von n kann nicht nur zwischen belegt und frei unterschieden werden, sondern man kann diskrete „Belegtheitsstufen“ einführen für eine genauere Beschreibung der Normalverteilung. Diese Funktion wird im Moment noch nicht angewendet, da die externen Module nur binäre *occupancy grids* verwenden.

3.1.4 Integration in andere IPA Module

Die Aufteilung in interne und externe Karte muss in das Interface zu anderen Modulen und diesen selbst übernommen werden. Dazu gehören u.a. das Mapdiff-Interface, der Roadmap-Server und der SLAM-Server. Das Mapdiff-Interface ist die Schnittstelle vom Long-Term SLAM zum Roadmap-Server und ggf. zukünftig anderen Services. In der vorherigen Implementierung wurden nur die Zellen ins ROS Interface *gepublished*, die sich durch neu einfallende Sensormessungen verändern. Dies bietet eine speicher- und recheneffizientere Methode, als die ganze Karte zu versenden. Nun wird die umgewandelte Karte versendet, sowie nur deren Veränderungen. Dazu gehören nun zusätzlich zu der Zustandsänderung der Zelle auch die Formänderung der Ellipse, d.h. wenn die Normalverteilung sich so signifikant verändert, dass sie im feinen Raster bemerkbar sind, zum Beispiel durch Drehung oder Größenänderung etc., dann sollte die Zelle ebenfalls versendet werden, obwohl der Zustand der internen Zelle immer noch belegt ist. Abb. 3.4 zeigt die Anbindung vom Roadmap-Server an den Long-Term-SLAM. Mann kann gut an den „schrägen“ Wänden in der Karte erkennen, dass die auf den Ellipsen basierende exportierte Karte für die Pfadgenerierung verwendet wird.

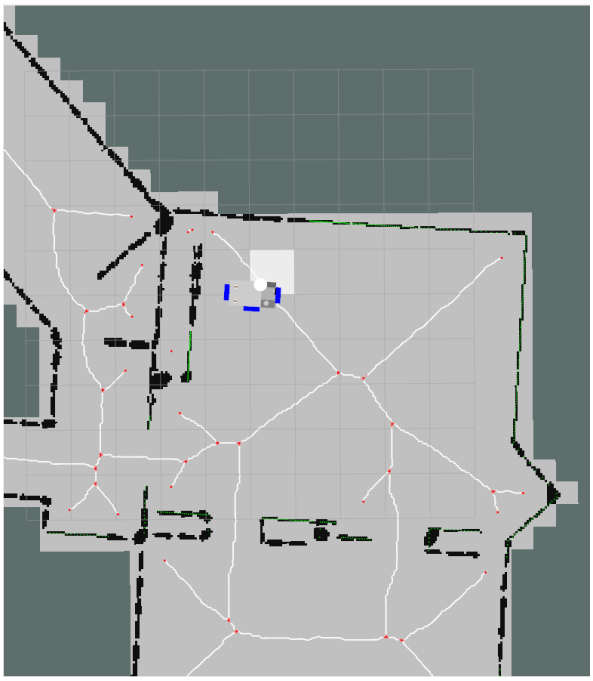


ABBILDUNG 3.4: Exportierte Karte in Anbindung mit *roadmap server*

Analog sind die Änderung für die Kartensynchronisation im kooperativen SLAM mit dem SLAM Server notwendig. Dazu werden aber die NDT Karte verwendet, da sie die „originale“ Karte zur Lokalisierung ist. Die Initialisierung der Server Karte durch vorgefertigte Bitmaps erfolgt analog zu Unterkapitel 3.1.1. Dennoch muss

die Synchronisationsmethodik der Klientkarten überdacht werden. In der vorherigen Version wurden die Zellen, die in Konflikt zueinander stehen, synchronisiert, indem die Zeitstempel verglichen wurden. Der Wert mit der aktuelleren Zelle wurde dafür verwendet. Dies ist auch in diesem Fall möglich, dennoch ist dies keine gute Umsetzung einer Kartenfusion und es ist unklar, ob dadurch genauere Kartierungen enstehen. Die detailliertere Untersuchung des Problems fiel aber nicht in meinen Aufgabenbereich.

3.2 Modulare Integration mehrerer Sensoren im SLAM

Es ist üblich bei autonomen Fahrzeugen wie mobilen Robotern oder selbstfahrende Vehikeln LiDAR Sensoren (*light detection and ranging*) einzusetzen, da diese eine hohe Messgenauigkeit bei guten Umgebungsbedingungen besitzen. Dadurch kann eine hohe Lokalisierungsgenauigkeit und präzise Kartierungen erreicht werden. Aus diesem Grund bauen die meisten SLAM Algorithmen mit einer *occupancy gridmap* auf LiDAR Sensoren auf, u.a. *gmapping* von ROS oder auch hier der Long-Term SLAM. Natürlich ist es möglich andere Sensoren wie RaDAR Sensoren (*radio detection and ranging*), Ultraschallsensoren, Kameras (mono/stereo, 3D), Infrarotsensoren etc. in den Algorithmus zu integrieren. Auch indirekte Informationen wie WLAN Latenzzeiten zu mehreren Routern können dazu verwendet werden mehr Informationen aus der Umgebung zu gewinnen.

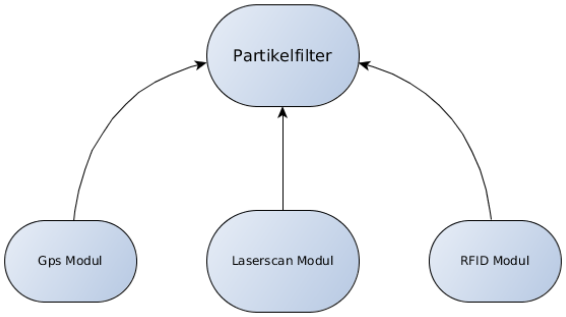


ABBILDUNG 3.5: Modulare Struktur der Sensoranbindungen an den Long-Term-SLAM

Mit diesem Hintergrund folgt meine nächste größere Aufgabe. Dazu soll der Long-Term SLAM so umstrukturiert werden, dass man mehrere Sensoren verwenden kann. Darunter gehören Modifikationen am Sensorinterface und an der Implementierung des Partikelfilters.

3.2.1 Umstrukturierung der Sensormodule

Der vorherige Aufbau war, dass es eine globale Klasse `LaserScanModule` gab, bei dem eine Callback-Funktion getriggert wurde, sobald eine Sensormessung vorliegt. Diese wurde verarbeitet, indem an den `SlamManager` eine Anforderung (*slam request*) für einen SLAM Schritt gesendet wird. Der `SlamManager` antwortet dann mit einem Odometrieschritt der existierenden Partikel. Daraufhin werden so viele Threads gestartet wie Partikel vorhanden sind, die die Partikel in ihrer Pose optimieren und gewichten und danach die Messung in der Karte übernehmen. Schließlich werden die Partikel nach [7] selektiv geresampelt.

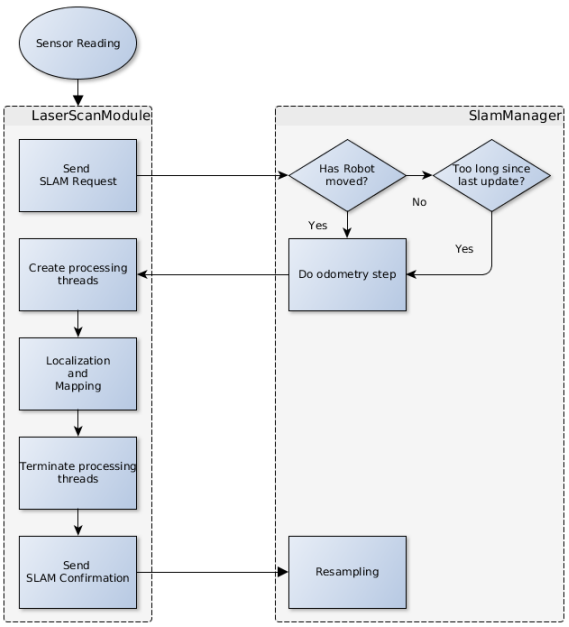


ABBILDUNG 3.6: Flussdiagramm der Prozessfolge bei einer Laserscanmessung

Abb. 3.6 zeigt nochmal wie die einzelnen Prozesse unter den Funktionsklassen aufgeteilt war. Um nun weitere Sensoren einzubinden und redundante Codefragemente zu vermeiden, bietet es sich an eine `SensorModule` Oberklasse einzuführen, die gemeinsame Funktionalitäten implementiert. Darunter gehören neben den generellen Verwaltungsfunktionen wie Sensoranbindung zum `SlamManager` auch die Posenoptimierung, die unabhängig von den Sensoreigenschaften ist. Die Unterklassen enthalten demnach die für jeden Sensor spezifischen Funktionen, wie zum Beispiel die Lokalisierung der Partikel über die Gewichtungsfunktionen und Kartierungsfunktionen. Es ändert sich dadurch in der Prozessabfolge nichts, dennoch hat die Klassenstruktur nun modulare Eigenschaften. Somit muss man nur noch die Gewichtungs- und Mappingfunktionen in einer neuen Unterklasse definieren, um ein Interface zu neuen Sensorfunktionalitäten aufzubauen, welche dann

vom `SlamManager` verwendet werden können. Ein Abhängigkeitsbaum dazu ist in Abb. 3.7 dargestellt.

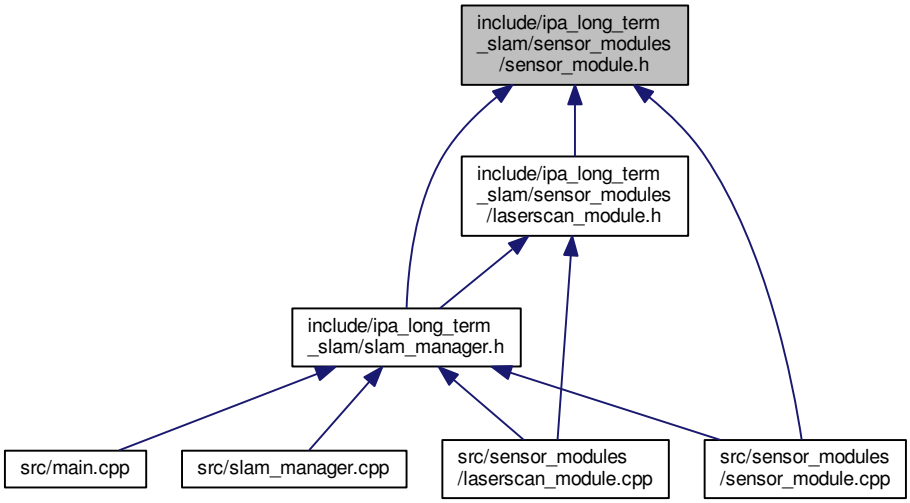


ABBILDUNG 3.7: Abhängigkeitsbaum von `SlamManager` zu `SensorModule` und `LaserScanModule` aus dem Dokumentationswerkzeug *Doxygen*

3.2.2 Strukturanpassung des Partikelfilters

Nun da die Grundstruktur gegeben ist, muss entschieden werden wie man die verschiedenen Daten synchronisiert. Bei einem einzigen Sensor ist die Abarbeitungsfolge simpel: es werden direkt die Daten verarbeitet, die als erstes ankommen. Dies ist schon durch die ROS Messagingstruktur (FIFO Pipeline / *message dropping*) garantiert und der Datenpuffer kann sequentiell abgearbeitet werden. Da bei mehreren Sensoren die Datenpakete nun asynchron an unterschiedlichen ROS Schnittstellen ankommen, ist eine zeitliche Sortierung der Datenpakete nicht mehr garantiert. Dieses Problem kann durch die folgenden Synchronisationsmethoden gelöst werden:

- Man definiert sich einen gemeinsamen Puffer, indem die Datenpakete nach dem Zeitstempel sortiert werden. Dieser Puffer wird dann sequentiell abgearbeitet. Der Nachteil dieser Methode ist, dass die gespeicherten Pakete zusätzlich Speicherplatz einnehmen. Bei zu vielen Daten kann es vorkommen, dass diese nicht mehr rechtzeitig abgearbeitet werden können. Dies kann analog zu ROS verhindert werden, indem die Pufferlänge begrenzt wird.

- Die schließlich angewendete Methode ist sehr simpel, aber auch effizient. Man aktualisiert nach jedem SLAM Schritt die aktuelle Zeit und alle Pakete, die älter sind, werden verworfen. Dabei ist die Verarbeitung der Pakete auch asynchron. Dennoch verliert man viele Informationen durch diese Methode (Im Nachhinein wurde dies durch meinen Betreuer erweitert, indem die vergangenen Daten nicht mehr verworfen werden. Dessen Partikelpose werden in der Zeit zurückverschoben. Anschließend führt man die Lokalisierungs- und Mappingschritte aus und verschiebt sie zurück in die aktuelle Zeit. Für nähere Informationen kontaktieren Sie Stefan Dörr).

3.2.3 Normalisierung der Partikelgewichte

Eine weitere Modifikation am Algorithmus, die durchgeführt werden musste, war die Normalisierung der Gewichte in der Lokalisierungsphase. Aus einem mir unbekannten Grund wurde dies in der vorherigen Implementierung nicht vollständig umgesetzt. Das Ziel der Normalisierung ist es, dass jedes Datenpaket einer Sensormessung unabhängig vom Sensortyp den gleichen Einfluss auf die Pose der Partikel hat, wenn die Umstände gleich sind. In der Literatur wird per Konvention die Summe aller Partikelgewichte auf eins normiert (siehe 3.4, $N_{Partikel}$ ist die Anzahl der Partikel).

$$\sum_{i=1}^{N_{Partikel}} w_i = 1 \tag{3.4}$$

Dadurch erkennt ein Partikel wie gut seine Pose relativ zu anderen Partikeln ist. Dabei geht aber verloren wie gut bzw. schlecht die Messung mit der Karte im Allgemeinen übereinstimmt. Diese werden beim selektiven Resampling aufsummiert bis dieser durchgeführt wird. Nach diesem Schritt sind die Gewichte implizit in der diskreten Verteilung der Partikel enthalten. Diese repräsentiert nun die neue Zustandsraumverteilung. Dabei macht es keinen Sinn die Gewichte zu behalten. Deswegen müssen diese unter Berücksichtigung von Glg. 3.4 zurückgesetzt werden. Man sich auf

$$w_i = \frac{1}{N_{Partikel}} \qquad \forall i \in [1, N_{Partikel}] \subseteq \mathbb{N} \tag{3.5}$$

geeinigt.

Die Kalkulation der Gewichte selbst vor dem Normierungsschritt in Glg. 3.4 wurde ebenfalls geringfügig überarbeitet. Da wir nun zwei *observation models* durch die Einführung der NDT Karte haben, muss in zwei Fälle unterschieden werden.

Beim normalen binären Belegungraster wird nun

$$w = \sum_{k=1}^{N_{Laser}} \exp\left(-\frac{1}{2\sigma}\delta_i\right) \tag{3.6}$$

verwendet. δ_i ist die kleinste Distanz zu der nächsten belegten Zelle, N_{Laser} ist die Anzahl der Laserscanpunkte und σ ist ein Skalierungsfaktor. Die Glg. 3.6 garantiert, dass das Gewicht bei niedrigen Distanzen δ_i groß wird und bei großen Distanzen das Gewicht $w \to 0$ konvergiert. Für die NDT Karte wird die vom Praktikanten, der die NDTs implementiert hat, eingeführte Variante verwendet (siehe Glg. 3.7). Dabei werden aus den Laserscanpunkten Gaußverteilungen gebildet, die mit den schon in den Zellen vorhandenen Verteilungen verglichen werden. Dabei notiert $N_{Container}$ die Anzahl der Zellen, in denen die Laserpunkte fallen und μ, $\mathbf{\Sigma}$ die entsprechenden Mittelwerte und Kovarianzmatrixen der Verteilungen.

$$w = \sum_{k=1}^{N_{Container}} \exp\left(-\frac{1}{2}(\mu_{cell} - \mu_k)^T(\mathbf{\Sigma}_{cell} + \mathbf{\Sigma}_k)^{-1}(\mu_{cell} - \mu_k)\right) \tag{3.7}$$

Diese Gleichung lehnt an der Mahalanobis Distanz einer multivariaten Verteilung an. Es wurde eine Alternative dazu implementiert, die die Ähnlichkeit der beiden Verteilungen über die Kullback-Leibler Divergenz berechnet. Für zwei bivariate Normalverteilungen kann sie zu

$$\begin{aligned} D_{KL}(P_1||P_2) &= \int_{-\infty}^{\infty} p_1(x) \log \frac{p_1(x)}{p_2(x)} dx \\ &= \frac{1}{2}\left(\log \frac{\det \mathbf{\Sigma}_2}{\det \mathbf{\Sigma}_1} - 2 + \mathrm{tr}(\mathbf{\Sigma}_2^{-1}\mathbf{\Sigma}_1) + (\mu_2 - \mu_1)^T \mathbf{\Sigma}_2^{-1}(\mu_2 - \mu_1)\right) \end{aligned} \tag{3.8}$$

hergeleitet werden und daraus ergeben sich für die Partikelgewichte:

$$w = \sum_{k=1}^{N_{Container}} \exp\left(-D_{KL,k}\right). \tag{3.9}$$

Die Evaluation beider Methoden wurde aus Zeitgründen noch nicht durchgeführt. Dennoch lässt sich aus der Anzahl der Rechenoperationen schließen, dass Glg. 3.9 mehr Zeit in Anspruch nimmt als Glg. 3.7. Der Einfluss der Kovarianzmatrizen wird unterschiedlich behandelt und inwiefern dies zur Genauigkeit der Lokalisierung beiträgt bleibt leider offen.

3.2.4 Anwendungsbeispiel: GPS

Im Zuge dieser Vorbereitungen wurde ein neues Sensormodul `GpsModule` (später `PoseMeasurementModule`) erstellt. Dieses Modul verarbeitet eingehende GPS Daten in Kooperation mit dem `LaserScanModule`. Zum Testen des Aufbaus wurden Sensoraufzeichnungen von einen am IPA modifizierten BMW i3 verwendet. Dieser hat auf einer Teststrecke Odometriedaten sowie Laserscan- und GPS- Messungen durchgeführt und mit Hilfe von ROS-Tools aufgenommen (*rosbag*). Diese Aufzeichnungsdateien (*bagfiles*) können wiedergegeben werden, um so in Anbindung mit dem Long-Term SLAM eine Lokalisierung und Kartierung durchzuführen.

ABBILDUNG 3.8: SLAM in der Simulation mit den *bagfiles* des IPA BMW i3

Man geht hier von einer GPS Messung aus, die mit einer mittelwertfreien Geußverteilung verrauscht ist. Die Gewichtung der Partikel erfolgt dementsprechend, d.h.

$$w = exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}_{mess})^T\Sigma^{-1}(\mathbf{x}-\mathbf{x}_{mess})\right) \tag{3.10}$$

, wobei $\mathbf{x}$ die Partikelposition und $\mathbf{x}_{mess}$, Σ die GPS Messung und ihr Rauschen sind.

Da die Laserscanmessungen durch die Umgebung (Gras etc.) sehr verrauscht sind, wäre die Lokalisierungsgenauigkeit gering und demnach die Kartierung relativ unpräzise. Mit den GPS Koordinaten jedoch kann sie verbessert werden. Dies erkennt man in Abb. 3.8, wobei der gelbe Pfeil die verrausche GPS Messung ist. Daher ist dieses Modul vorallem im Außeneinsatz interessant, da die GPS Daten

weiträumig zur Verfügung stehen und relativ präzise einen Schätzwert für die absolute Position liefert. Damit kann u.a. die Initialisierung des Standorts übernommen werden und dadurch vereinfacht sich das globale Lokalisierungsproblem enorm.

3.2.5 Anwendungsbeispiel: RFID

Es wurde ebenfalls eine Anbindung zu RFID-Sensordaten erstellt. Das Sensormodul `RfidModule` soll bei fahrerlosen Transportfahrzeugen (FTF), bei denen RFID Sensoren an der Unterseite angebracht sind, zum Einsatz kommen. Dabei erkennen die Sensoren im Boden angebrachte Marker. In diesem Testbeispiel wurden nur Positionsangaben hinsichtlich einer Koordinatenachse in der Sensorfläche zurückgegeben. Daher geht man von einer Gaussverteilung in einen und eine Rechteckverteilung in der anderen Achse aus (siehe Abb. 3.9).

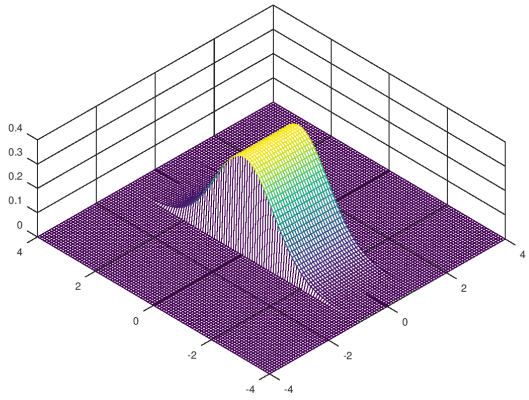


ABBILDUNG 3.9: Wahrscheinlichkeitsverteilung der Sensormessung beim RFID Sensor

Auch hier liefert der Sensor nur Lokalisierungsinformationen. Im Gegensatz zum GPS-Sensor benötigt man hier eine statische Karte der Landmarken. Natürlich wäre es möglich die Position der Landmarken während der Laufzeit in eine leere Karte kartieren. Der Zweck des Sensormoduls ist es aber zusätzliche Informationen der Umgebung (in diesem Fall die Karte der Landmarken) sinnvoll zu verwerten und dadurch die Lokalisierung zu verbessern.

Dieses Modul ist insbesondere für Präzisionsbewegungen interessant, da aufgrund der Sensorgenauigkeit die Position des Roboters innerhalb der möglichen Messfläche sehr genau ist. In unserem Beispiel benötigt man mindestens drei RFID Sensoren um eine eindeutige Pose im Raum bestimmen zu können. Anwendungen finden sich daher im industriellen Bereich z.B. in der Manufaktur, wo die Umgebung größtenteils deterministisch ist.

3.3 KLD-Sampling

Bisher haben wir beim Partikelfilter eine statische Anzahl an Partikel, d.h. jedes Mal hält der Algorithmus $N_{Partikel}$ Hypothesen von seiner Pose und Karte. Beim Resampling Schritt werden zwar Partikel dubliziert oder zerstört, aber in der Gesamtzahl bleiben sie gleich. Die Effizienz des Algorithmus kann aber erheblich gesteigert werden, in der Sampleset sich adaptiv verändert. D.h. wenn das Vertrauen in die Partikel gering ist, sollen mehr Hypothesen aufgestellt werden, und bei einer hohen Partikeldichte soll die Anzahl verringert werden. Basierend auf [4] wurde in ROS schon das *amcl* Package (Adaptive Monte Carlo Localization) implementiert. Diese Methode geht von einer bekannten Karte aus und addressiert auch das globale Lokalisierungsproblem.

3.3.1 Implementierung

In diesem Projekt soll nur das adaptive Verhalten der Partikelanzahl umgesetzt werden. Es dient auch hier [4] wieder als Grundlage. Die Idee ist es über die Kullback-Leibler Divergenz (daher auch KLD-Sampling) zwischen der echten a-posteriori-Wahrscheinlichkeit[3] vom Roboterzustand $p(x_t|x_{t-1}, u_{t-1})$ und die durch die Samples repräsentierte *maximum likelihood estimate* (MLE) eine Genauigkeit zu garantieren. Dabei definiert man einen Grenzwert für die KLD und daraus kann man unter Annahme bestimmter Eigenschaften, die im Verlauf des Berichts näher erläutert werden, die benötigte Anzahl an Hypothesen bestimmen für dies der Fall ist:

$$n \doteq \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right\}^3 \tag{3.11}$$

Dazu benötigt man den Grenzwert des Fehlers ϵ von der echten Verteilung zur MLE, das obere $1-\delta$ Quantil[4] $z_{1-\delta}$ der Standardnormalverteilung $\mathcal{N}(0,1)$ und die Anzahl der Träger k von der MLE. ϵ und $z_{1-\delta}$ können vordefiniert werden. k hingegen wird im Algorithmus iterativ bestimmt. Die Größe der Träger muss ebenfalls vordefiniert werden. Nähere theoretische Hintergründe und die Herleitung von Gleichung 3.11 können in [4] nachgelesen werden.

In der vorherigen Implementierung des Resamplingschrittes wurden die Partikel durch Umsortieren der Elemente innerhalb des STL-Kontainers, in dem die Partikel gespeichert sind, gesampelt (siehe Abb. 3.10). Der Speicheroverhead der dabei

[3] x_t notiert den Zustand des Roboters und u_t den Kontrolleingang (hier: Odometrie) zum Zeitpunkt t

[4] δ ist die Wahrscheinlichkeit, dass der Fehler unter ϵ ist und ein Quantil ist ein Lagemaß in der Statistik und kann als Schwellenwert interpretiert werden,

entsteht ist der Vektor, der die Anzahl der zu kopierenden Partikel zählt. Dieser ist aber dadurch, dass es nur ein Vektor aus Integern ist, in jedem Fall um einige Größenordnungen kleiner als der Partikelvektor. Dadurch garantiert man, dass der Speicheraufwand des Samplingschritts gering bleibt. Aus der Implementierung des KLD-Sampling-Algorithmus aber folgt zwangsläufig, dass sich der Partikelvektor nach jedem Resamplingschritt sich entweder vergrößern oder verkleinern kann. Da der neu generierte Vektor auf ihrem letzten Zustand basiert, ist das Umsortieren nicht ohne weiteres möglich.

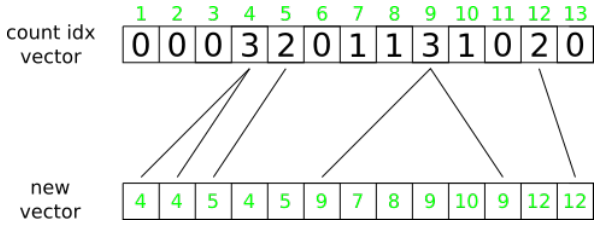


ABBILDUNG 3.10: Resampling der Partikel durch Umsortierung des Partikelvektors bei statischer Partikelanzahl

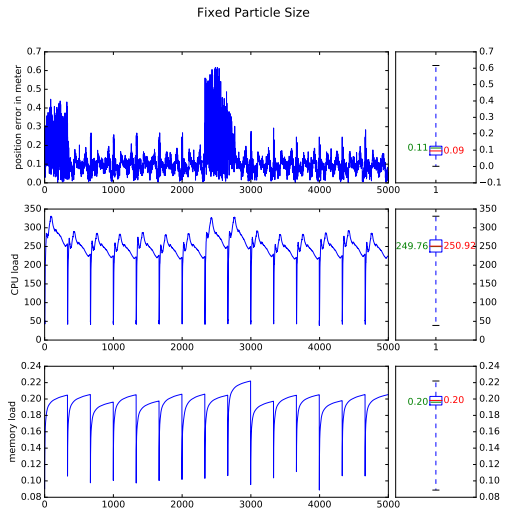
Daher ist ein weiterer Schritt notwendig um diese Methode beizubehalten. In diesem Schritt muss garantiert werden, dass die benötigten Partikel erhalten bleiben bevor der Vektor „schrumpft“. Die gesampelten Indizes müssen auch entsprechend angepasst werden. Hierbei wurden die Partikel, die nicht im neuen Vektor vorkommen, gelöscht, sodass die notwendigen Partikel an den Anfang des Vektors sortiert werden.

Ein anderer naiver Ansatz ist den alten Zustand in eine Zwischenvariable zu kopieren und um damit den neuen Vektor zu generieren. Dies skaliert aber mit der Größe des Partikelvektors und der Größe eines Partikels. Momentan wird diese Methode verwendet, da kein zusätzlicher Sortierschritt und somit keine zusätzliche Rechenleistung benötigt wird.

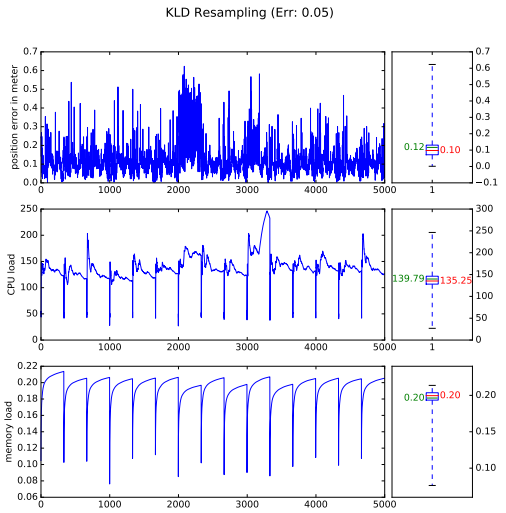
3.3.2 Testergebnisse

Um die tatsächliche Leistungsverbessrung evaluieren zu können, wurde mehrere Testreihen durchgeführt. Dabei müssen sowohl die optimalen Parametergrößen für die aktuelle Konfiguration des Long-Term-SLAMs ermittelt werden, als auch die Unterschiede zu der statischen Samplingstruktur erkennbar sein.

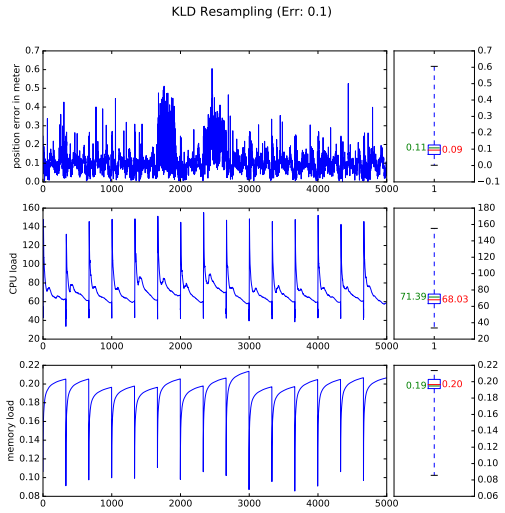
Da die Größe der Träger einen sehr großen Einfluss auf die Änderung der Partikelanzahl hat, wurde sie so eingestellt, dass die Anzahl zwischen 10 und 500 schwankt. Dies ist bei der Zellgröße der externen Karte und einem Winkelschritt von $\pi/6$ rad der Fall. Der Parameter ϵ wurde in der Menge $\{0.05, 0.1, 0.2\}$ evaluiert.



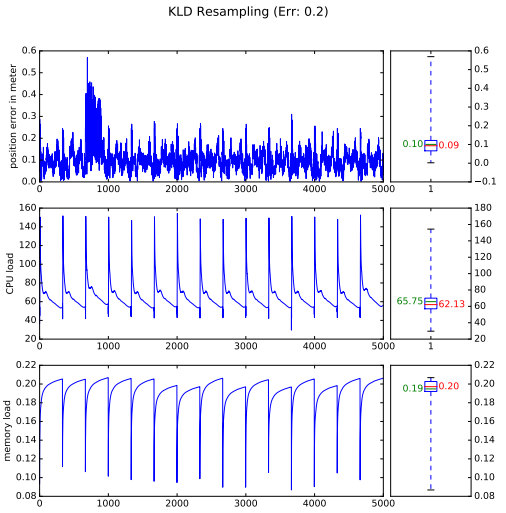
(A) Referenzmessung



(B) KLD-Resampling (ϵ: 0.05)



(C) KLD-Resampling (ϵ: 0.1)



(D) KLD-Resampling (ϵ: 0.2)

ABBILDUNG 3.11: Simulationsergebnisse der Roboterlokalisierung

Es wurden aufgenommene Sensormessdaten für den Lokalisierungsalgorithmus verwendet und 15 Wiederholungen pro Konfiguration durchgeführt, um konsistente und vergleichbare Ergebnisse zu generieren. Die in Abb. 3.11 dargestellten Graphen wurden bei einer Testfahrt in der simulierten IPA Fabrikhalle aufgenommen. Dabei wird die geschätzte Position zu jedem Zeitpunkt mit der echten Position verglichen. Zusätzlich wurde die Rechen- und Speicherauslastung gemessen, um die Performance beider Algorithmen zu vergleichen.

Man erkennt, dass beim KLD-Resampling weniger Rechenkapazität notwendig sind. Die Speichernutzung ist hingegen gleichmäßig geblieben. Der Lokalisierungsfehler ist bei allen vier Konfigurationen in etwa gleich. nur bei 3.11b ist der durchschnittliche Fehler geringfügig größer. Eigentlich erwartet man bei geringem ϵ kleine Lokalisierungsfehler, da die Wahrscheinlichkeitsverteilung der Partikelwolke gut repräsentiert wird. Dies ist hier möglicherweise durch andere Effekte wie häufige Änderung der Partikelanzahl nicht der Fall. Ansonsten bemerkt man ein häufigeres Auftreten von Spikes und größere Ausschläge in den Lokalisierungsfehlern der KLD-Samplingdurchläufen, die mit größerem ϵ abnehmen. Ein möglicher Grund dafür wäre, dass die Ausschläge genau dann gemessen werden, wenn die Dichte der Partikel gering ist und daraufhin mehr generiert werden.

Kapitel 4

Eindrücke und Ausblick

Da mein Aufgabenbereich sich nach dem gerichtet hat, was die höchste Priorität hatte bzw. welche Aufgaben in dem Projekt zu jenem Zeitpunkt entstanden, war mein Einsatzgebiet sehr weit gefächert. Dadurch konnte ich ein gutes Bild vom Gesamtprojekt des Long-Term SLAMs entwickeln und konnte meine Fähigkeiten in teilweise sehr unterschiedlichen Bereichen weiterentwickeln.

Meine vorherige Arbeitserfahrung im Bereich der Regelungstechnik und Embedded Systems, deswegen ist die praktische Arbeit an mobilen Robotern neu für mich. Entsprechendes theoretisches Hintergrundwissen, die nicht in meinen Vorlesungsreihen behandelt wurden, musste ich mir vor dem Praktikumszeitraum sowie während der Einarbeitungszeit aneignen. Darunter gehört u.a. die grundlegenden Themen Bayesfilter, Partikelfilter, Mapstrukturen sowie *motion models*. Dahingegen fiel mir die Einarbeitung in die programmiertechnischen Tools wie *git*, ROS und Linux leichter. Die meiste Zeit nahm aber das Verständnis vom Long-Term SLAM Projekt in Anspruch, da sie eine sehr komplexe Struktur hat und aufgrund ihres Entwicklungsstadiums nicht detailliert dokumentiert war.

Die Arbeit selbst am Projekt ist sehr wertvoll für mich, da sie mir Erfahrungen vermittelt hat, die über den Stoff der eigentlichen Aufgabe hinausgehen. Zum Beispiel ist mir bei den Nebenaufgaben an der Multithreadingstruktur klar geworden wie wichtig eine gründliche Vorbereitung, Durchführung und Dokumentation von Konzepten ist, damit sie am Ende funktioniert und für Dritte verständlich ist. Desweiteren konnte ich erste Erfahrungen im Programmierworkflow sammeln. Dazu gehört neben der Implementierung der Funktionalität u.a. das stetige Optimieren der Performance, Debugging und Testen der Software in Simulationsumgebungen. Dabei sind Tools wie gdb und *Automatic Testing Frameworks* (ATF) sehr nützlich. Ansonsten sind Shellskripte in Kombination mit Pythonskripts auch sehr nützlich, wenn man redundante Aufgabe automatisieren möchte.

Von der Thematik her habe ich nun ein sehr tiefes Verständnis von Lokalisierungs und Mappingalgorithmen bekommen, welche mir vorher nicht bekannt waren. Darüberhinaus habe ich auch Einblicke in die Computergrafik bekommen, die

ich in diesem Themengebiet nicht erwartet hätte. Der Sinn von mathematischen Grundvorlesungen im Bachelorstudium sowie die theoretische Informationstechnik ist mir vorher nicht so bewusst geworden als wie im Praktikum.

Zusammenfassend kann ich sagen, dass dieses Praktikum mir einen wichtigen Einblick in die reale Arbeitswelt gezeigt hat, mit welcher man seine Karriere neu überdenken kann. Dieser Erfahrungsschatz geht über dem thematischen Wissen hinaus und hilft beim evaluieren neuer Möglichkeiten.

Anhang A

Tägliche Aufzählung ausgeführter Arbeiten

Dieser Anhang listet die im Praktikum durchgeführten Arbeiten stichpunktartig in Tagesabschnitte zusammengefasst auf. Das Pflichtpraktikum wurde aufgrund eines Auslandsemesters vom 01.03.17 - 05.07.17 in zwei Teile aufgeteilt, dessen Zeitraum jeweils länger als ein Monat beträgt.

A.1 Praktikum Zeitraum 1 (01.11.16-28.02.17)

01.11.16	Organisatorisches
02.11.16	CodingIntroduction: ROS programming style, C++ programming style
03.11.16	CodingIntroduction: IPA style guide, IPA modules and structure
04.11.16	SLAM/ROS: What is SLAM?, Rao-Blackwell Particle Filter, HMM-Cell Map
07.11.16	SLAM/ROS: ROS Messaging Structure, ROS Navigation Stack, ROS Master
08.11.16	SLAM/ROS: IPA LTS SLAM, IPA Navigation Sandbox, Current Issues
09.11.16	Offload hard-coded timeouts to ROS Parameters
10.11.16	Publish feature IDs
11.11.16	Sensor Modules: Konzept: Wie umstrukturieren -> top/low level functionality class

14.11.16	Sensor Modules: Implementierung: Sensor Modules/Laserscan Module classes
15.11.16	Sensor Modules: Debugging in der Klassenstruktur
16.11.16	Sensor Modules: Debugging von zusätzlich aufgespaltenen Funktionen
17.11.16	CLTS-Video: Konzeptentwicklung, Szenarioentwurf
18.11.16	CLTS-Video: Modulintegration Cooperative LTS
21.11.16	CLTS-Video: Modulintegration SLAM Server
22.11.16	CLTS-Video: Debugging
23.11.16	CLTS-Video: Simulation des SLAM Szenarios
24.11.16	CLTS-Video: bagfile Aufnahme des Szenarios
25.11.16	Szenarioüberarbeitung durch Feedback vom Betreuer
28.11.16	CLTS-Video: Simulation des neuen Szenarios
29.11.16	CLTS-Video: Multiple Access Bug in HMM Cell
30.11.16	Szenarioüberarbeitung durch Feedback vom Betreuer
01.12.16	CLTS-Video: Finale Simulation
02.12.16	CLTS-Video: Recherche Videobearbeitung
05.12.16	CLTS-Video für CLONI-Projektbesuch: Videobearbeitung
06.12.16	„interne“ Resolution: Konzeptionierung, Vorgehensweisen, Stand der Technik
07.12.16	„interne“ Resolution: Recherche Gridalgorithmen, Computergrafik, Bildverarbeitung
08.12.16	„interne“ Resolution: Recherche Umwandlung Rasterkarten in kontinuierliche Karten
09.12.16	„interne“ Resolution: Prototypenausarbeitung -> Rhombus und Wavefront Algorithmus
12.12.16	„interne“ Resolution: Umsetzung vom Rautenalgorithmus, initiale Performancetests
13.12.16	„interne“ Resolution: Umsetzung vom Wavefrontalgorithmus
14.12.16	„interne“ Resolution: Anpassungen am Map Modul, Server Modul
15.12.16	„interne“ Resolution: Anpassungen am Roadmap Modul, Austausch mit anderen Praktikanten
16.12.16	„interne“ Resolution: Debugging: Fehlende Getter Funktionen

19.12.16	„interne“ Resolution: Debugging: Linker Error
20.12.16	Multiple Access Bug: Analyse vom Ursprung des Fehlers in der Execution Chain
21.12.16	Multiple Access Bug: Try and Error Vorgehen zur Implementierung eines Quick Fix
22.12.16	Multiple Access Bug: Recherche und in-depth Analyse zur robusten Fehlerbeseitigung
23.12.16	Multiple Access Bug: Teamwork zur Beseitigung des Problems, neue Aufgabeneinteilung
26.12.16	Urlaub/Feiertag/Überstunden
27.12.16	Urlaub/Feiertag/Überstunden
28.12.16	Urlaub/Feiertag/Überstunden
29.12.16	Urlaub/Feiertag/Überstunden
30.12.16	Urlaub/Feiertag/Überstunden
02.01.17	Urlaub/Feiertag/Überstunden
03.01.17	„interne“ Resolution: Auffrischung BFS, DFS, A* etc., rigide Transformation
04.01.17	„interne“ Resolution: Prototypenausarbeitung scaled Transform
05.01.17	„interne“ Resolution: Prototypenausarbeitung sheared Ellipse
06.01.17	„interne“ Resolution: Implementierungsansätze der ausgearbeiteten Prototypen
09.01.17	„interne“ Resolution: Scaled Transform erweist sich als zu komplex zum Umsetzen
10.01.17	„interne“ Resolution: Sheared Ellipse Tutorials in Python Recherche
11.01.17	„interne“ Resolution: Suche von Analogi in C++ Welt zum Referenzalgorithmus
12.01.17	„interne“ Resolution: Debugging inkompatibler Bibliotheken
13.01.17	„interne“ Resolution: Vergleich Sheared Ellipse mit Wavefront Methode

16.01.17 „interne“ Resolution: Rückfall auf Wavefrontalgorithmus, Verfeinerung des Algorithmus

17.01.17 „interne“ Resolution: Aufteilung der Unterfunktionen in verschiedene Bibliotheken

18.01.17 „interne“ Resolution: Performanceanalyse: Wavefront und Raute

19.01.17 „interne“ Resolution: Performanceoptimierung: Lookup tables und Schleifenanalyse

20.01.17 „interne“ Resolution: Performanceoptimierung: Umwandlung lokaler Variablen in Referenzen

23.01.17 GPS-Modul: Konzeptionierung der Integration in bestehendes Framwork

24.01.17 GPS-Modul: Modulframework Umstrukturierung

25.01.17 GPS-Modul: Modulframework: Optimize Pose Funktion in top level class

26.01.17 GPS-Modul: Modulframework: Anbindung an den SLAM Manager überarbeiten

27.01.17 GPS-Modul: Modulframework: Reconfigure Variablen anpassen zum Debugging und Tuning

30.01.17 GPS-Modul: Gewichtungsfunktion implementieren: Annahme Gaußverteilung

31.01.17 GPS-Modul: Gewichtungsfunktion optimierung, zusätzlicher Gain für manuelle Gewichtung des Sensoreinflusses

01.02.17 GPS-Modul: Diversität der Partikel in ersten Tests nicht gewährleistet: Debugging

02.02.17 GPS-Modul: Anpassung des Partikelfilters durch Präzisionsunterschiede verschiedenener Sensoren

03.02.17 GPS-Modul: Partikelfilteranpassung: Kovarianzeinstellung der Messung

06.02.17 GPS-Modul: Bug in der Implementierung der Mahalanobis Distanz, Recherche und Bugfix

07.02.17 GPS-Modul: BWM i3 bagfiles zum Testen der Funktionalität

08.02.17 GPS-Modul: Anpassung der bagfiles an Testszenario, Cropping und Filtern

09.02.17 GPS-Modul: Recherche bash scripting zum Automatisieren der Vorprozessierung

10.02.17 GPS-Modul: Umsetzung der bash scripts

13.02.17	GPS-Modul: Simulation mit Hilfe der bagfiles im Flughafenszenario
14.02.17	GPS-Modul: In der Simulation auffallende Bugs analysieren und fixen
15.02.17	GPS-Modul: Erneute Simulation zur Analyse der Performance des integrierten Moduls
16.02.17	GPS-Modul: Performanceanalyse in anderer Testumgebung
17.02.17	GPS-Modul: Implementierung verschiedener Anbindungen zu anderen Modulen
20.02.17	GPS-Modul: Übergabe der ausstehenden Aufagben an Nachfolger
21.02.17	GPS-Modul: Übergabe und Organisatorisches
22.02.17	Urlaub/Feiertag/Überstunden
23.02.17	Urlaub/Feiertag/Überstunden
24.02.17	Urlaub/Feiertag/Überstunden
27.02.17	Urlaub/Feiertag/Überstunden
28.02.17	Urlaub/Feiertag/Überstunden

A.2 Praktikum Zeitraum 2 (01.08.17-12.09.17)

01.08.17 Wiedereinarbeitung: Stand des LTS, Organisatorisches

02.08.17 Wiedereinarbeitung: Codeanalyse neuer Funktionalitäten und Task/Worker Struktur

03.08.17 KLD-Sampling: Referenzpaper zur Implementierung des AMCL in ROS

04.08.17 KLD-Sampling: Anpassung des Algorithmus und IPA LTS

07.08.17 KLD-Sampling: Integration des Resamplingalgorithmus für dynamische Partikelpools

08.08.17 KLD-Sampling: Erster Implementierungsansatz vor dem Resamplingschritt erweist sich als unpraktisch

09.08.17 KLD-Sampling: Implementierung der inkrementellen KLD bound Abfrage im Resamplingschritt

10.08.17 KLD-Sampling: Simulation des Algorithmus in gazebo, debugging von Samplingfehlern

11.08.17 KLD-Sampling: Debugging weiterer Samplingfehler

14.08.17 RFID-Modul: Konzeptionierung: Analogon zu vorher implementiertes GPS-Modul

15.08.17 RFID-Modul: Aufbau des Modulframeworks (analog zu GPS Modul)

16.08.17 RFID-Modul: Modulframework: Anbindungen zu SLAM Manager, reconfigure Parameter

17.08.17 RFID-Modul: Strukturelle Unterschiede in der bottom layer class auflösen

18.08.17 RFID-Modul: Finale Strukturanpassung in der Sensormodul Klassenstruktur

21.08.17 Einarbeitung ATF: Github repo auschecken und zum Laufen bringen

22.08.17 Einarbeitung ATF: Anpassung des ATF an eigene Testszenarien, Ideeausarbeitung für einen Jenkins Server

23.08.17 KLD-Sampling: Performanceanalyse mithilfe des ATFs erweist sich als unintuitiv

24.08.17 KLD-Sampling: Performanceanalyse mithilfe von shell scripts, CPU und MEM Tests

25.08.17 KLD-Sampling: Parametertuning vom Samplingalgorithmus auf Basis der Analyseergebnisse

28.08.17	KLD-Sampling: weitere Performanceanalyse mit verfeinerten Parametern
29.08.17	KLD-Sampling: Aufzeichnung der Ergebnisse für die Dokumentation
30.08.17	RFID-Modul: Implementierung Gewichtungsfunktion für vorhandenen RFID Sensor
31.08.17	RFID-Modul: Datenblatt des RFID Sensors durchlesen und geeignete Gewichtung synthetisieren
01.09.17	RFID-Modul: Implementierung einer Gauß-/Rechteckverteilung als Observationsmodell
04.09.17	RFID-Modul: Initiales Observationmodell erweist sich als suboptimal: Ursachenuntersuchung
05.09.17	RFID-Modul: Erweiterung der Gewichtsfunktion um spärliche Partikeldichten zu berücksichtigen
06.09.17	RFID-Modul: Testen von neuer Gewichtsfunktion in Simulationsumgebung
07.09.17	Urlaub/Feiertag/Überstunden
08.09.17	Urlaub/Feiertag/Überstunden
11.09.17	RFID-Modul: Testen von neuer Gewichtsfunktion mit bagfiles
12.09.17	Organisatorisches und Übergabe

Literatur

[1] P. Barsch. “Entwicklung eines SLAM-Verfahrens für den Dauereinsatz in dynamischen Umgebungen zur kooperativen Navigation von mobilen Robotersystemen”. Technische Universität Dresden, 2016.

[2] S. Dörr u. a. “Cooperative Longterm SLAM for Navigating Mobile Robots in Industrial Applications”. In: *International Conference on Multisensor Fusion and Integration for Intelligent Systems* (2016).

[3] E. Einhorn und H. Gross. “Generic NDT mapping in dynamic environments and its application for lifelong SLAM”. In: *Robotics and Autonomous Systems (Vol. 69)* (2015).

[4] D. Fox. “KLD-Sampling. Adaptive Particle Filters”. In: *Advances in Neural Information Processing Systems, NIPS (Vol. 14)* (2001).

[5] *Fraunhofer Gesellschaft Webseite*. URL: https://www.ipa.fraunhofer.de/.

[6] G. Grisetti, C. Stachniss und W. Burgard. “Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters”. In: *IEEE Transactions on Robotics (Vol. 23)* (2007).

[7] G. Grisetti, C. Stachniss und W. Burgard. “Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling”. In: *IEEE International Conference on Robotics and Automation, 2005* (2005).

[8] M. L. V. Pitteway. “Algorithm for drawing ellipses or hyperbolae with a digital plotter”. In: *The Computer Journal (Vol.10)* (1967).

[9] *Robohub Webseite*. URL: http://robohub.org/ros-101-intro-to-the-robot-operating-system/.

[10] *Robot Operating System Webseite*. URL: www.ros.org/.

[11] S. Thrun, W. Burgard und D. Fox. *Probabilistic Robotics*. MIT Press, 2006.

[12] G. D. Tipaldi, D. Meyer-Delius und W. Burgard. “Lifelong localization in changing environments”. In: *The International Journal of Robotics Research, SAGE* (2013).

Ort, Datum	Unterschrift Student	Unterschrift Betreuer