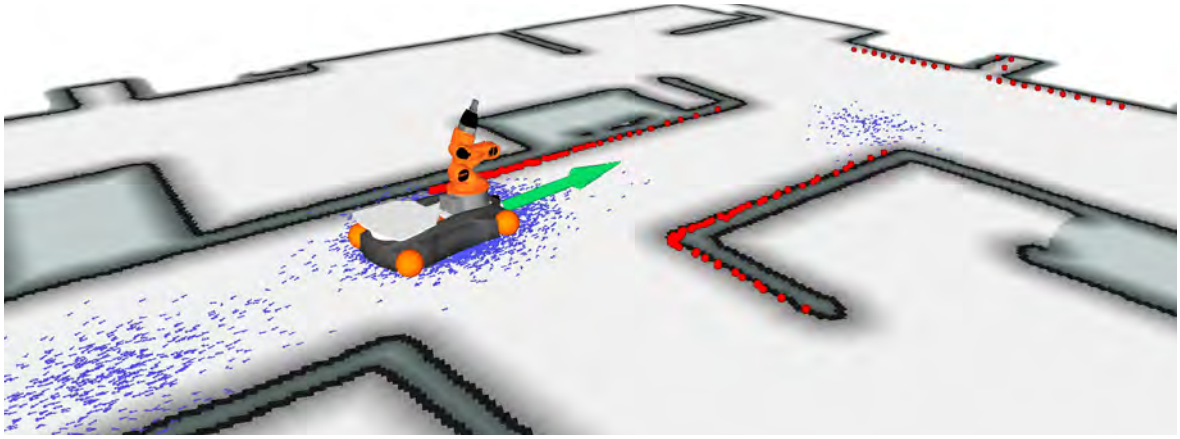
Masterarbeit

Entwicklung eines Lokalisierungssystems für fahrerlose Transportsysteme

WS 2016/17

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM



vorgelegt von:	Daniel Ammon
Fakultät:	efi
Studiengang:	Master of Applied Research
Matrikelnummer:	2 254 880
Erstgutachter:	Prof. Dr. Stefan May
Zweitgutachter:	Prof. Dr. Jörg Arndt

Prüfungsrechtliche Erklärung

Ich, Daniel Ammon, Matrikel-Nr. 2 254 880, versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

_____

Unterschrift

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Algorithmen

Mathematische Notationen

x	beschreibt eine skalare Größe.
$\vec{x}$	beschreibt einen Vektor.
$\mathbf{X}$	beschreibt eine Matrix.
M	beschreibt ein kartesisches Koordinatensystem.
x^M	beschreibt die Variable x bezogen auf das Koordinatensystem M.
$\widetilde{x}$	verdeutlicht, dass es sich um eine Schätzung der Variable x handelt.
x_t	beschreibt die Variable x zu der aktuellen Rekursion.
x_{t-1}	beschreibt die Variable x zu der zurückliegenden Rekursion.
x_{t+1}	beschreibt die Variable x zu der nächsten Rekursion.
$x_{1...t}$	beschreibt die Gruppe der Variablen von der ersten Rekursion bis zu der aktuellen Rekursion $\{x_1, x_2, \ldots, x_t\}$.
x^n	beschreibt eine einzelne Variable, aus einer Gruppe von ähnlichen Variablen.
$\vec{x}^i$	beschreibt die Pose eines einzelnes Partikels des Partikelfilters.
$rot(\phi)$	beschreibt eine 2D-Rotationsmatrix um den Winkel ϕ.
$p(x)$	beschreibt die Wahrscheinlichkeitsdichtefunktion der Variable x.
$p(x = a)$	beschreibt die Wahrscheinlichkeit für das Ereignis $x = a$.
$p\left(x \mid y\right)$	beschreibt die Wahrscheinlichkeitsdichtefunktion der Variable x unter der Voraussetzung y (bedingte Wahrscheinlichkeit).
$p\left(x = a \mid y = b\right)$	beschreibt die Wahrscheinlichkeit für das Ereignis $x = a$ unter der Voraussetzung $y = b$.

1 Einleitung

Laut der KUKA AG wurde in den 1970er-Jahren der Grundstein für die heutige Industrie 4.0 durch deren damals einzigartigen sechsachsigen Industrieroboter namens Famulus gelegt[1]. Seither wurden, nicht nur von KUKA, viele Industrieroboter hergestellt, um Produktionsabläufe zu optimieren. Abbildung 1a zeigt drei Industrieroboter, wie sie heutzutage erhältlich sind, in einem klassischen Fertigungsumfeld. Solch ein Robotersystem bietet eine hohe Flexibilität und ein hohes Optimierungspotenzial für die industrielle Fertigung vieler Produkte. Eines kann es jedoch nicht: sich fortbewegen.

(a) (b) (c) (d)

Abbildung 1: Verschiedene Robotersysteme: (a) Industrieroboter der KUKA AG [`https://www.kuka.com`] (b) Rettungsroboter Schrödi [TH-Nürnberg] (c) Transportroboter der Firma Kivasystems [`http://www.kivasystems.com`] (d) Transportroboter der Firma Fetch Robotics [`http://www.fetchrobotics.com`]

Industrie 4.0 ist ein mächtiger Begriff, der sehr unterschiedlich ausgelegt wird. Einigkeit besteht jedoch darin, dass die Flexibilitätsanforderungen an Produktionsstätten steigen werden. Robotersysteme, die sich selbständig durch die Fertigungsumgebung bewegen können und dort Aufgaben erledigen, könnten den steigenden Flexibilitätsanforderungen gerecht werden.

Abbildung 1b zeigt einen Rettungsroboter der Technischen Hochschule Nürnberg (TH-Nürnberg) namens Schrödi. Dieser kann sich zwar fortbewegen und dank des Kettenantriebs sogar Treppen erklimmen, doch er bewegt sich nicht selbstständig. Stattdessen er wird – wie bei vielen Rettungsrobotern üblich – durch eine Fernsteuerung gesteuert.

Abbildung 1c zeigt einen Transportroboter der Firma Kivasystems. Er wird in den Lagerhallen des Versandhandels “amazon“ verwendet, um Güter zu transportieren. Auch Abbildung 1d zeigt ein solches System: den Transportroboter der Firma Fetch Robotics. Beide Systeme können sich selbstständig durch ihre Einsatzumgebung bewegen und Aufgaben erledigen. Folgende Aussage verdeutlicht die Besonderheit der Robotersysteme von Kivasystems und Fetch Robotics:

> Die **selbstständige** Bewegung von A nach B erfordert die **Kenntnis über den eigenen Standort** relativ zu den Zielpositionen A und B.

[1] `https://www.kuka.com/über-kuka/unternehmensstruktur/kuka-roboter` (Zugriff: 03.12.2016)

Die Kenntnis über den eigenen Standort ist für jede kontrollierte autonome Bewegung eines beliebigen Robotersystems erforderlich. Das selbstständige Erreichen und Beibehalten dieser Kenntnis eines mobilen Roboters wird Roboterlokalisierung genannt und ist Gegenstand dieser Arbeit.

1.1 Aufgabenstellung

Die Aufgabe dieser Arbeit besteht in der Entwicklung eines Lokalisierungssystems für fahrerlose Transportsysteme. Neben der Implementierung des Lokalisierungssystems muss zusätzlich ein geeignetes Sensorkonzept entworfen werden.

Das Transportsystem agiert im Innenbereich, beispielsweise in einer Fabrik, wobei die Umgebung veränderlich aber bekannt ist. Das System soll möglichst modular und autark beschaffen sein, um eine einfache Erweiterung und Wartung zu gewährleisten. Um eine kostengünstige Integration zu ermöglichen, soll das Lokalisierungssystem wenige bis keine Anforderungen an den Einsatzort haben. Der Positionierungsfehler des Systems muss im Größenbereich weniger Zentimeter beziehungsweise weniger Grad liegen.

1.2 Struktur der Arbeit

Nachdem in Kapitel 1 bisher die Aufgabenstellung und die Motivation der Arbeit dargelegt wurden, wird nun der Inhalt der folgenden Kapitel und somit der Aufbau der Arbeit beschrieben.

Um eine gemeinsame Kommunikationsbasis und ein Verständnis des Gesamtprojekts zu schaffen, werden in Kapitel 2 einige Definitionen und Anforderungen beschrieben. Kapitel 3 gibt einen Überblick über den Stand der Technik bezüglich Roboterlokalisierung anhand der Literatur. In Kapitel 4 befindet sich der theoretische Teil zu dem in dieser Arbeit verwendeten Partikelfilteralgorithmus zur Roboterlokalisierung. Kapitel 5 erläutert den gewählten Lösungsansatz sowie Details bezüglich der Implementierung der Lokalisierungssoftware und des Vorgehens während der Softwareimplementierungsphase. Die Verifikation des entwickelten Lokalisierungssystems ist in Kapitel 6 in Form mehrerer Tests und deren Auswertungen zu finden. Abschließend wird in Kapitel 7 eine Zusammenfassung der Ergebnisse und ein Ausblick auf weitere Arbeiten gegeben.

2 Definitionen und Anforderungen

Anforderungen stellen eine Beschreibung des Projekts dar und sorgen für überprüfbare Eigenschaften. Der Erfolg eines Projekts, kann mit einem Abgleich der anfangs aufgestellten Anforderungen bewertet werden. Zur Erfassung, Gruppierung und Darstellung der Anforderungen für dieses Projekt werden Herangehensweisen aus [22] verwendet.

Um Anforderungen definieren zu können, werden vorerst einige Definitionen und Begrifflichkeiten präzisiert, um eine gemeinsame Kommunikationsbasis zu erschaffen. Anschließend werden in diesem Kapitel einzelne Punkte aus der resultierenden Anforderungsliste (siehe Tabelle 2) erläutert. Weitere Projektanforderungen resultieren aus der im ersten Teil des Projekts angestellten Literaturstudie [2] und aus Erfahrungen während der Arbeit mit autonomen Robotersystemen im Robotiklabor der TH-Nürnberg. Die Markierung [→ Anforderungsliste] hebt Gedanken hervor, die sinngemäß in der Anforderungsliste wiederzufinden sind.

2.1 Begriff “Fahrerloses Transportsystem“

Der Begriff “Fahrerloses Transportsystem“ (FTS), wie er in dieser Arbeit verwendet wird, resultiert aus der Zusammenarbeit mit der Firma Robert Bosch GmbH (Bosch). Abbildung 2 zeigt deren Prototypen eines fahrerlosen Transportsystems.

Abbildung 2: Prototyp des FTS von Bosch: Der Prototyp wie er bei Bosch in der Fertigung eingesetzt werden soll. Die Traglast beträgt 200 kg und das Gehäuse wird mit einem 3D-Drucker hergestellt. Die Energieversorgung erfolgt durch zwei austauschbare Akkus aus den Bosch eigenen Akku-Bohrschraubern. [Bosch GmbH]

Es handelt sich um ein Transportsystem mit Differenzialantrieb. Die am hinteren Teil des Fahrzeugs angebrachte Ladefläche kann Lasten bis 200 kg aufnehmen und mittels eines Hebebühnen-Mechanismus zirka 5 cm anheben. Gegenstand des “AutoBod[2]-Projektes“ von Bosch ist es, durch eine Flotte von autonom arbeitenden Transportfahrzeugen die Produktivität und Flexibilität der Fertigung zu erhöhen. Die TH-Nürnberg fungiert hierbei als Forschungspartner und unterstützt mit der hier vorliegenden und anderen studentischen Arbeiten.

Da das AutoBod-System nicht, wie bei FTS oft üblich, in eigenen Zonen in der Fabrik, sondern zusammen mit beispielsweise Mitarbeitern agieren soll, ergeben sich neue Anforderungen. Diese werden von Bosch mithilfe des Prototyps ermittelt.

[2] AutoBod steht für “autonomer Bodenroller“

Eine dieser Anforderungen besteht in der selbstständigen Lokalisierung der Fahrzeuge in der Fabrikhalle. Die Entwicklung eines Lokalisierungssystems zu diesem Zweck bildet den Kern der vorliegenden Arbeit und ist eine Unterstützung der prototypischen Entwicklung des Bosch eigenen FTS.

Da das zu entwickelnde Lokalisierungssystem auch Verwendung im Robotiklabor der TH-Nürnberg finden soll, kann der Begriff fahrerloses Transportsystem in dieser Arbeit auch mit "autonomer mobiler Roboter" gleichgesetzt werden. In Kapitel 6.4 sind alle Hardwareplattformen auf denen das Lokalisierungssystem bis zum Zeitpunkt der Fertigstellung dieser Arbeit Verwendung findet aufgeführt.

2.2 Begriff "Lokalisierung"

Abbildung 3 zeigt die Einordnung des Begriffs der Lokalisierung und wie dieser in der vorliegenden Arbeit aufzufassen ist. Lokalisierung betitelt im Allgemeinen die Bestimmung des Ortes (lat. locus ⇒ Ort). Übertragen auf die Robotik beziehungsweise auf autonome Transportfahrzeuge bezeichnet Lokalisieren das Bestimmen des eigenen Standortes. Wird zum Verbessern des Lokalisierungszustands eine geplante Bewegung durchgeführt, wird von aktiver Lokalisierung gesprochen. In der Robotik werden viele Anstrengungen dem Bereich SLAM (*Simultaneous Localization and Mapping*) gewidmet. Hierbei wird während der Lokalisierung des Roboters ein digitales Abbild der Umgebung erstellt. Der Roboter kann dabei ferngesteuert werden. Ist dies nicht der Fall und der Roboter führt seine Bewegungen autonom aus, handelt es sich um eine Exploration. Hierbei zeichnet der Roboter, während er autonom durch unbekanntes Terrain navigiert, eine Repräsentation seiner Umgebung auf. Währenddessen muss er seine eigene Pose[3] kennen, also sich lokalisieren.

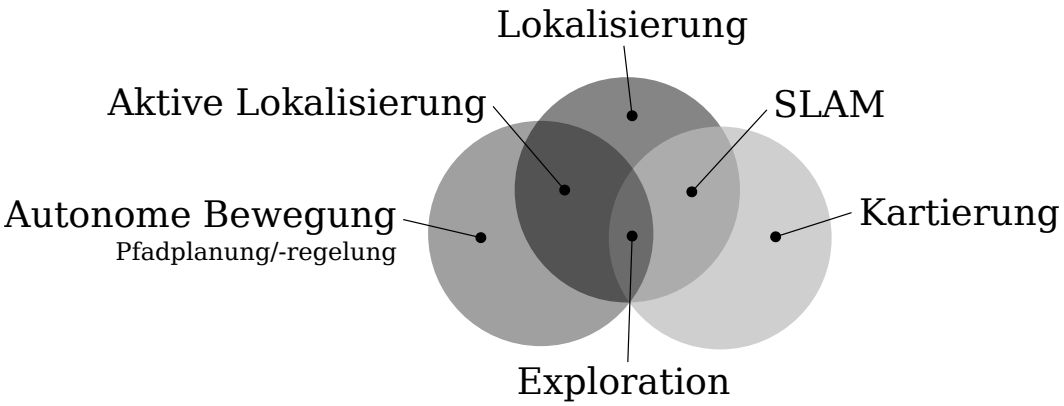


Abbildung 3: Begriff Lokalisierung: Abgrenzung des Begriffs Lokalisierung von naheliegenden Disziplinen der Robotik. [eigene Abbildung; Idee aus [19]]

Diese Arbeit befasst sich hauptsächlich mit dem reinen Lokalisierungsbereich. Mit einer digitalen Repräsentation der Umgebung kann während der Lokalisierung ein Bezug des Roboters zur realen Umgebung hergestellt werden. Mithilfe von SLAM kann durch eine Kartierung der Umgebung ein solches Abbild angelegt werden. Somit spielt auch der Bereich SLAM eine Rolle in dieser Arbeit.

[3] Die Pose beschreibt die Position und Orientierung des Roboters im zweidimensionalen Raum.

Der Bereich Exploration wird im Ausblick dieser Arbeit kurz thematisiert. Dieser spielt jedoch in der Arbeit selbst keine Rolle, da es sich um eine vor der Lokalisierungsphase bekannte Umgebung handelt (Fabrikhalle, Laborumgebung) und so keine reine Exploration, wie beispielsweise bei der Rettungsrobotik, nötig ist. Den Bereich der aktiven Lokalisierung greift der folgende Absatz erneut auf.

Der Begriff "Roboterlokalisierung" kann auf weitere Art und Weise untergliedert werden. Die folgende Untergliederung wird im Kontext des vorliegenden Anwendungsfalls untersucht. Ihre Art ist aus [25] übernommen:

- globale Lokalisierung, Tracking und Relokalisierung
- aktive und passive Lokalisierung

Globale Lokalisierung, Tracking und Relokalisierung

Die Ortung eines Roboters bei vorher unbekannter Pose wird als globale Lokalisierung bezeichnet. Ein Beispiel hierfür ist das Einschalten eines Robotersystems an einer dem Roboter unbekannten Pose. Damit das Robotersystem handlungsfähig ist, muss es seine Pose in einer globalen Lokalisierungsphase herausfinden. Meist kennt der Roboter hierfür eine Repräsentation seiner Einsatzumgebung – beispielsweise eine zuvor erstellte Umgebungskarte. Hat der Roboter eine ausreichend gute Annahme über seine initiale Pose getroffen, kann er Aufträge annehmen und ausführen – etwa neue Posen im Einsatzort anfahren. Hierzu muss er seine Pose verfolgen, um während der Bewegung im lokalisierten Zustand zu bleiben. Diese Art von Lokalisierung wird als Tracking (engl. tracking ⇒ verfolgen) bezeichnet. Verliert der Roboter das Wissen über seine Pose oder erkennt er, dass die Annahme über seine Pose fehlerhaft ist, kann der Versuch der Relokalisierung eingeleitet werden. Der Roboter versucht sich hierbei neu zu orten, beispielsweise durch eine erneute globale Lokalisierungsphase.

Eine globale Lokalisierungsphase kann durch die Vorgabe der Startpose des Roboters ausgelassen werden. Auch die Fähigkeit der Relokalisierung ist für den Einsatz eines Transportroboters nicht essenziell, da im Falle des Verlusts der Ortung, das System angehalten werden kann, und weitere Schritte durch zum Beispiel Servicepersonal eingeleitet werden können. Das Verfolgen der eigenen Pose ist jedoch Grundbestandteil der Roboterlokalisierung und somit die Mindestanforderung an eine erfolgreiche Roboterlokalisierung.

Somit muss das zu entwickelnde System ein Tracking der Pose des Roboters ermöglichen [→ Anforderungsliste]. Es sollte jedoch auch die Möglichkeit der globalen Lokalisierung und der Relokalisierung bieten, da dies die Integration und Verwendung des mit dem Lokalisierungssystem versehenen Roboters stark vereinfacht [→ Anforderungsliste].

Aktive- und passive Lokalisierung

Bei einer aktiven Lokalisierung führt der Roboter gezielt Bewegungen aus, die zu seiner Lokalisierung beitragen. Ein Roboter, der sich nach dem Einschalten im unlokalisierten Zustand im Kreis dreht, um die Umgebung aufzunehmen, verwendet ein aktives Lokalisierungssystem. Passive Lokalisierungssysteme hingegen wirken nicht aktiv auf den Roboter ein. In dieser Arbeit wird ein solches passives Lokalisierungssystem entwickelt, das nicht auf den Betriebszustand des Roboters einwirkt [→ Anforderungsliste]. Dem Lokalisierungssystem übergeordnete Kontrollsysteme, wie Zustandsautomaten oder Roboterkontrollarchitekturen, können durch Einwirken auf den Lokalisierungsalgorithmus und den Bewegungszustand des Roboters trotzdem eine Art aktiver Lokalisierung damit betreiben (siehe Kapitel 7.2).

2.3 Projektanforderungen

Nachfolgend werden exemplarisch weitere Projektanforderungen sowie deren Herleitung genauer beschrieben.

Maximalgeschwindigkeiten

Das Lokalisierungssystem muss in der Lage sein, auch bei maximaler Bewegungsgeschwindigkeit die Lokalisierung zu gewährleisten. Bei industriell eingesetzten Transportrobotern müssen aus Sicherheitsgründen Geschwindigkeitsvorgaben eingehalten werden. Aus Recherchen in Zusammenarbeit mit der Firma Bosch geht hervor, dass es hier bezüglich voll autonomer Transportsysteme noch an Gesetzgebung und Vorgaben mangelt. Als Richtwert hat sich folgende Aussage herauskristallisiert: “Kein Punkt des Fahrzeugs darf eine Geschwindigkeit von 1 m/s überschreiten.“ Dieser Richtwert legt die maximale translatorische Geschwindigkeit des Fahrzeugs v_{max} auf 1 m/s fest [→ Anforderungsliste]. Für die maximale rotatorische Geschwindigkeit muss die Geometrie des Fahrzeugs untersucht werden. Maximale Drehgeschwindigkeiten entstehen beim Drehen um das kinematische Zentrum des Fahrzeugs (Differenzialantrieb vorausgesetzt). Bei maximaler Drehgeschwindigkeit muss sich der äußerste Punkt des Fahrzeugs langsamer als 1 m/s bewegen. Hierzu folgende Abbildung:

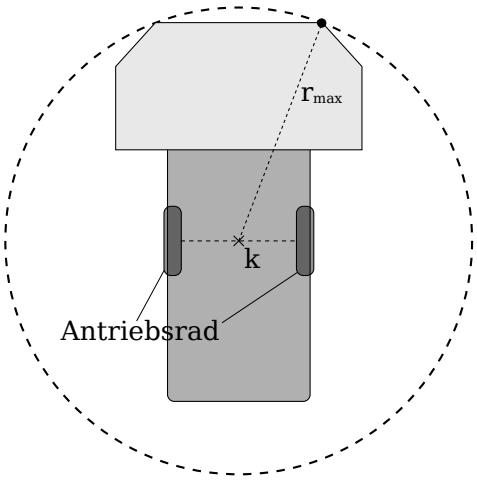


Abbildung 4: Drehgeschwindigkeit: Durch die Geometrie des Transportfahrzeugs und das mit k beschriebene kinematische Zentrum kann der maximale Radius r_{max} ermittelt werden. [eigene Abbildung]

Bei dem FTS von Bosch beträgt r_{max} 70 cm, womit auch in der Anforderungsliste gerechnet wird. Formel 1 verdeutlicht die Berechnung der maximalen Drehgeschwindigkeit ω_{max} [→ Anforderungsliste].

$$\omega_{max} = \frac{v_{max}}{r_{max}} = \frac{1\ m/s}{0{,}7\ m} \approx 1{,}43\ \frac{rad}{s} \approx 82\ \frac{^\circ}{s} \tag{1}$$

Genauigkeit

Die Genauigkeit des Lokalisierungssystems hängt stark von dem verwendeten Sensorkonzept, der Roboterplattform selbst und den Umgebungsbedingungen ab. Als Mindestanforderung wird dennoch im Bezug auf die zu ermittelnde Pose ein absoluter translatorischer Fehler $e_{t,max}$ von 10 cm und ein absoluter rotatorischer Fehler $e_{r,max}$ von 6 ° (≈ 0,1 rad) festgelegt [→ Anforderungsliste].

Aktualisierungsrate

Um die Pose des Transportfahrzeugs für weitere Systeme, beispielsweise ein Navigationssystem, zu verwenden, muss die Aktualisierungsrate sowie die Positioniergenauigkeit des System bekannt sein. Ein Richtwert für die Mindestanforderung an die Aktualisierungsrate des Systems lässt sich aus den vorher festgelegten Maximalgeschwindigkeiten und der gewünschten Positioniergenauigkeit errechnen. Bewegt sich das Transportfahrzeug mit maximaler Geschwindigkeit, darf die Posenänderung zwischen zwei Aktualisierungsvorgängen (Iterationen) des Lokalisierungssystems die maximal zugelassenen Fehlerwerte ($e_{t,max}$, $e_{r,max}$) nicht übersteigen. Formel 2 zeigt diesen Zusammenhang:

$$f_{min} = \max\left(\frac{v_{max}}{e_{t,max}}; \frac{\omega_{max}}{e_{r,max}}\right) = \max\left(10\,\text{Hz}; 13{,}7\,\text{Hz}\right) \approx 14\,\text{Hz} \tag{2}$$

Werden die oben festgelegten Geschwindigkeiten und Genauigkeiten in Formel 2 eingesetzt, ist das Ergebnis eine minimale Aktualisierungsrate von 14 Hz. Bei der Festlegung dominiert die Vorgabe aus der Drehgeschwindigkeit. Unter diesen Betrachtungen werden 20 Hz als minimale Aktualisierungsrate des Lokalisierungssystems festgelegt [→ Anforderungsliste].

Softwarestruktur

Partsch empfiehlt die Darstellung des Projekts als Anwendungsfall (*use case*), um die Sichtweisen von beteiligten Personen auf das zu entwickelnde System zu analysieren [22]. Abbildung 5 zeigt den so entstandenen Anwendungsfall.

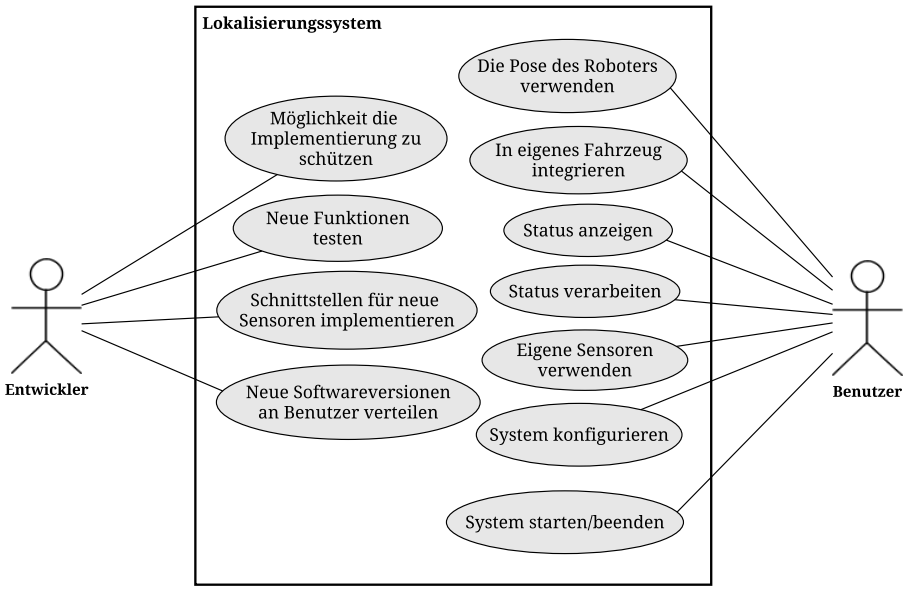


Abbildung 5: Anwendungsfall: Entwickler und Benutzer haben verschiedene Sichtweisen und somit verschiedene Systemanforderungen an das zu entwickelnde Lokalisierungssystem. [eigene Abbildung]

Der Benutzer möchte das Lokalisierungssystem in sein eigenes Fahrzeug integrieren und Informationen aus dem System weiterverwerten (zum Beispiel in einem Roboternavigationssystem). Neben dem Konfigurieren und Bedienen des Systems möchte er seine eigenen, eventuell schon vorhandenen, Sensorsysteme verwenden [→ Anforderungsliste]. Hierfür benötigt die Software eine klar definierte Schnittstelle für den Benutzer. Diese ermöglicht eine einfachere Anbindung der Software an das benutzereigene System und eröffnet dem Entwickler gleichzeitig weitere Vorteile. Er kann seine Entwicklung als Softwarebibliothek mit definierten Schnittstellen veröffentlichen [→ Anforderungsliste]. So kann der Entwickler neue Versionen der Bibliothek an den Benutzer verteilen und bei Bedarf den Quellcode der Entwicklung schützen. Nur die Schnittstellen müssen für den Benutzer zugänglich und dokumentiert sein, der Quellcode nicht.

Um das Lokalisierungssystem in möglichst viele Transportfahrzeuge integrieren zu können, muss es von möglichst wenig anderen Systemen abhängig sein. Viele Roboter werden mit ROS[4] (Roboter-Operating-System) betrieben. Ohne eine Abhängigkeit zu ROS kann das System auch auf anderen Plattformen, wie zum Beispiel Eigenentwicklungen des Benutzers, verwendet werden [→ Anforderungsliste].

[4] `http://www.ros.org`

3 Stand der Technik

Dieser Arbeit geht eine Literaturstudie voraus, die in Form einer wissenschaftlichen Veröffentlichung [2] vorliegt. Diese ist ein wichtiger Bestandteil des Literaturteils, da viele ihrer Aspekte neu interpretiert und dadurch in dieser Arbeit verwendet werden können. Die nachfolgenden Abschnitte erfassen den Stand der Technik bezüglich der Lokalisierung autonomer Robotersysteme im industriellen Umfeld.

3.1 Wichtige Rolle der Merkmale

Um auf hoher See zu navigieren, müssen Seefahrer ihre Pose bestimmen. In den Zeiten vor Satellitennavigationssystemen wurden Sterne zur Orientierung verwendet. Deren Position relativ zur Erde war bekannt und erforscht. Der Vergleich der sichtbaren Sterne am aktuellen Himmelsausschnitt mit vorher gespeicherten beziehungsweise notierten Daten über den Sternenhimmel, lies auf die eigene Pose auf der Erdoberfläche schließen. Die Sterne fungieren hierbei als Merkmale (engl. *features*) am Himmelsbild. Zur Lösung des Lokalisierungsproblems wird das Extrahieren, Speichern und Vergleichen von Merkmalen oft angewendet. Herausforderung hierbei ist es, Merkmale geschickt zu definieren, um sie möglichst effizient speichern und anschließend vergleichen zu können. Nachfolgend werden Beispiele für Merkmale, die bei der Problemstellung der Roboterlokalisierung Verwendung finden, diskutiert.

Optische Sensoren, wie Kameras, sammeln in kurzer Zeit sehr viele Informationen aus der Umgebung. Das Reduzieren und Konzentrieren der Daten durch die Definition von Merkmalen ist hier unumgänglich.

In Form einer wissenschaftlichen Publikation definieren Dias u. a. die Leuchten in einer Fabrikhalle als Merkmal. Nachdem die Positionen aller in der Umgebung befindlicher Leuchten aufgenommen und gespeichert wurden, können in der Lokalisierungsphase anhand der im aktuellen Bildausschnitt befindlichen Leuchten Rückschlüsse auf die aktuelle Pose gezogen werden. [5]

Thrun u. a. betrachten in aufgenommenen Bildern der Raumdecke die Helligkeit über das gesamte Bild verteilt. Nachdem Daten aus der Umgebung mit aufgenommen und gespeichert wurden, kann in der Lokalisierungsphase anhand der aktuell aufgenommenen Bilder ermittelt werden, ob sich der Roboter unter einer Lichtquelle befindet. Mit den gespeicherten Bilddaten können daraus Annahmen über die aktuelle Pose generiert werden. [27]

Durch das Anbringen von QR-Codes in der Umgebung können eindeutig detektierbare Merkmale eingebracht werden. Optische Sensoren können deren Pose relativ zum zu lokalisierenden Objekt bestimmen [30]. Da durch das Anbringen und die Eindeutigkeit der Merkmale deren Pose bekannt ist, kann beim Detektieren eines solchen Merkmals auf die eigene Pose geschlossen werden.

Hahnel u. a. verfolgen durch die Verwendung von RFID-Transpondern das gleiche Prinzip. Die Detektion der zuvor eingebrachten Features erfolgt hier jedoch nicht wie bei QR-Codes durch optische Sensoren, sondern mittels Funkübertragung. [14]

Funksysteme können auch zur Lokalisierung eingesetzt werden. Hier hat sich im Außenbereich das GPS etabliert. Da ein GPS-Signal in Innenräumen nur begrenzt verfügbar ist, wird hier nicht näher darauf eingegangen. D-GPS Systeme arbeiten auch in Innenräumen und können die Lokalisierung stark vereinfachen, sind aber meist sehr kostenintensiv und werden deshalb

auch nicht weiter betrachtet. Trotzdem gibt es viele, auf Funkübertragung basierende Ansätze, die für das vorliegende Problem eine Lösung darstellen können und deshalb nachfolgend erläutert werden.

Eine Methode zur Gewinnung eines der offensichtlichsten Merkmale bezüglich Funk ist das Messen der in der Umgebung verfügbaren Zugriffspunkte (engl. *access points, APs, router*) – genauer, das Messen derer Signalstärken (RSSI – Received Signal Strength Indication).

Wang u. a. verarbeiten die RSSI-Messungen erreichbarer Bluetooth-Zugriffspunkte, um durch Triangulation eine Schätzung der Position zu errechnen. [28]

Grossmann, Schauch und Hakobyan fassen die RSSI-Werte aller erreichbarer Zugriffspunkte zusammen, messen diese an verschiedenen Posen in der Umgebung und speichern sie in einer Datenbank. Durch einen Vergleich der in der Lokalisierungsphase gemessenen Signalstärken mit denen in der Datenbank gespeicherten Signalstärken entstehen Informationen zur aktuellen Pose (RSSI-Fingerprinting-Methode). [13] Das Anbringen gerichteter Empfangsantennen am zu lokalisierenden Objekt, ermöglicht zudem Rückschlüsse auf die Richtung des empfangenen Funksignals und somit auf die Orientierung des zu lokalisierenden Objekts. [23]

Klassische WLAN-Systeme haben den Nachteil, dass der Signalstärkenverlauf von vielen Einflussgrößen abhängt (Dämpfung, Abschattung, Mehrfachreflektionen). Ultrabreitbandsysteme (UWB-Systeme) bieten eine höhere Stabilität, da sie durch das breitere Frequenzband weniger anfällig gegen Störeinflüsse, wie Wände, Personen oder andere Hindernisse, sind. Solche UWB-Funksysteme können auch zur Lokalisierung von Robotern verwendet werden [7, 11].

Eine weitere Gruppe von Werkzeugen zur Gewinnung möglichst eindeutiger Merkmale ist die folgende Gruppe von Sensoren: 2D-Laserscanner, Ultraschall-Sensoren, Infrarot-Sensoren und Radar-Sensoren. Sie messen die Entfernung zu Hindernissen in der Umgebung. Durch Algorithmen wie SLAM werden die Informationen solcher Sensoren in Umgebungskarten gebündelt und gespeichert (beispielsweise [17, 12]). Können während der Lokalisierungsphase von solchen Sensoren aufgenommene Strukturen zur Deckungsgleichheit mit zuvor gespeicherten Umgebungskarten gebracht werden, können wiederum Hinweise über die derzeitige Pose des Sensors, respektive des Roboters erhalten werden (vgl. Kapitel 4.3).

3.2 Verarbeiten der Eigenbewegung

Merkmale aus der Umgebung tragen zur Lokalisierung bei. Der Mensch beobachtet seine Umgebung, sucht nach Merkmalen und orientiert sich an ihnen. Doch er kennt und verarbeitet auch seine eigene Bewegung. Diese liefert keine Informationen über die derzeitige absolute Pose, doch bei erfolgreicher Lokalisierung hilft sie die Lokalisierung aufrecht zu erhalten. Das Verarbeiten der Eigenbewegung wird oft mit dem Ausdruck Koppelnavigation (engl. *dead reckoning*) bezeichnet.

Die Kombination aus Odometriedaten und inertialen Messeinheiten (IMU - Inertial Measurement Unit) zur Koppelnavigation findet in der mobilen Robotik oft Verwendung. Viele im Handel erhältliche IMUs beinhalten einen Kompass. Im Innenbereich ist dieser aufgrund starker Störeinflüsse auf das Erdmagnetfeld jedoch nicht einsetzbar. [15, S. 23ff]

Zusätzlich können visuelle Odometriesysteme [20, 8] Informationen über die Eigenbewegung liefern. Eine zusätzliche Schlupfdetektion der Räder kann die Qualität der Odometriedaten verbessern [21].

3.3 Schlüsselfunktion - Sensordatenfusion

Tabelle 1 ist ein Überblick über einen Teil der in den beiden vorangegangenen Kapiteln erwähnten Lokalisierungstechnologien. Genauere Informationen und Hintergründe zu diesem Vergleich sind der dieser Arbeit vorangestellten Literaturstudie [2] zu entnehmen.

Tabelle 1: Vergleich verschiedener Lokalisierungstechnologien

	RSSI	RSSI (Fingerpr.)	UWB	IMU	Odometrie	Visuelle Odometrie	Optische Merkmale
Unabhängigkeit	-	-	--	+++	+++	+++	+++
Robustheit	--	--	-	+++	+++	--	--
Genauigkeit	--	-	+	+	+	+	++
Kostensparend	++	++	--	+	+	+	+
Absolut	✓	✓	✓	✗	✗	✗	✓

Alle Systeme unterliegen gewissen Vor- und Nachteilen. Der Einsatz eines Systems reicht somit nicht aus, um allen Anforderungen gerecht zu werden. Nachteile eines Systems müssen mit den Vorteilen eines anderen an dieser Stelle ausgeglichen werden. Um genügend Robustheit zu erreichen, müssen die Daten mehrerer Sensorsysteme vereint werden.

Das Kombinieren und Zusammenführen mehrerer Sensorsysteme wird Sensordatenfusion genannt. Die Kombination mehrerer Signale resultiert in einer Ausgangsgröße – in dieser Arbeit in der Pose des zu lokalisierenden Objektes. Zur Sensordatenfusion existieren mehrere Methoden in der Literatur, von denen im Folgenden drei erläutert werden.

Wenn die Ungenauigkeit der zu fusionierenden Messdaten hinreichend genau durch eine Gauß-Verteilung beschrieben werden kann, bildet der Kalman-Filter (KF) [16] ein optimales Werkzeug zur Fusionierung mehrdimensionaler, gegebenenfalls voneinander abhängiger, Größen. Einem KF kann zudem noch ein lineares Systemmodell – meist ein Bewegungsmodell – hinterlegt werden, um das Ausgabesignal durch eine modellbasierte Schätzungen zu stützen.

Ein Extendet-Kalman-Filter (EKF) lässt im Gegensatz zu einem KF auch nichtlineare Abhängigkeiten im Systemmodell zu. Im Vorfeld dieser Arbeit wurde ein solcher EKF zum Fusionieren von Odometriedaten mit IMU-Daten aus dem FTS-Prototypen von Bosch (siehe Kapitel 2.1) untersucht. Es zeigte sich ein deutlicher Informationsgewinn durch das Fusionieren der Einzeldaten (vgl. [1]).

Im Zusammenhang mit den Schlagwörtern Roboterlokalisierung und Sensordatenfusion wird in der Literatur oft der Begriff Monte-Carlo-Lokalisierung[5] genannt. Dieser Überbegriff beschreibt die Lokalisierung und Sensordatenfusion mittels Partikelfiltern. Unter vielen Anderen verwenden die Publikationen [24], [4], [27] und [9] einen Partikelfilter zur Sensordatenfusion und Roboterlokalisierung. Der Partikelfilter stellt ein hoch flexibles Werkzeug zur Sensordatenfusion dar und bietet folgende Vorteile:

- Es kann die globale, die lokale und die Relokalisierung eines Robotersystems betrachtet werden, da zu einem Zeitpunkt verschiedene Annahmen über die derzeitige Pose getroffen und verfolgt werden können. Dies ist möglich, da ein Partikelfilter, im Gegensatz zu KF und EKF, multimodale Verteilungen abbilden kann [15, S. 201].
- Die Integration und parallele Verwendung mehrerer, verschiedener Sensormodelle ist möglich (Lokalisierung mit heterogener Sensorik) [→ Anforderungsliste].
- Die Implementierung kann sehr modular aufgebaut werden, da die Anzahl der Sensoren und der Zeitpunkt der Integration eines Sensormesswertes nicht festgelegt sind.
- Durch das Abtasten der Realität mit "Partikeln" kann die Rechenintensität gegenüber anderer Verfahren, wie zum Beispiel der rasterbasierten Markov-Lokalisierung, gesenkt werden [9].

[5] Monte-Carlo-Algorithmen sind Algorithmen, die nach einem Zufallsprinzip arbeiten. Der Zusammenhang zum Stadtbezirk Monte Carlo entsteht durch die dortige Spielbank - eine der Bekanntesten Spielbanken der Welt. [`https://de.wikipedia.org/wiki/Monte-Carlo-Algorithmus`]

4 Theorie Partikelfilter

Dieses Kapitel beschreibt die theoretischen Grundlagen zur Monte-Carlo-Lokalisierung. Nach einer grundsätzlichen Beschreibung des Partikelfilters in Kapitel 4.1, wird in den nachfolgenden Kapiteln genauer auf die Bereiche Resampling-Schritt, Sensormodell und Bewegungsmodell des Partikelfilters eingegangen.

Auf mathematische Herleitungen wird verzichtet, stattdessen werden die verschiedenen Schritte erläutert, die bei der Verwendung eines Partikelfilters zur Lokalisierung von Robotern durchgeführt werden. Alle mathematischen Herleitungen können in [25] und [15] nachgeschlagen werden. Vorausgesetzt werden Grundlagen der in der Robotik verwendeten Mathematik (Koordinatensysteme, Transformationen) und der Wahrscheinlichkeitstheorie (Nomenklatur in der Wahrscheinlichkeitstheorie, bedingte Wahrscheinlichkeiten). Diese können bei Bedarf in [25] und [15] nachgeschlagen werden. Alle mathematischen Notationen sind Seite 6 zu entnehmen.

4.1 Filterkern

Der Vektor $\vec{x}$ beschreibt die Pose des zu lokalisierenden Objekts:

$$\vec{x} = \begin{pmatrix} x \\ y \\ \phi \end{pmatrix} \approx \vec{\tilde{x}} \tag{3}$$

Im Fall dieser Arbeit ist $\vec{x}$ ein Vektor mit drei Dimensionen: Die Position bestehend aus der x- und y-Koordinate sowie der Orientierung in der xy-Ebene ϕ des Objekts. Das eindeutige Bestimmen von $\vec{x}$ löst das Lokalisierungsproblem. Da es unmöglich ist, die Pose des Roboters eindeutig ohne Fehler zu bestimmen, wird $\vec{\tilde{x}}$ als Schätzung beziehungsweise Annahme von $\vec{x}$ eingeführt.

Zur Erläuterung des Partikelfilter-Algorithmus spielt Formel 4 eine zentrale Rolle. Sie beschreibt den rekursiv ausgelegten Algorithmus des Partikelfilters im mathematischen Sinne:

$$\underbrace{p\left(\vec{x}_t \mid \vec{u}_{1\ldots t}, z_{1\ldots t}\right)}_{\text{neuer Systemzustand}} = \eta \cdot \underbrace{p\left(\vec{x}_{t-1} \mid \vec{u}_{1\ldots t-1}, z_{1\ldots t-1}\right)}_{\text{alter Systemzustand}} \cdot \underbrace{p\left(\vec{x}_t \mid \vec{x}_{t-1}, \vec{u}_t\right)}_{\text{Bewegung}} \cdot \underbrace{p\left(z_t \mid \vec{x}_t\right)}_{\text{Messung}} \tag{4}$$

Nachfolgend wird Formel 4 zur Erläuterung des Partikelfilters analysiert.

Markow-Lokalisierung

Der Grundgedanke eines Partikelfilters basiert auf der Markow-Eigenschaft beziehungsweise der Markow-Lokalisierung. Die Markow-Eigenschaft beschreibt den Zusammenhang stochastischer Ereignisse. Genauer setzt sie die Gedächtnislosigkeit stochastischer wiederkehrender und aufeinanderfolgender Prozesse voraus. Der stochastische Prozess hängt damit nicht von allen vorangegangen Prozessen, sondern nur von seinem direkten Vorgänger ab. Formel 5 zeigt diesen Zusammenhang anhand der Zufallsvariable x.

$$p\left(x_t \mid x_0, x_1, \ldots, x_{t-1}\right) \overset{MARKOW}{\underset{\downarrow}{=}} p\left(x_t \mid x_{t-1}\right) \tag{5}$$

Die Variable x nimmt zu verschiedenen Zeitpunkten verschiedene Werte an. Ihr aktueller Wert x_t hängt jedoch bei Erfüllung der Markow-Eigenschaft nur von ihrem vorherigen Zustand x_{t-1} ab. Zur Berechnung des aktuellen Systemzustandes muss statt aller vorheriger Zustände nur der letzte Zustand des Systems bekannt sein. Durch die Annahme der Markow-Eigenschaft kann der Partikelfilteralgorithmus rekursiv ausgelegt werden. In der Vergangenheit liegende Systemzustände müssen nicht betrachtet werden, wodurch der Speicherbedarf und der Rechenaufwand verringert werden kann.

Eine weiteres Merkmal des Partikelfilters ist die Trennung in Bewegungsschritt und Messschritt. Abbildung 6 beschreibt dieses Merkmal und damit die Grundidee der Markow-Lokalisierung.

Wahrscheinlichkeitsdichtefunktionen

Formel 4 besteht fast gänzlich aus Wahrscheinlichkeitsdichtefunktionen. Nur der Normalisierungsfaktor η stellt eine Ausnahme dar. Er steht für einen skalaren Wert, der die Dichtefunktionen normalisiert, sodass:

$$\int p\left(\vec{x}_t \mid \vec{u}_{1\ldots t}, z_{1\ldots t}\right) d\vec{x}_t = 1.$$

Im Verlauf dieses Kapitels wird auf η näher eingegangen. In diesem Abschnitt wird das Augenmerk jedoch auf die Dichtefunktionen gelegt. Der neue Systemzustand resultiert aus der Multiplikation dreier Wahrscheinlichkeitsdichten. Um dies im rein mathematischen Sinn nach Formel 4 zu bewerkstelligen, müssten diese Wahrscheinlichkeitsdichten funktional beschrieben sein, um sie ganzheitlich multiplizieren zu können. Die funktionale Beschreibung wäre jeweils eine Funktion mit drei Freiheitsgraden, da der Systemzustand $\vec{x}$ aus drei Dimensionen besteht (siehe Formel 3). Modelle, die im Bewegungs- beziehungsweise Messschritt eine funktionale Beschreibung der Wahrscheinlichkeitsdichten generieren, sind entweder sehr rechenintensiv oder sehr eingeschränkt in der Komplexität der Wahrscheinlichkeitsdichten (vgl. Kalman-Filter lässt nur eine Normalverteilung zu). Der Partikelfilter löst diese Problematik indem er die Wahrscheinlichkeitsdichten nicht als geschlossene Funktionen, sondern mit sogenannten Partikeln (*samples*) abbildet beziehungsweise abtastet.

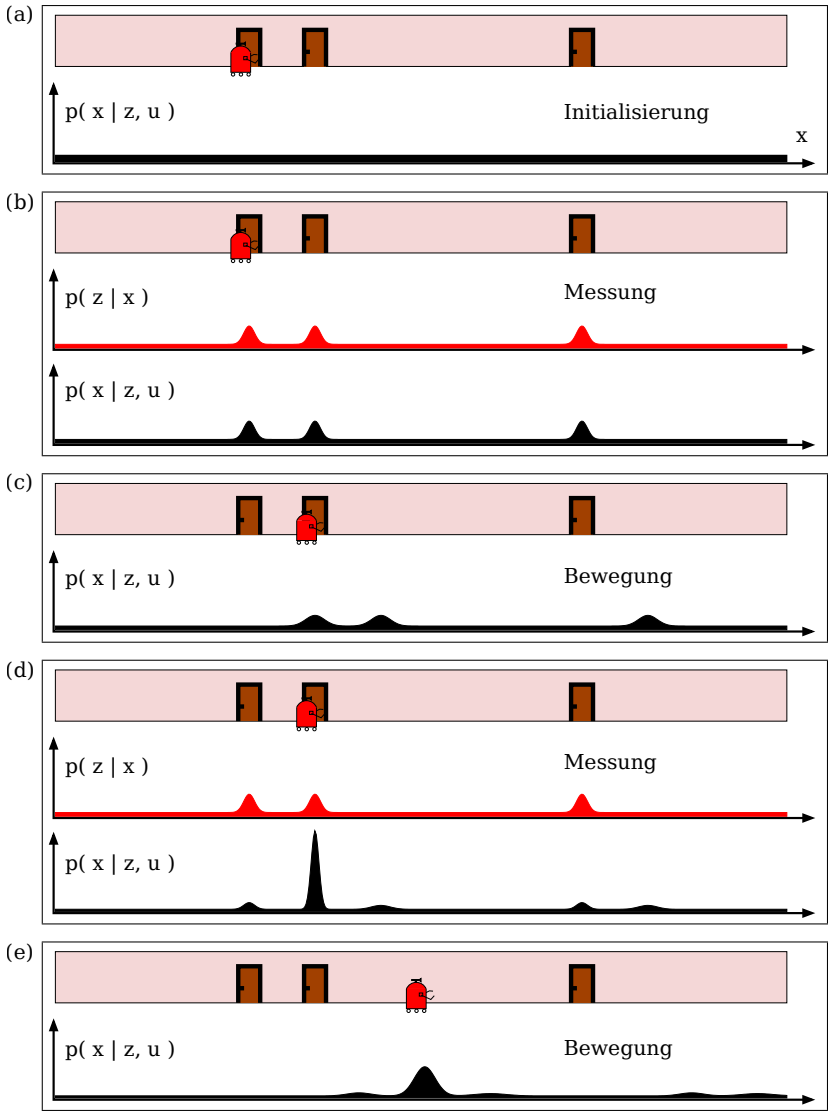


Abbildung 6: Markow-Lokalisierung: Die Ausdrücke $p(z \mid x)$ und $p(x \mid z, u)$ werden in den folgenden Abschnitten genauer erläutert werden. Zu sehen ist ein Roboter beim Lösen eines eindimensionalen Lokalisierungsproblems. Er trägt ein Sensorsystem mit sich, das es ihm ermöglicht Türen zu detektieren, wenn er sich direkt vor ihnen befindet. Des Weiteren kennt der Roboter durch verbaute Radencoder seine eigene Bewegung. Der Roboter wird in bekannter Umgebung eingesetzt, deshalb sind ihm die x-Positionen der Türen bekannt. Lediglich seine eigene Position kennt er nicht (Selbstlokalisierung in bekannter Umgebung). Im ersten Schritt (a) initialisiert er die Annahme über seine eigene Position ($p(x \mid z, u)$) gleichverteilt über die gesamte Umgebung (globale Lokalisierung). In Schritt (b) detektiert das Sensorsystem die vor ihm befindliche Tür. Diese Messung führt zu einer neuen Annahme über die eigene Position. Der Roboter muss sich aufgrund dieser Messung mit hoher Wahrscheinlichkeit vor einer Tür befinden ($p(z \mid x)$). Die Kombination der letzten Annahme über die Position mit der neuen Information liefert die neue Annahme über die aktuelle Position $p(x \mid z, u)$. In Schritt (c) wird eine Bewegung durchgeführt. Der Roboter kennt seine Eigenbewegung, deshalb modifiziert er die Annahme über seine Position passend zur Bewegung. In Schritt (d) nimmt er eine neue Messung auf. Er detektiert erneut eine Tür. Wieder entsteht eine Annahme über die Position aufgrund der Messung ($p(z \mid x)$). Die Kombination von Messung und vorheriger Annahme führt jedoch nun zu einer sehr eindeutigen Schätzung der Eigenposition, repräsentiert durch $p(x \mid z, u)$. Die eindeutige Anordnung der Türen lässt den Roboter aufgrund von Bewegung und Messungen vermuten, dass er sich an der zweiten Tür befindet. Dies entspricht in diesem Fall auch der Wahrheit. Die globale Lokalisierungsphase wäre in diesem Schritt abgeschlossen. Schritt (e) deutet einen weiteren Bewegungsschritt bei bekannter Position an (Tracking). [[25]; Text modifiziert]

Partikel

Durch die Verwendung von Partikeln kann auf die funktionale Beschreibung der Wahrscheinlichkeitsdichten verzichtet werden. Die finite Anzahl der verwendeten Partikel führt zu einer Unterabtastung (*subsampling*) des Ergebnisraums. Hieraus resultiert einerseits eine Ungenauigkeit der Abbildung, andererseits jedoch auch eine Verringerung der Rechenintensität (siehe auch Kapitel 4.2). Jedes einzelne Partikel besteht aus einer Pose $\vec{x}^i$ und einem dieser Pose zugeordneten Gewicht w^i. Das Gewicht ist ein skalarer Wert, unabhängig von der Dimension der zu schätzenden Größe. Abbildung 7 zeigt das Abbilden einer Wahrscheinlichkeitsdichtefunktion durch Partikel an einem eindimensionalen Beispiel.

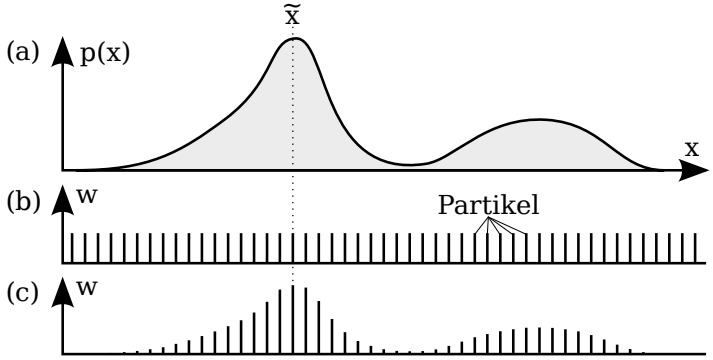


Abbildung 7: Wahrscheinlichkeitsdichten abbilden: Diese Abbildung zeigt das Abbilden von Wahrscheinlichkeitsdichten durch Partikel. Ausgangslage (a) ist eine eindimensionale Wahrscheinlichkeitsdichte deren Funktion $p(x)$ nicht in geschlossener Form vorliegt. Für bestimmte Werte x kann jedoch der Funktionswert errechnet werden. Durch das gleichverteilte "Streuen" einer festgelegten Anzahl an Partikeln in Schritt (b) wird die Abtastrate von $p(x)$ festgelegt. Jedem Partikel ist dadurch ein eindeutiger x-Wert ($\dot{x}^i$) zugeordnet. In Schritt (c) werden die Partikel gewichtet. Jedem Partikel wird eine Gewichtung resultierend aus der Wahrscheinlichkeitsdichtefunktion zugeordnet. Das Gewicht w des Partikels i ergibt sich folgendermaßen: $w^i = p(x = \dot{x}^i)$. Nun repräsentieren die Partikel in Schritt (c) die Wahrscheinlichkeitsverteilung $p(x)$. [sinngemäß [25]]

Systemzustand

Der Systemzustand $p\left(\vec{x}_t \mid \vec{u}_{1\ldots t}, z_{1\ldots t}\right)$ wird durch eine Wahrscheinlichkeitsverteilung dargestellt (siehe Formel 4). Genauer unterliegt er einer bedingten Wahrscheinlichkeit. Um den Ausdruck zu verstehen, wird die Beschreibung von $\vec{u}$ und z vorausgegriffen.

Die Zufallsvariable $\vec{u}$ steht für den Roboter betreffende Messdaten bezüglich seiner eigenen Bewegungen (*control data*). Analog hierzu werden Messdaten über die Umgebung des Roboters mit der Zufallsvariable z bezeichnet (*measurement data*). Der Ausdruck $p\left(\vec{x}_t \mid \vec{u}_{1\ldots t}, z_{1\ldots t}\right)$ beschreibt also eine Betrachtung der Pose des Roboters $\vec{x}_t$ unter Abhängigkeit aller in der Vergangenheit ausgeführten Messungen. Dies kann als Grundidee der Roboterlokalisierung aufgefasst werden – das Herausfinden der eigenen Pose unter Zuhilfenahme verschiedener Messsysteme.

Eine Wahrscheinlichkeitsverteilung kann kaum verwendet werden, um einen Roboter zu steuern. Die letztendliche Posenschätzung $\vec{\widetilde{x}}_t$ ergibt sich erst durch das Maximieren des Systemzustandes über $\vec{x}_t$:

$$\vec{\widetilde{x}}_t = \max_{\vec{x}_t} \underbrace{(p\left(\vec{x}_t \mid \vec{u}_{1\ldots t}, z_{1\ldots t}\right))}_{Systemzustand} \tag{6}$$

Aus dieser Betrachtung wird zudem ersichtlich, dass der Normalisierungsfaktor η in Formel 4 nicht interessant ist. Er skaliert die Wahrscheinlichkeitsdichten um einen konstanten Faktor und hat somit keinen Einfluss auf den Wert von $\vec{\widetilde{x}}_t$.

Da der Systemzustand im Partikelfilter durch einzelne Partikel repräsentiert wird, kann die Pose des Partikels mit dem höchsten Gewicht als resultierende Posenschätzung $\vec{\widetilde{x}}_t$ dienen (siehe auch Kapitel 4.2 und $\widetilde{x}$ in Abbildung 7).

Messung

Der Messschritt des Partikelfilters wird durch den folgenden, aus Formel 4 bekannten Ausdruck dargestellt.

$$\underbrace{p\left(z_t \mid \vec{x}_t\right)}_{Messung} \tag{7}$$

Ausdruck 7 beschreibt die Messwahrscheinlichkeit (engl. *measurement probability*) und beinhaltet das Sensormodell des Filters. Auch hier handelt es sich um eine bedingte Wahrscheinlichkeit. In Worten ausgedrückt beschreibt $p\left(z_t \mid \vec{x}_t\right)$ die Wahrscheinlichkeitsverteilung einer Messung z, vorausgesetzt der Roboter befindet sich zur Iteration t an der Pose $\vec{x}$. Messungen werden in den Partikelfilter integriert, indem der Gewichtungsfaktor der Partikel modifiziert wird. Die Wahrscheinlichkeitsdichte von $\vec{x}_t$ liegt als Partikelwolke vor. Das Gewicht jedes Partikels wird unter Betrachtung von $p\left(z_t \mid \vec{x}_t\right)$ so modifiziert, dass eine hohe Messwahrscheinlichkeit auch das Gewicht erhöht.

Für diese Vorgehensweise muss die Wahrscheinlichkeitsdichtefunktion $p\left(z_t \mid \vec{x}_t\right)$ nicht in geschlossener Lösung vorliegen. Es wird lediglich an bestimmten Posen ihr Zahlenwert errechnet und dieser über das Gewicht der Partikel in den Filter integriert (siehe Kapitel 4.3).

Bewegung

Der Bewegungsschritt des Partikelfilters wird durch den folgenden, aus Formel 4 bekannten Ausdruck dargestellt.

$$\underbrace{p\left(\vec{x}_t \mid \vec{x}_{t-1}, \vec{u}_t\right)}_{Bewegung} \tag{8}$$

Ausdruck 8 beschreibt das odometrische Modell des Filters. Es handelt sich um eine bedingte Wahrscheinlichkeit – genauer um die Wahrscheinlichkeitsverteilung der Roboterpose $\vec{x}$ zum Zeitpunkt t unter Voraussetzung des vorherigen Systemzustands $\vec{x}_{t-1}$ und der aktuellen Messung der Eigenbewegung $\vec{u}_t$ (beispielsweise Odometrie- und IMU-Daten). Aufgabe des odometrischen Modells ist es, unter Eingabe der Eingangsgrößen $\vec{x}_{t-1}$ und $\vec{u}_t$ die neue Wahrscheinlichkeitsverteilung der Roboterpose $\vec{x}_t$ zu bestimmen. Im Falle eines Partikelfilters bedeutet dies das Verschieben jedes Partikels aus der letzten Rekursion des Filters um die

gemessene, zurückgelegte Wegstrecke $\vec{u}_t$ seit der letzten Iteration. Da jede Messung mit einem Messfehler beaufschlagt ist, sorgt das odometrische Modell zusätzlich für eine Modellierung dieser Messungenauigkeit (siehe Kapitel 4.4). Abbildung 8 visualisiert den Bewegungsschritt des Partikelfilters.

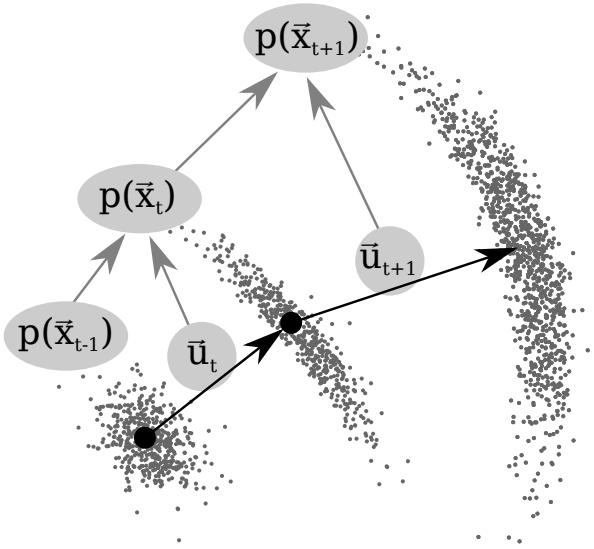


Abbildung 8: Bewegungsschritt Partikelfilter: Der neue Systemzustand resultiert jeweils aus dem letzten Systemzustand und der aktuellen Bewegungsmessung. Durch das Abbilden der Messungenauigkeit durch das odometrische Modell (siehe Kapitel 4.4) nimmt die Streuung der Partikel mit jeder Rekursion zu. [eigene Abbildung]

Resampling

Aufgrund der andauernden Bewegung eines Roboters nimmt die Streuung der Partikelwolke unter Betrachtung des odometrischen Modells stetig zu. Bei einer globalen Lokalisierung müssen initial alle Partikel über den gesamten Ergebnisraum gestreut werden. Beide Mechanismen führen zu weitläufig gestreuten Partikelwolken. Der sogenannte “Resampling“-Schritt sorgt für die Ausdünnung der Partikelwolke an unwahrscheinlichen Aufenthaltsorten des Roboters. Partikel mit niedrigem Gewicht werden an Positionen von Partikeln mit höherem Gewicht verschoben. Dies sorgt für eine Verdichtung der Partikel an “interessanten“ Stellen in der Umgebung. Abbildung 9 zeigt die globale Lokalisierungsphase mit Resampling-Schritt auf Basis einer Umgebungskarte.

Ein starker Vorteil dieser Methode ist eine Erhöhung der Genauigkeit. Durch eine stärkere Partikeldichte an der wahren Pose des Roboters steigt die “Abtastrate“ der Wahrscheinlichkeitsdichtefunktionen und das Modell durch die Partikelwolke wird genauer. Für hohe Genauigkeiten werden weitaus weniger Partikel benötigt, da sie am “Ort des Geschehens“ konzentriert sind. Ein entscheidender Nachteil des Resampling-Schritts ist jedoch die Gefahr der Fehllokalisierung. Bildet der Filter eine einzige Partikelwolke am falschen Ort, ist es nicht ohne Weiteres möglich die wahre Pose wiederzufinden, da nicht in jeder Rekursion der gesamte Ergebnisraum untersucht wird. Nur die Regionen, an denen sich Partikel befinden, werden in Betracht gezogen. Trotz der Nachteile ist der Resampling-Schritt fester Bestandteil des Partikelfilters. Für genauere Erläuterungen zum Resampling-Schritt sei auf Kapitel 4.2 verwiesen.

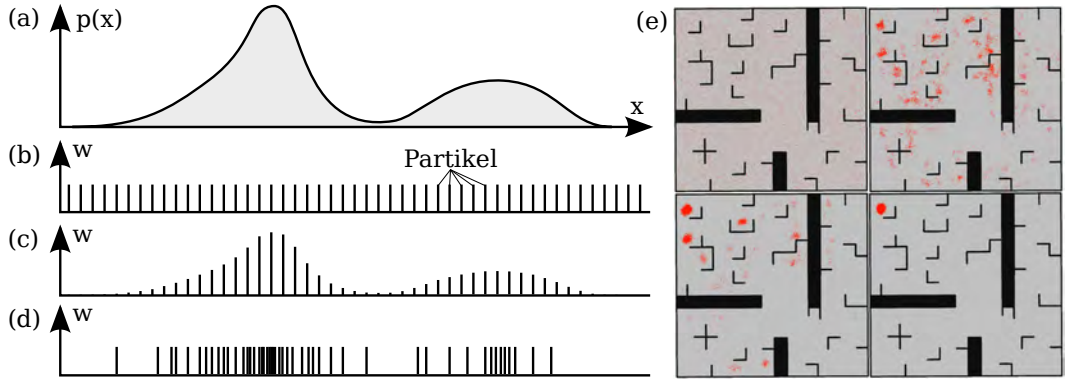


Abbildung 9: Umverteilen der Partikel: Durch das Umlagern von Partikeln mit weniger Gewicht zu Positionen von Partikeln mit mehr Gewicht entstehen "Partikelwolken" in "interessanten" Gebieten. Die Anzahl der Partikel bleibt konstant. Schritt (d) erweitert Abbildung 7 um den sogenannten Resampling-Schritt. Nach dem Resampling-Schritt bildet nicht mehr das Gewicht, sondern die Dichte der Partikel die Wahrscheinlichkeitsdichtefunktion $p(x)$ ab. Illustration (e) zeigt eine globale Lokalisierungsphase mit Resampling (von links oben, nach rechts unten). Die Partikelwolke (rot) "verdichtet" sich an der wahren Pose des Roboters. [a-d: sinngemäß [25]; e: eigene Abbildung]

Clusteranalyse

Die Gewichtung der Partikel entsteht durch den Messschritt und steuert den Resampling-Schritt. Nach dem Resampling-Schritt wird das Gewicht aller Partikel auf einen konstanten Wert gesetzt, sodass die Summe aller Partikel-Gewichte eins ergibt. So bildet nach dem Resampling-Schritt nicht mehr die Gewichtung der Partikel sondern deren Dichte die Verteilungsfunktion der Systemvariable ab (siehe Abbildung 9d). Deshalb verändert sich das Extrahieren der Posenschätzung. Formel 6 gewinnt durch den Resampling-Schritt an Komplexität. Es genügt nun nicht mehr, die Pose des Partikels mit maximalem Gewicht als Posenschätzung zu verwenden. Zur Maximierung des Systemzustandes müsste nun die Dichte der Partikelwolke analysiert werden. Der Schwerpunkt der Partikelwolke stellt eine Möglichkeit dar, aus der Partikelwolke eine Posenschätzung $\vec{\tilde{x}}_t$ zu erhalten. Diese Vorgehensweise findet bei dieser Arbeit Anwendung. Es sei darauf hingewiesen, dass auch durch eine Clusteranalyse der Partikelwolke und anschließende Schwerpunktsbildung der Cluster eine Posenschätzung errechnet werden kann. Angenommen es wird keine Clusteranalyse verwendet und die Partikelwolke verdichtet sich während einer globalen Lokalisierungsphase an zwei unterschiedlichen Posen, dann liegt der Schwerpunkt der Gesamtwolke fälschlicher Weise zwischen den beiden Posen (vgl. Abbildung 26). Wird eine Clusteranalyse verwendet liegt die extrahierte Posenschätzung auf der größeren Partikelwolke. Da in dieser Arbeit die globale Lokalisierungsphase jedoch erst beendet wird, wenn sich ein Großteil der Partikelwolke an einer Pose befindet, kann eine Clusteranalyse ausgelassen werden.

Algorithmus

Nach den vorangegangenen Überlegungen kann nun das Grundgerüst des Algorithmus eines Partikelfilters gezeigt werden (siehe Algorithmus 1). Der rekursive Charakter kommt durch die Arbeit mit immer derselben Partikelwolke zustande. Die aus Rekursion $t-1$ entstehende Partikelwolke wird in Rekursion t wieder für den Bewegungs-, Mess- beziehungsweise Resampling-Schritt verwendet.

Algorithmus 1 Partikelfilter

```
1:  P = initialisierePartikelwolke(AnzahlPartikel)   ▷ Global oder mit Initialposenvorgabe
2:  while 1 do                                                          ▷ Hauptschleife
3:      if Odometrie.neueMessung() then
4:          Bewegungsmodell.verarbeite(Odometrie, P)                ▷ Verschiebe Partikel
5:      end if
6:
7:      for all Sensoren do          ▷ Mehrere Sensormodelle können integriert werden
8:          if Sensor.neueMessung() then
9:              Sensormodell.verarbeite(Sensor, P)        ▷ Modifiziere Gewicht der Partikel
10:         end if
11:     end for
12:
13:     if Resampler.Timer.abgelaufen() then
14:         Resampler.resample(P)                          ▷ Neuverteilung der Partikel
15:         Resampler.Timer.neustart()
16:     end if
17:
18:     return extrahierePosenschätzung(P)   ▷ Clusteranalyse || Schwerpunktsberechnung
19: end while
```

Letztendlich implementiert der Algorithmus die Formel 9. Sie unterscheidet sich von Formel 4 nur durch den Teil der Messung. Falls mehrere Sensoren und deren Sensormodelle vorhanden sind, können diese auch in den Filter integriert werden. Das kann bei richtiger Wahl der Sensorik zur Verbesserung der Lokalisierung führen, da die Umgebung besser aufgenommen werden kann.

$$\underbrace{p\left(\vec{x}_t \mid (\cdot)\right)}_{Systemzustand} = \eta \cdot \underbrace{p\left(\vec{x}_{t-1} \mid (\cdot)\right)}_{Rekursion} \cdot \underbrace{p\left(\vec{x}_t \mid \vec{x}_{t-1}, \vec{u}_t\right)}_{Bewegung} \cdot \underbrace{p\left(z_t^1 \mid \vec{x}_t\right)}_{Messung\ 1} \cdot \ldots \cdot \underbrace{p\left(z_t^n \mid \vec{x}_t\right)}_{Messung\ n} \tag{9}$$

4.2 Resampling-Algorithmus

Der Resampling-Schritt sorgt für die Neuverteilung der Partikel. Der folgende Abschnitt beschreibt den dazu verwendeten Algorithmus in eigenen Worten. Die folgende Herangehensweise stammt aus [25] und [15].

Der Resampling-Prozess kann im stochastischen Sinn als “Ziehen mit Zurücklegen“ betrachtet werden. Aus der zum Zeitpunkt des Resampling-Schritts vorliegenden Partikelwolke wird durch das “Ziehen mit Zurücklegen“ eine neue Partikelwolke mit gleicher Partikelanzahl generiert. Dabei kann jeder Partikel mit einer seinem Gewicht proportionalen Wahrscheinlichkeit gezogen werden. Das Drehen einer Drehscheibe stellt einen solchen Prozess dar. Bei jeder Drehung stoppt die Drehscheibe an einem Partikel. Dieses wird der neuen Partikelwolke hinzugefügt. Das Drehen der Drehscheibe wird so lange wiederholt, bis die neue Partikelwolke die gewünschte Größe hat.

Abbildung 10a zeigt eine solche Drehscheibe, um den Prozess zu verdeutlichen. Partikel mit höherem Gewicht erhalten eine größeres Segment auf der Drehscheibe und werden deshalb statistisch öfter gezogen. Jedem “gezogenen“ Partikel wird ein konstantes Gewicht zugewiesen. So haben nach dem Resampling-Schritt alle Partikel das selbe Gewicht.

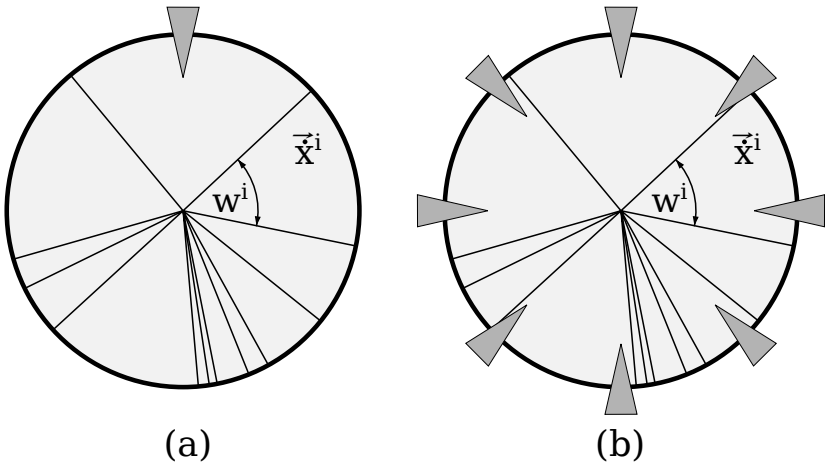


Abbildung 10: Resampling: (a) Standard-Resampling verdeutlicht an einer Drehscheibe. Jedes Kreissegment steht für ein Partikel aus der aktuellen Partikelwolke. Die Größe der Segmente ist proportional zum Gewicht w^i des jeweiligen Partikels $\vec{x}^i$ ausgelegt. Das feststehende Dreieck an der Oberseite der Drehscheibe zeigt an, welches Segment in der jeweiligen Iteration ausgewählt wird. Die Drehscheibe wird so oft gedreht, bis genügend “neue“ Partikel selektiert wurden. (b) Folgt dem selben Prinzip, nur sind in diesem Fall mehrere feststehende Anzeiger angebracht. Mit jeder Drehung werden acht Partikel in die “neue“ Partikelwolke gewählt. [sinngemäß [25]]

Eine Erweiterung des Standard-Resampling-Prozesses stellt das low-variance-resampling (siehe [25]) dar. Hier wird bei jeder Drehung der Drehscheibe nicht ein Segment, sondern mehrere Segmente in äquidistantem Abstand selektiert (siehe Abbildung 10b). Dies sorgt für eine höhere Chance, Partikel mit kleineren Gewichten zu ziehen, da in jedem Schritt Partikel aus allen “Bereichen“ der Scheibe gezogen werden.

Der in dieser Arbeit entwickelte Partikelfilter kann mit beiden Resampling-Methoden betrieben werden. Für das Testen des Filters wird das Standard-Resampling verwendet, da es besser reproduzierbare und dabei ausreichend gute Ergebnisse liefert.

4.3 Sensormodell 2D-Laserscanner

Im Messschritt des Filters wird die Information der Messung eines Sensors über das Gewicht der Partikel in den Filter integriert. Das Gewicht w^i eines einzelnen Partikels $\vec{x}_t^i$ wird hierfür mit der Messwahrscheinlichkeit $p\left(z_t \mid \vec{x}_t\right)$ (vgl. Ausdruck 7) der aktuellen Messung $\hat{z}_t$ multipliziert (siehe Formel 10).

$$w^i = w^i \cdot p\left(z_t = \hat{z}_t \mid \vec{x}_t = \vec{x}_t^i\right) \tag{10}$$

Das Abbilden der Messwahrscheinlichkeit wird am Beispiel eines 2D-Laserscanners erklärt. Die Herangehensweise stammt aus [25, 26] und wird nachfolgend mit eigenen Worten erläutert.

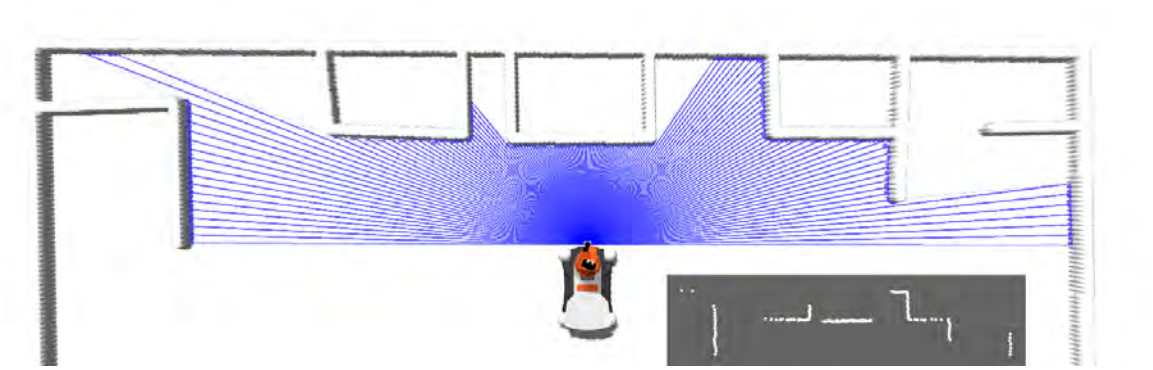


Abbildung 11: 2D-Laserscanner: Mittels einer Simulation erstellte Abbildung einer 2D-Laserscanner-Messung. An der Vorderseite des Robotermodells ist ein 2D-Laserscanner montiert. Die Umgebung wird mit einzelnen Laserstrahlen (blau) abgetastet. Das graue Feld rechts unten zeigt die resultierende Messung (weiß), wie sie der Roboter erhält. [eigene Abbildung]

2D-Laserscanner tasten die Umgebung in einer Ebene, der sogenannten Scannebene, mit einzelnen Entfernungsmessungen ab. Abbildung 11 zeigt die Simulation der Laserstrahlen eines 2D-Laserscanners in der Umgebung. Die Strahlen werden von Hindernissen in der Umgebung reflektiert. Messgröße eines 2D-Laserscanners sind die Entfernungen zu den Auftreffpunkten der einzelnen Laserstrahlen. Abbildung 11 (rechts unten) visualisiert eine typische Messung aus einem 2D-Laserscanner. Jeder Punkt steht für den Auftreffpunkt eines Laserstrahls. Stellen, an denen Laserstrahlen reflektiert werden, stellen potentielle Hindernisse für einen Roboter dar.

Eine Messung aus einem 2D-Laserscanner besteht aus n Entfernungsmessungen z_t^j wobei $j \in [1; n]$. Statistisch werden die Einzelmessungen einer Messung voneinander unabhängig betrachtet. So lässt sich die Messwahrscheinlichkeit der Gesamtmessung als Multiplikation der Einzelmessungen beschreiben:

$$p\left(z_t \mid \vec{x}_t\right) = \prod_{j=1}^{n} p\left(z_t^j \mid \vec{x}_t\right) \tag{11}$$

Der Ausdruck $p\left(z_t^j \mid \vec{x}_t\right)$ beschreibt die Wahrscheinlichkeitsdichte des Auftretens einer einzelnen Entfernungsmessung unter der Voraussetzung, dass sich der Laserscanner zum Zeitpunkt t an der Pose $\vec{x}$ befindet. Um Aussagen über diese bedingte Wahrscheinlichkeit treffen zu können, wird eine Umgebungskarte beziehungsweise eine Wahrscheinlichkeitskarte verwendet.

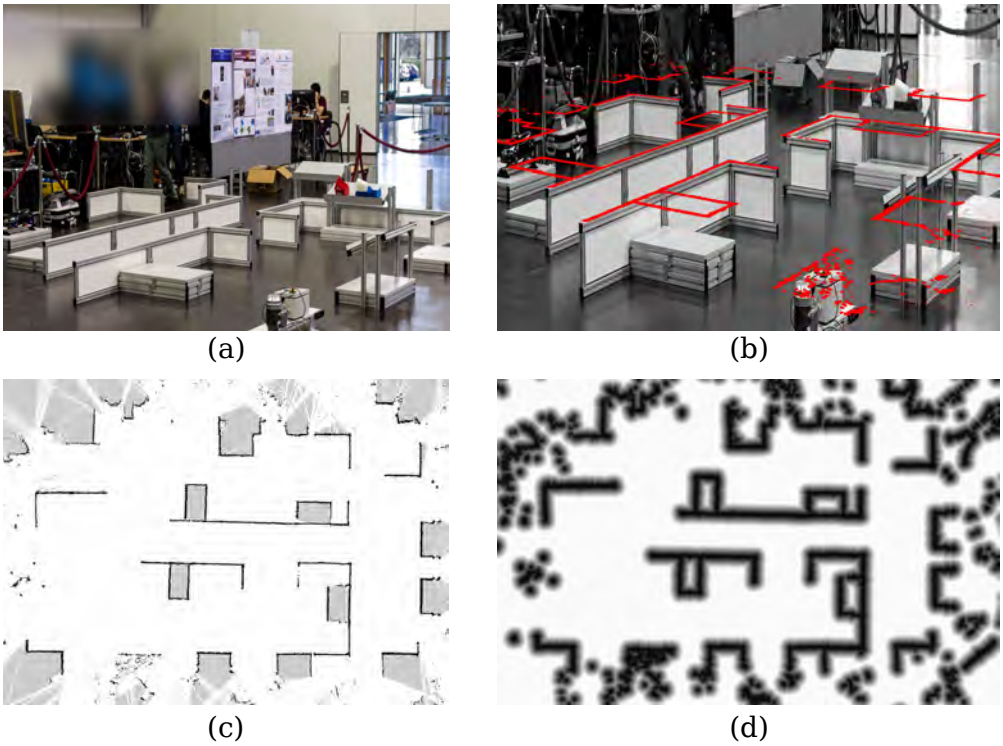


Abbildung 12: Abbild der Umgebung: (a) Die reale Umgebung – das Einsatzgebiet eines zu lokalisierenden Roboters. In diesem Fall die Arena des RoboCup@Work Workshops 2016 in Magdeburg (b) Das durch einen 2D-Laserscanner und SLAM erstellte Abbild der Umgebung (rot) in die reale Umgebung hinein projiziert. Die Höhe der Projektion entspricht nicht der Scannebene. In Wirklichkeit liegt diese nur wenige Zentimeter über dem Boden. (c) Die mittels SLAM erstellte Umgebungskarte (Occupancy-Grid-Map) in der Form, in der sie dem Lokalisierungssystem zugeführt wird. (d) Die aus der Umgebungskarte berechnete Wahrscheinlichkeitskarte. [eigene Abbildung]

Die Abbildungen 12a bis 12c visualisieren die Entstehung einer Umgebungskarte durch einen 2D-Laserscanner und SLAM. Es entsteht ein 2D-Abbild (eine Art Grundriss) der Umgebung in Höhe der Scannebene. Um das hier beschriebene und verwendete Sensormodell verwenden zu können, muss eine solche Umgebungskarte der Einsatzumgebung vorliegen. Schwarze Bereiche in der Umgebungskarte repräsentieren belegte Bereiche und somit Hindernisse in der Umgebung. Im Optimalfall reproduziert die Messung aus einem Laserscanner die zuvor kartierte Umgebung und alle Entfernungsmessungen treffen auf Übergängen von freien zu belegten Bereichen auf. Abbildung 13a visualisiert diesen Fall. Durch Messrauschen der Entfernungsmessungen, Fehlmessungen und dynamische Hindernisse tritt dieser Fall jedoch selten auf. Abbildung 13b zeigt ein realistischeres Abbild einer 2D-Laserscannermessung. Der Großteil der Einzelmessungen liegt nicht mehr auf den erwarteten Positionen, sondern nur

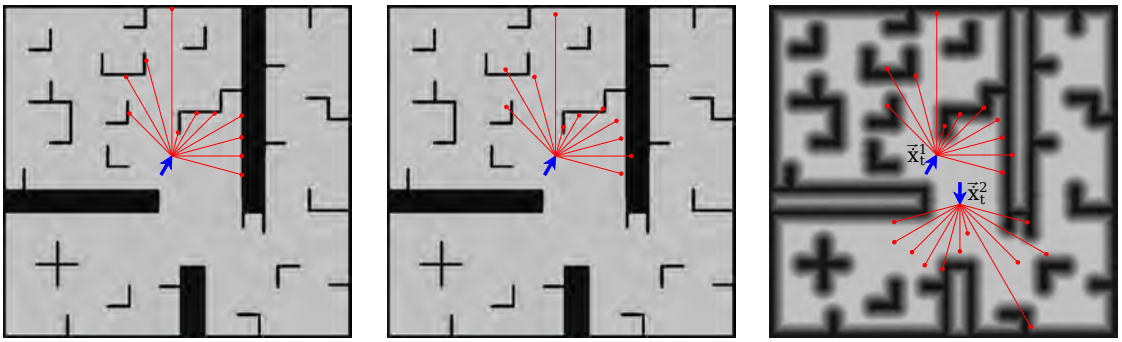


Abbildung 13: Wahrscheinlichkeitskarte: (a) Eine Laserscannermessung ohne Messrauschen und unter der Voraussetzung, dass die abgebildete Umgebungskarte die Umgebung perfekt widerspiegelt. Der Pfeil zeigt die wahre Pose des Sensors, die Linien zeigen die Laserstrahlen. (b) Die reale Messung eines Laserscanners. Die Entfernungsmessungen sind verrauscht und decken sich nicht vollständig mit der Umgebungskarte. (c) Eine Wahrscheinlichkeitskarte, in der das Messrauschen eines Laserscanners berücksichtigt wird, indem die Kanten unscharf abgebildet sind, auf denen das Auftreffen der Entfernungsmessungen erwartet wird. Visualisiert ist zudem die Bewertung zweier Partikel $\vec{x}_t^1$ und $\vec{x}_t^2$ anhand der Laserscannermessung und der Wahrscheinlichkeitskarte. Die Messpunkte werden unter Betrachtung der Posen der beiden Partikel transformiert. Treffen die Laserstrahlen vorwiegend in dunklen Bereichen der Wahrscheinlichkeitskarte auf (siehe $\vec{x}_t^1$), wird das Gewicht des Partikels mit einem höheren Faktor multipliziert. Treffen viele Laserstrahlen in hellen Bereichen auf (siehe $\vec{x}_t^2$), wird das Gewicht des Partikels mit einem niedrigeren Faktor multipliziert. Durch die Erhöhung des Gewichts wird der Partikel an der wahren Pose $\vec{x}_t^1$ "überleben". Der falsch positionierte Partikel $\vec{x}_t^2$ wird durch sein niedrigeres Gewicht im Resamplingschritt eliminiert. [eigene Abbildung]

noch in deren Nähe. Dieses Verhalten wird von den in Abbildung 12d und Abbildung 13c gezeigten Wahrscheinlichkeitskarten (m_p) berücksichtigt. Die Kanten, auf denen das Auftreffen der Entfernungsmessungen erwartet wird, werden unscharf abgebildet.

Auf Basis dieser Wahrscheinlichkeitskarte kann ein Wert proportional zur Messwahrscheinlichkeit für jede Entfernungsmessung und jedes Partikel ermittelt werden. Dies geschieht durch das Abrufen des Wertes der Wahrscheinlichkeitskarte am Auftreffpunkt der Entfernungsmessung. Je höher der (Grau-)Wert am Auftreffpunkt, desto wahrscheinlicher ist die Entfernungsmessung, da sie nahe an einem Hindernis liegt.

Verschiedene Partikel repräsentieren verschiedene Posenannahmen des Roboters. So ergeben sich für verschiedene Partikel auch verschiedene Auftreffpunkte der Entfernungsmessungen in der Umgebungskarte und damit unterschiedliche Messwahrscheinlichkeiten. Um mit diesem Modell die Messwahrscheinlichkeit für ein bestimmtes Partikel $\vec{x}_t^i$ berechnen zu können, müssen die Messpunkte der aktuellen Laserscannermessung $\hat{z}_t$ an die Pose des zu bewertenden Partikels verschoben (transformiert) werden. Dabei muss die Robotergeometrie beachtet werden, da $\vec{x}_t^i$ nicht die Pose des Laserscanners, sondern die Pose des Roboters beschreibt. Abbildung 12d zeigt schematisch zwei Partikel in der Umgebung. Jedem Partikel wird die aktuelle Scanner-Messung zugeordnet. Das Aufmultiplizieren der Grauwerte an den so entstandenen Messpunkten in der Wahrscheinlichkeitskarte ergibt $p\left(z_t = \hat{z}_t \mid \vec{x}_t = \vec{x}_t^i\right)$ und somit den das Gewicht modifizierenden Faktor für jedes einzelne Partikel (siehe Formel 10).

Werden einem Partikel überwiegend Entfernungsmessungen in unbelegten Bereichen der Umgebungskarte zugeordnet (Partikel $\vec{x}_t^2$ in Abbildung 13c), sinkt sein Gewicht. Damit steigt die Wahrscheinlichkeit, dass es im nächsten Resampling-Schritt nicht mehr "gezogen" beziehungsweise ausgewählt wird.

Die Wahrscheinlichkeitskarte wird einmalig beim Start des Lokalisierungssystems aus der bereitgestellten Umgebungskarte errechnet. Weitere Informationen zur Berechnung der Wahrscheinlichkeitskarte können [25] entnommen werden.

4.4 Omnidirektionales Bewegungsmodell

Das Bewegungsmodell sorgt für die Bewegung, also das Verschieben, der Partikel. Zusätzlich bildet das Bewegungsmodell die Messunsicherheit der Sensoren ab, die die Eigenbewegung messen. Das Modell eines Differenzialantriebs ist aus [25] entnommen. Das für diese Arbeit selbst erstellte beziehungsweise implementierte Modell eines omnidirektionalen Antriebs wird nachfolgend beschrieben.

Die Odometriedaten eines Roboters liegen in Form einer absoluten Pose vor. Diese entsteht durch Aufsummieren aller Odometriemessungen über die Zeit. Sie ist meist mit einem großen Driftfehler beaufschlagt und stellt die wahre Pose des Roboters deshalb unzureichend genau dar. Zum Verschieben der Partikel wird deshalb die relative Veränderung der aus den Odometriedaten berechneten Pose zwischen zwei Messungen oder Zeitpunkten verwendet (siehe [1] – Relative Odometrie).

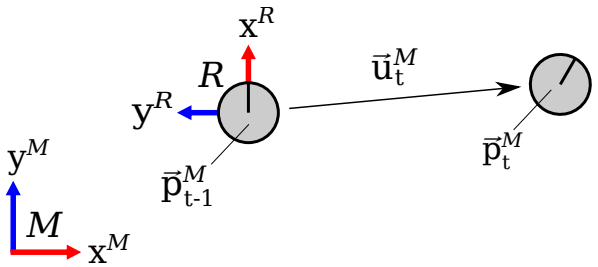


Abbildung 14: Omnidirektionales Bewegungsmodell: Visualisierung des Bewegungsschritts eines Roboters mit omnidirektionaler Kinematik von einer Pose $\vec{p}_{t-1}^{\,M}$ zur nächsten $\vec{p}_t^{\,M}$. Es gilt $\vec{p}_t^{\,M} = \vec{p}_{t-1}^{\,M} + \vec{u}_t^{\,M}$. Die Bezeichner M (Map) und R (Roboter) beschreiben das feststehende Karten-Koordinatensystem und das bewegliche Roboter-Koordinatensystem. Der Verschiebungsvektor $\vec{u}_t^{\,M}$ beschreibt die Differenz der beiden Posen im Kartenkoordinatensystem M. [eigene Abbildung]

Abbildung 14 zeigt zwei Posen zu verschiedenen Zeitpunkten. Alle Roboterposen werden auf das feststehende Kartenkoordinatensystem M bezogen. Der Verschiebungsvektor $\vec{u}_t^{\,M}$ entsteht durch Subtraktion der Pose $\vec{p}_{t-1}^{\,M}$ von $\vec{p}_t^{\,M}$ und repräsentiert die aktuelle Bewegungsmessung. Wird dieser für das in Abbildung 14 gezeigte Szenario gebildet, entsteht hauptsächlich eine Verschiebung in x^M-Richtung. Der Roboter bewegt sich in seinem eigenen Koordinatensystem R jedoch nicht in x-Richtung[6], sondern seitlich.

Bei einer omnidirektionalen Kinematik kann es sinnvoll sein, für verschiedene Bewegungsrichtungen verschiedene Messunsicherheiten abzubilden. Beispielsweise kann der Schlupf von Mecanum-Rädern bei Bewegungsrichtungen zur Radachse höher als bei Bewegungen senkrecht zur Radachse[7] sein.

Um das Messrauschen abhängig von der Bewegungsrichtung des Roboters auszulegen, muss der Verschiebungsvektor zuerst in das Roboterkoordinatensystem transformiert, beziehungsweise "gedreht" werden. Die Pose $\vec{p}_{t-1}^{\,M}$ beinhaltet den hierfür benötigten Drehwinkel Φ_{t-1}^M:

$$\vec{p}_{t-1}^{\,M} = \left(x_{t-1}^M, y_{t-1}^M, \Phi_{t-1}^M\right)^T$$

Die Drehung von $\vec{u}_t^{\,M}$ um $-\Phi_{t-1}^M$ ergibt den Verschiebungsvektor auf das Roboterkoordinatensystem bezogen:

$$\vec{u}_t^{\,R} = rot\left(-\Phi_{t-1}^M\right) \cdot \vec{u}_t^{\,M} = \left(\Delta x_t^R, \Delta y_t^R, \Delta\Phi_t^R\right)^T$$

[6] In der Robotik beschreibt eine Bewegung in x-Richtung meist eine Bewegung nach vorne in Richtung Stirnseite des Roboters

[7] Vorausgesetzt, alle Radachsen verlaufen Parallel zueinander. Beispielroboter für diese Anordnung: KUKA youBot (`http://www.youbot-store.com/`)

Nun kann $\vec{u}_t^R$ mit dem gewünschten Messrauschen versehen werden. Die Parameter a_1, a_2 und a_3 skalieren dabei das Messrauschen bei einer Vorwärts/Rückwärts-, Seitwärts- und Drehbewegung des Roboters. Zudem wird das Messrauschen von der Differenz der beiden Posen skaliert. Je weiter sich der Roboter zwischen zwei Messungen bewegt hat, desto höher wird das Messrauschen ausgelegt. Gleichung 12 zeigt dies, wobei der Ausdruck $sampleFrom(\mathcal{N}(a;b))$ die Stichprobe einer Normalverteilung mit $\mu = a$ und $\sigma^2 = b$ darstellt.

$$\hat{\vec{u}}_t^R = \begin{pmatrix} \Delta x_t^R + sampleFrom(\mathcal{N}(0; a_1 \cdot \Delta x_t^R)) \\ \Delta y_t^R + sampleFrom(\mathcal{N}(0; a_2 \cdot \Delta y_t^R)) \\ \Delta \Phi_t^R + sampleFrom(\mathcal{N}(0; a_3 \cdot \Delta \Phi_t^R)) \end{pmatrix} \tag{12}$$

Der Verschiebungsvektor $\hat{\vec{u}}_t^R$ beschreibt die mit zufälligem Messrauschen beaufschlagte Verschiebung im Roboterkoordinatensystem. Die Posen der einzelnen Partikel liegen auch bezogen auf das Kartenkoordinatensystem vor. Durch Drehung des Verschiebungsvektors um den Orientierungswinkel ($\Phi^{n,M}$) des Partikels n, wird die Verschiebung des Partikels bezüglich des Kartenkoordinatensystems errechnet:

$$rot\left(\Phi^{n,M}\right) \cdot \hat{\vec{u}}_t^R = \hat{\vec{u}}_t^{n,M}$$

Die neue Pose des Partikels kann nun durch eine Addition der alten Partikelpose mit dem Verschiebungsvektor berechnet werden:

$$\vec{x}_t^n = \vec{x}_{t-1}^n + \hat{\vec{u}}_t^{n,M}$$

Jedes Partikel wird auf diese Art um die relative Posenänderung aus den empfangenen Odometriedaten verschoben. Abbildung 8 zeigt die Ausbreitung einer Partikelwolke mithilfe eines odometrischen Modells.

5 Entwurf und Implementierung

Nach der Definition der Projektanforderungen und der anschließenden Erläuterung der theoretischen Grundlagen kann der ausgewählte Lösungsansatz verdeutlicht werden. Nachfolgende Abschnitte beschreiben diesen sowie den Entwurf und die darauffolgende Implementierung der Software für das Lokalisierungssystem.

5.1 Lösungsansatz

Der Lösungsansatz für das zu entwickelnde Lokalisierungssystem resultiert aus den in der Anforderungsliste festgehaltenen Vorgaben (Tabelle 2) und den Erkenntnissen aus dem Literaturteil (Kapitel 3).

Sensorkonzept

Wie aus dem Literaturteil Kapitel 3 ersichtlich, können verschiedene Sensorsysteme für die Roboterlokalisierung verwendet werden. Durch Sensordatenfusion können auch mehrere Sensorsysteme kombiniert werden. Der flexible Einsatz des Lokalisierungssystems in industriellem Umfeld stellt Anforderungen an die verwendeten Sensoren. Hieraus resultiert das grundlegende Sensorkonzept des Lokalisierungssystems.

In industriellem Umfeld herrschen oft raue Bedingungen. Staub und Flüssigkeitspartikel in der Luft können Kamerasysteme sehr schnell beeinträchtigen. Sie sind daher nicht für einen Einsatz im industriellen Umfeld geeignet, beziehungsweise können nur zusätzlich zum Grundkonzept verwendet werden (siehe Verwendung CeilCam-System – Kapitel 6.1).

Für manche Sensorsysteme müssen Veränderungen an der Umgebung, wie das Anbringen von Marken (RFID, QR) vorgenommen werden. Ein markenbasiertes Sensorsystem kann nicht ad-hoc verwendet und deshalb den hohen Flexibilitätansprüchen in der modernen Fertigung nicht gerecht werden.

Auch Systeme, die auf Funksignale angewiesen sind (WLAN, UWB), können nur unterstützend eingesetzt werden, da die flächendeckende Bereitstellung eines solchen Signals eine zu starke Anforderung an die Umgebung darstellt. Dies wiederum würde die Flexibilität und Anwendbarkeit des Systems herabsetzen.

Odometriedaten sollten auf jedem FTS oder Robotersystem vorhanden sein. Auch eine IMU kann mit minimalem Aufwand und geringen Kosten in jedes bestehende System integriert werden. Beide Systeme sind sehr widerstandsfähig gegen Störungen aus der Umgebung und können so auch gut in industriellem Umfeld eingesetzt werden.

Aus sicherheitstechnischen Gründen muss ein FTS (noch) mit einem kostenintensiven 2D-Sicherheitslaserscanner ausgestattet werden. Dieser kann zusätzlich zum Lokalisieren und zum Kartieren der Umgebung verwendet werden. Bei steigendem Bedarf fahrerloser Transportsysteme könnten die Sicherheitsregelungen dahingehend angepasst werden, dass kostensparende Sensoren, beispielsweise Radar-Sensoren anstelle von kostenintensiven 2D-Sicherheitslaserscannern die Sicherheit gewährleisten. So können auch kostensparende Laserscanner[8] anstatt teurer Sicherheitslaserscanner verbaut und zur Lokalisierung eingesetzt werden (siehe Kobuki-Flotte – Kapitel 6.4).

[8] SLAMTEC RPLIDAR ≈ 380 €, NEATO XV-11 ≈ 150 €; Stand 10/2016

Resultierend aus diesen Überlegungen wird folgende Sensorkombination zur Lokalisierung festgelegt: IMU, Odometrie und ein 2D-Laserscanner [→ Anforderungsliste]. Damit wird ein einfach zu integrierendes Lokalisierungssystem, welches nicht auf externe Markierungen angewiesen ist (ad-hoc-Lokalisierungssystem), realisiert. Im Laborumfeld, beziehungsweise nach Anpassungen der Sicherheitsregelungen bei FTS, kann auch der "Low-Cost"-Gedanke mit diesem Sensorkonzept bedient werden. Die Festlegung auf diese Sensoren soll die Möglichkeit weitere Sensoren zu integrieren nicht ausschließen.

Fusionsalgorithmus

Werden verschiedene Sensoren zur Lokalisierung verwendet, müssen diese durch einen Fusionsalgorithmus vereint werden. Aus Kapitel 3 geht hervor, dass der Partikelfilter-Algorithmus hierfür eine sehr flexible Methode darstellt. Er setzt voraus, dass mindestens zwei Messsysteme vorhanden sind: eines, das die Eigenbewegung des zu lokalisierenden Objektes misst und eines, das Informationen aus der Umgebung des zu lokalisierenden Objektes bereitstellt. Nach der Implementierung eines Sensor- beziehungsweise Bewegungsmodells für das vorliegende Sensorkonzept (siehe Kapitel 4.3 und 4.4) kann der Partikelfilter-Algorithmus allen Anforderungen gerecht werden, weshalb er als Fusionsalgorithmus für die Problemstellung dieser Arbeit gewählt wird.

Gesamtkonzept

Abbildung 15 zeigt das entwickelte Gesamtkonzept des Lokalisierungssystems.

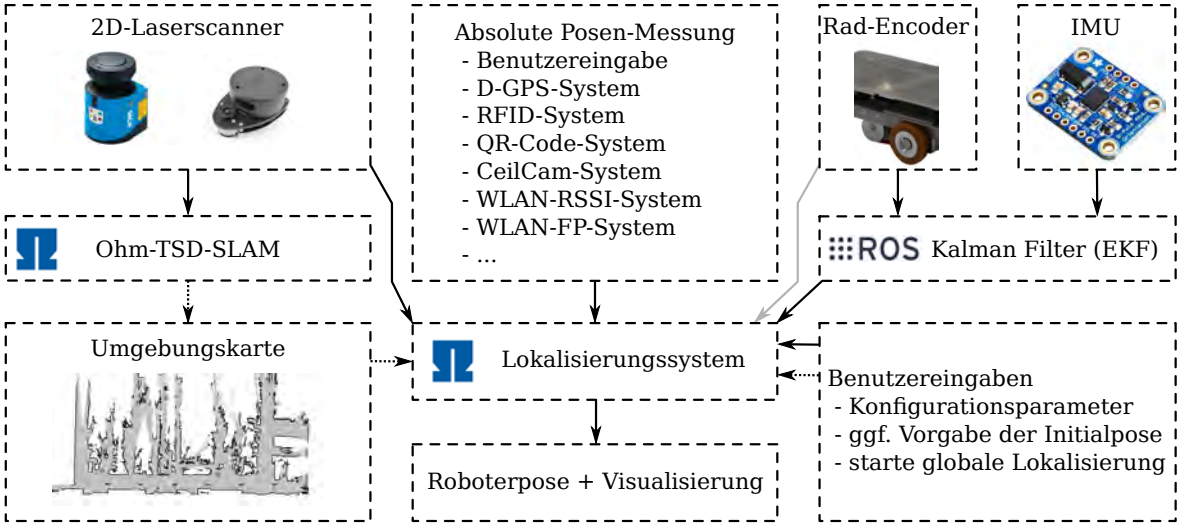


Abbildung 15: Gesamtkonzept Roboterlokalisierung: Die Pfeilstriche visualisieren den Datenfluss. Gepunktete Pfeilstriche repräsentieren einen einmaligen Datenfluss, durchgezogene Pfeilstriche repräsentieren einen kontinuierlichen Datenfluss während der Lokalisierungsphase. Die Odometriedaten können dem Lokalisierungssystem entweder über Rad-Encoder oder über ein fusioniertes Odometriesignal bereitgestellt werden. In der Abbildung ist die Fusion von Rad-Encoder- und IMU-Daten mittels eines Kalman Filters dargestellt. [eigene Abbildung]

Durch die Fusion von 2D-Laserscanner-, Odometrie- und absoluten Posendaten kann das Lokalisierungssystem eine Schätzung der Roboterpose bereitstellen. Hierfür benötigt es ein Abbild der Umgebung in Form einer Umgebungskarte (vgl. Kapitel 4.3). Diese muss vor der

Lokalisierungsphase einmalig aus der realen Einsatzumgebung erstellt werden. Hierfür wird zum Beispiel der an der Technischen Hochschule entwickelte SLAM-Algorithmus ohm-tsd-slam[9] und ein 2D-Laserscanner verwendet. Die Odometriedaten können direkt von Rad-Encodern abgegriffen werden. Es empfiehlt sich jedoch eine vorgelagerte Fusion der Rad-Encoder-Daten mit einer auf dem Roboter montierten IMU (vgl. [1]). Als Fusionsalgorithmus können die ROS-Pakete robot-pose-ekf[10] oder robot-localization[11] verwendet werden.

[9] `wiki.ros.org/ohm_tsd_slam`
[10] `wiki.ros.org/robot_pose_ekf`
[11] `wiki.ros.org/robot_localization`

5.2 Verwendete Entwurfsmuster

Da es sich um ein überwiegend softwarebasiertes Projekt handelt, wird die Sprache UML als Modellierungswerkzeug des Entwurfs gewählt. Ein auf diesem Weg entstandenes Klassendiagramm kann gut in der Programmiersprache C++ umgesetzt werden. Der so entstandene Entwurf der Software ist diesem Dokument angehängt (Siehe Abbildung 44).

Beim Entwerfen von Software können Entwurfsmuster helfen, gängige Strukturen zu überschauen beziehungsweise passende Strukturen für eine Problemstellung zu finden. Gamma u. a. stellen in ihrem Nachschlagewerk [10] eine Sammlung von Entwurfsmustern zur Verfügung. Die unter Zuhilfenahme dieser Sammlung ausgewählten Softwareentwurfsmuster werden nachfolgend vorgestellt.

Verwendung von Schnittstellen (Interfaces)

Aus den Anforderungen geht hervor, dass die Software in Form einer Softwarebibliothek ausgelegt werden muss. Schnittstellen stellen die Verbindung zwischen Softwarebibliothek und Nutzer der Bibliothek dar. Sie erhöhen die Integrierbarkeit des Systems und ermöglichen den Schutz der Implementierung. Nachfolgende Abbildung verdeutlicht die Implementierung einer Schnittstelle am Beispiel der Laser-Updater-Klasse.

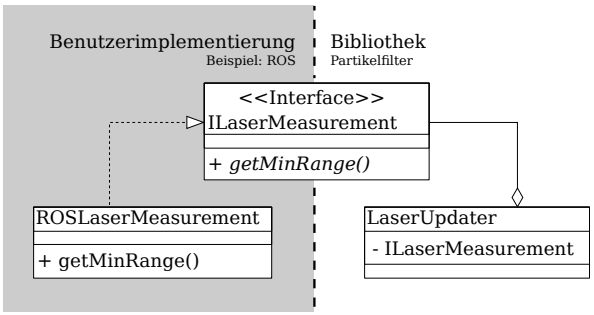


Abbildung 16: Verwendung von Schnittstellen: Der Benutzer versorgt Bibliotheksfunktionen über eine Schnittstelle mit Daten. Hier am Beispiel der Laser-Updater-Klasse gezeigt. [eigene Abbildung]

Die Klasse *LaserUpdater* ist für das Management der Messdaten aus einem 2D-Laserscanner zuständig. Sie ist in der im Zuge dieser Arbeit entwickelten Bibliothek definiert und verlangt nach einem Objekt, welches die Schnittstelle *ILaserMeasurement* bereit stellt. In dieser Schnittstelle sind alle für die Bibliothek interessanten Funktionen bezüglich der Messdaten aus einem Laserscanner abstrakt definiert. Der Benutzer muss das Objekt zur Laufzeit an die Bibliothek übergeben.

Auf diese Weise muss der Benutzer die Klasse *LaserUpdater* nicht kennen, sowie die Bibliothek zur Kompilierzeit *ROSLaserMeasurement* nicht kennen muss. Die Implementierung kann vor dem Benutzer geheim gehalten werden und zugleich kann der Benutzer seine eigenen Sensordaten integrieren. Des Weiteren kann die Bibliothek so von ROS und anderen Systemen unabhängig gehalten werden.

Komposition der Updater-Klassen

Die Updater-Klassen aktualisieren die Partikelwolke des Filters, indem sie neue Messdaten integrieren und dabei die Gewichtung der Partikel oder die Partikel selbst modifizieren. Alle

Updater-Klassen werden in der Klasse *FilterController* gruppiert. Diese Struktur wird “Komposition“ genannt. In diesem Fall ist dies eine “Zusammenstellung“ verschiedener Elemente oder, genauer gesagt, verschiedener Updater-Klassen (siehe Abbildung 17).

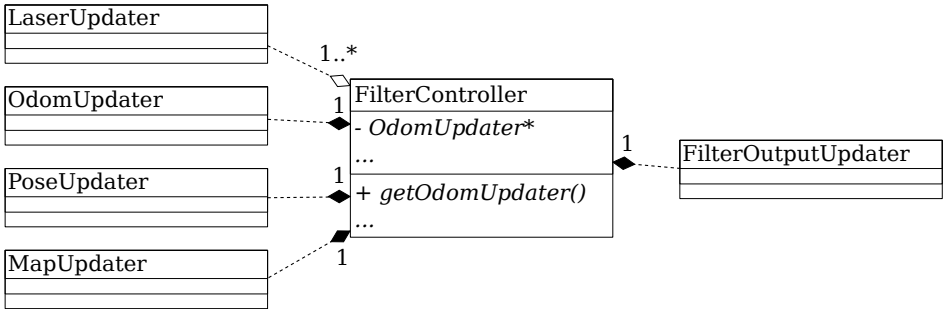


Abbildung 17: Komposition der Updater-Klassen: Die Klasse *FilterController* ist eine Komposition aus verschiedenen Updater-Klassen. [eigene Abbildung]

Strukturmuster Dekorierer

Bei der Implementierung der Wahrscheinlichkeitskarte (siehe Kapitel 4.3) wird ein Entwurfsmuster namens Dekorierer (*decorator*) verwendet. Da die Wahrscheinlichkeitskarte aus einer Occupancy-Grid-Map resultiert, würde normalerweise von einer Occupancy-Grid-Map geerbt werden, um die zusätzlichen Funktionen der Wahrscheinlichkeitskarte zu implementieren. Die Occupancy-Grid-Map wird jedoch vom Benutzer zur Verfügung gestellt und ist somit der Bibliothek zur Kompilierzeit noch nicht bekannt. Nur die vom Benutzer bereitzustellende Schnittstelle (*IMap*) ist der Bibliothek zur Kompilierzeit bekannt. Eine Lösung hierfür ist ein Dekorierer. Er “dekoriert“ ein zur Kompilierzeit unbekanntes, nur durch eine Schnittstelle definiertes Objekt mit neuen Funktionen. In diesem Fall wird die vom Benutzer zugeführte Occupancy-Grid-Map (*ROSMap*) mit der Funktion *getProbability*() “dekoriert“. Abbildung 18 verdeutlicht diesen Zusammenhang.

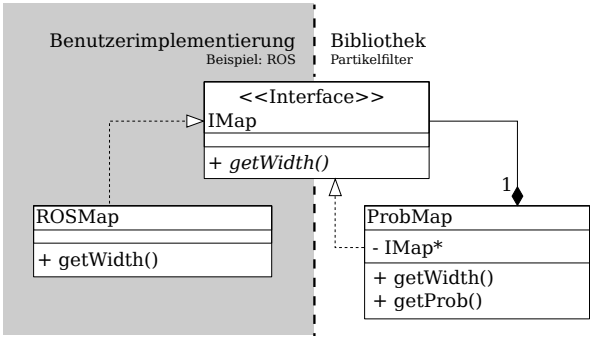


Abbildung 18: Strukturmuster Dekorierer: Die Wahrscheinlichkeitskarte (*ProbMap*) implementiert die Schnittstelle *IMap* selbst und hält zugleich eine Referenz auf ein weiteres, die Schnitstelle *IMap* implementiertes, Objekt. So kann *ProbMap* die Implementierung von *IMap* aus *ROSMap* nutzen und zusätzlich weitere Funktionen wie *getProb*() zur Verfügung stellen. [eigene Abbildung]

Erzeugungsmuster Fabrikmethode

Zur Instanziierung des Lokalisierungssystems wird eine Fabrikmethode verwendet. Der Benutzer kennt die Fabrikmethode, jedoch nicht ihre Implementierung. Durch den Aufruf der Fabrikmethode erhält der Benutzer ein *FilterController*-Objekt, das ihm jedoch nur über eine Schnittstelle namens *IFilterController* bekannt gemacht wird. Intern legt die Bibliothek beim Aufruf der Fabrikmethode das *FilterController*-Objekt an und instantiiert dadurch das Lokalisierungssystem für den Benutzer.

Der Benutzer bedient den Filter nur durch die Schnittstelle *IFilterController*. So kann die Implementierung inklusive aller Header-Dateien geschützt werden. Der Benutzer benötigt zum Anlegen und Steuern des Filters nur Zugriff auf die Deklaration der Fabrikmethode und die Schnittstelle *IFilterController*.

5.3 Werkzeuge zur Implementierung

Aus dem erarbeiteten Entwurf kann nun die Implementierung der Software erfolgen. Auf die Implementierung selbst[12] wird an dieser Stelle nicht eingegangen. Stattdessen werden in diesem Abschnitt die verwendeten Werkzeuge sowie die Vorgehensweise beim Implementieren des Lokalisierungssystems beschrieben.

Während der gesamten Implementierung wird der Ansatz verfolgt, möglichst früh lauffähigen Quellcode zu erzeugen, um mit allen Systembausteinen frühstmöglich in die Testphase zu gelangen (“Prototyping“-Strategie). Dies erleichtert die Fehlersuche und Implementierung dahingehend, dass kleinere und somit einfacher zu überblickende Einheiten bearbeitet und untersucht werden können.

Abbildung 19 verdeutlicht den Implementierungsprozess neuer Systembausteine. Das Gesamtsystem, also das zu entwickelnde Lokalisierungssystem, besteht aus mehreren Systembausteinen, welche wiederum aus einzelnen Teilfunktionen bestehen. Jeder Systembaustein wird so früh wie möglich im Gesamtsystem getestet. Auch bei der Entwicklung von Teilfunktionen werden Testmechanismen der Implementierung vorgelagert, um frühstmöglich Fehler und Probleme bei der Implementierung aufzudecken und zu beheben.

[12] Die gesamte Implementierung ist auf der dieser Arbeit beigelegten CD abgelegt.

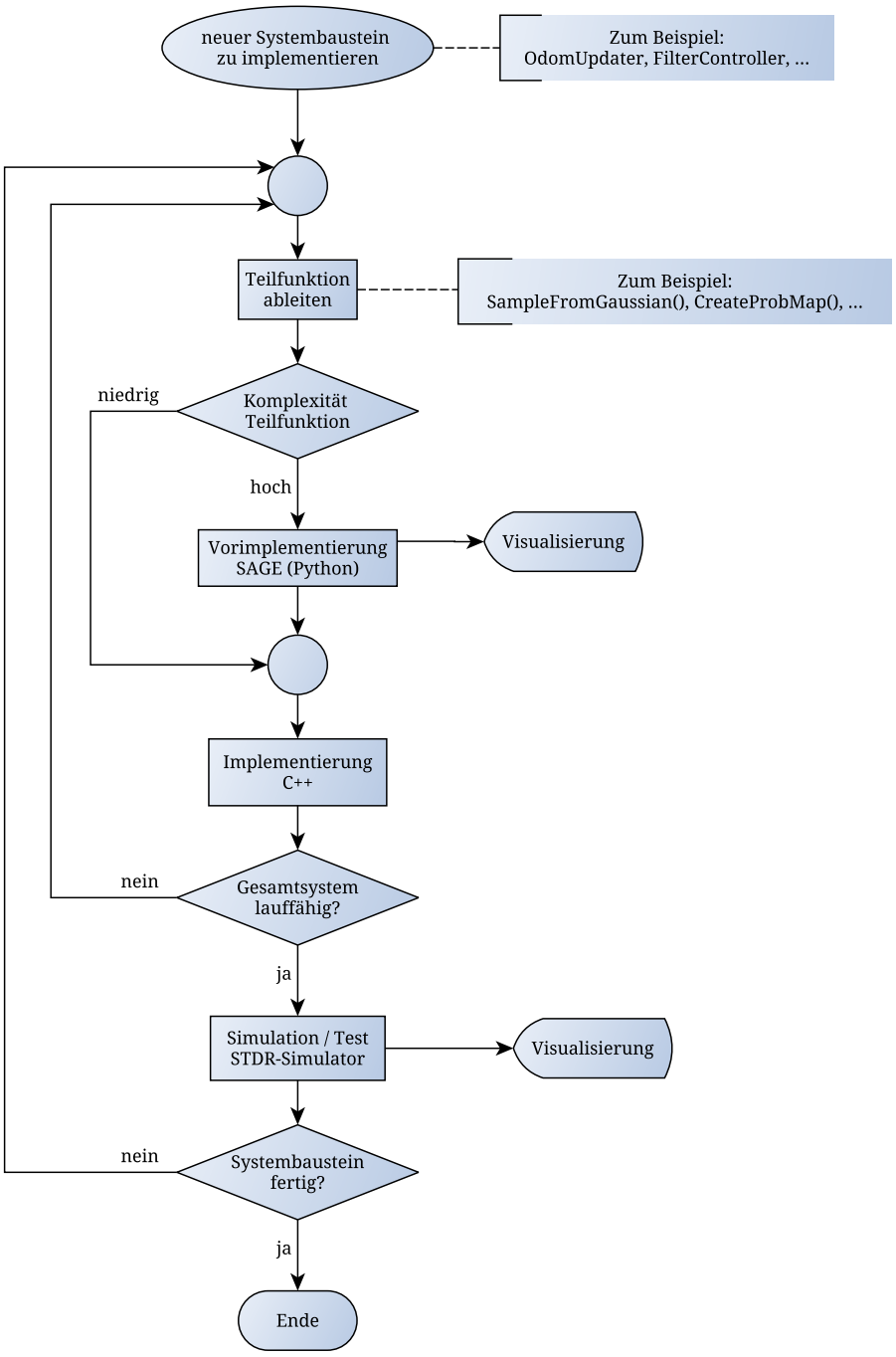


Abbildung 19: Implementierungsprozess neuer Systembausteine: Jeder Systembaustein wird in Teilfunktionen zerlegt. Bei hoher Komplexität werden diese vorimplementiert und visualisiert. Sobald der Systembaustein im Gesamtsystem lauffähig ist, kann seine Funktion mithilfe einer Simulation getestet und visualisiert werden. Dieser Prozess wird wiederholt, bis alle Teilfunktionen eines Systembausteins und schließlich alle Systembausteine des Gesamtsystems erfolgreich implementiert sind. [eigene Abbildung]

Vorimplementierung mit SageMath

Wie in Abbildung 19 erkennbar, wird der Implementierung komplexer Teilfunktionen eine Vorimplementierung vorangestellt. Dabei wird die Programmiersprache Python[13] zusammen mit dem Softwarepaket SageMath[14] verwendet. SageMath bietet eine Zusammenstellung von verschiedenen Softwarepaketen mit vorrangig mathematischem Kontext. Der Benutzer bedient SageMath über eine Konsole oder über das sogenannte "Notebook". In letzterem kann der Benutzer Seiten zu bestimmten Themen anlegen. Auf den Seiten selbst können Berechnungen, Visualisierungen und Beschreibungen beliebiger Art platziert werden. Abbildung 20 zeigt die Vorimplementierung dreier Funktionen in Form einer solchen SageMath-Notebook-Seite.

Die Verwendung von SageMath ermöglicht eine effiziente, isolierte Betrachtung einzelner Teilfunktionen. Grafische Ausgaben können in kurzer Zeit programmiert werden. Da Python eine interpretierte Sprache ist, ist kein Kompiliervorgang notwendig, wodurch der Arbeitsablauf beim Entwickeln einzelner Funktionen einfacher und ergonomischer ist. Programmierfehler können durch die Vorimplementierung schnell entdeckt und gelöst werden.

Gegenüber dieser Vorteile steht der Nachteil der "doppelten" Implementierung der Funktionen. Da die Portierung von Python zu C++ jedoch gut zu bewältigen ist, wird dieser Mehraufwand in Kauf genommen, um von den Vorteilen zu profitieren.

[13] `https://www.python.org`
[14] `www.sagemath.org`

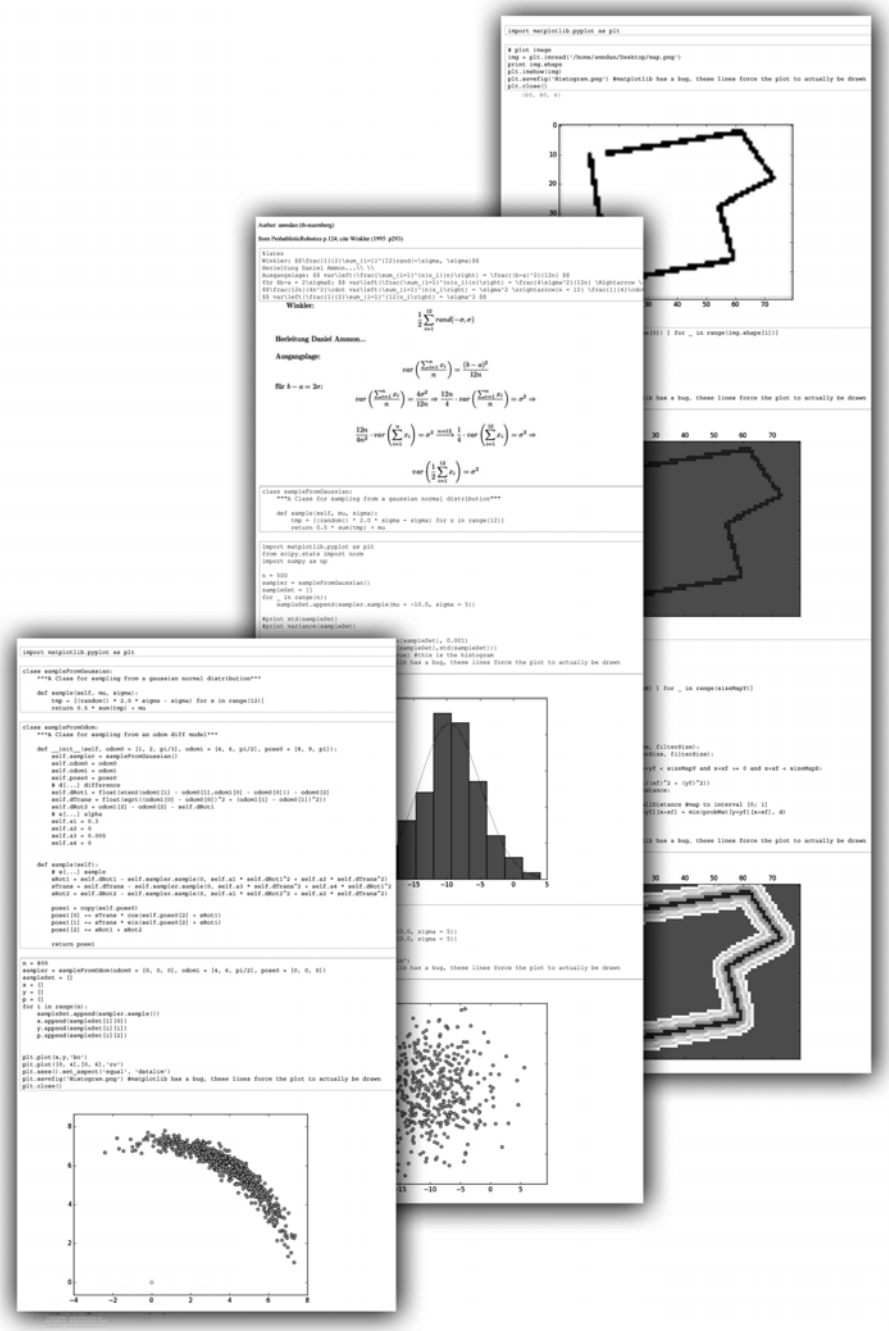


Abbildung 20: Vorimplementierung mit SageMath: Diese Abbildung zeigt drei Seiten aus dem SageMath Notebook, das zur Vorimplementierung einzelner Funktionen genutzt wird. Nicht der abgebildete Quellcode an sich wird damit verdeutlicht, sondern die Möglichkeit Quellcode, grafische Visualisierungen und mathematische Beschreibungen auf einer Notebook-Seite zu kombinieren. Beschreibung folgt von links unten nach rechts oben: Vorimplementierung der Funktion zum Generieren von Partikeln aus einem odometrischen Modell; Vorimplementierung der Funktion zum performanten Generieren einer normalverteilten Partikelwolke (vgl. [29]) inklusive Herleitung der Methode; Vorimplementierung der Funktion zum Erstellen einer Wahrscheinlichkeitskarte aus einer Occupancy-Grid-Map. [eigene Abbildung]

Simulation mittels STDR-Simulator

Sobald genügend Teilfunktionen eines Systembausteins implementiert sind, kann dieser in das Gesamtsystem integriert und mithilfe einer Simulation getestet werden (siehe Abbildung 19). Hierfür wird das ROS-Paket STDR-Simulator[15] verwendet. Dieser Simulator bietet die Möglichkeit, zweidimensionale Sensordaten eines Roboters zu simulieren, um dadurch die Umgebung und das Robotersystem selbst nachzustellen. Das Programm rviz[16], welches fester Bestandteil von ROS ist, wird verwendet, um die Simulation zu visualisieren. Abbildung 21 zeigt einen Testlauf mittels des STDR-Simulators, visualisiert mit der Software rviz.

Abbildung 21: Simulation mittels STDR-Simulator: Diese Abbildung visualisiert die Simulation mittels der STDR-Simulator-Software und die Visualisierung dessen mit der Software rviz. Im linken Bereich können anzuzeigende Datenströme aktiviert oder deaktiviert werden. Zusätzlich ist ein Eingabefenster für die Tastatureingabe zum Steuern des simulierten Roboters zu sehen. Im mittleren Bereich sind das Koordinatensystem eines nicht lokalisierten Roboters mit 2D-Laserscannerdaten und eine über die gesamte Karte gleichverteilte Partikelwolke zu sehen (globale Lokalisierungsphase). Im rechten Bereich ist der Verlauf der Lokalisierungsphase dargestellt (von oben nach unten) – durch Bewegung des Roboters und Aufnahme von Laserdaten verdichtet sich die Partikelwolke an der wahren Pose des Roboters. [eigene Abbildung]

Durch die Simulation können neue Funktionen und Systembausteine genau so getestet werden, wie direkt am Robotersystem. Dies spart Zeit und Ressourcen, da für die Implementierung die Hardware nicht verfügbar und betriebsbereit sein muss.

[15] `wiki.ros.org/stdr_simulator`
[16] `wiki.ros.org/rviz`

Versionierung mittels Git

Ein wichtiger Bestandteil der Implementierungsphase ist der Umgang mit den entstehenden Daten. Quellcode muss gesichert und versioniert werden, um eventuell auf ältere Stände in der Historie zurückgreifen zu können. Ein weit verbreitetes Werkzeug hierfür ist Git[17]. Es ermöglicht das Arbeiten mehrerer Personen an einem Projekt und die Versionierung und Sicherung der Projektdaten. Jeder Softwarestand hat eine eindeutige ID. Die vorliegende Arbeit basiert auf folgendem Softwarestand:

URL: `https://github.com/amndan/ohm_pf.git`
Commit-ID: `e1df-1360-4969-eb67-38c9-f800-8400-8f7a-4b5d-0435`

[17] `https://git-scm.com`

5.4 Dokumentation der Software

Das Werkzeug Doxygen[18] bietet die Möglichkeit aus kommentiertem Quellcode eine Software-Dokumentation zu erstellen. Das entwickelte Lokalisierungssystem ist vollständig kommentiert und die Doxygen-Dokumentation liegt dem Quellcode bei. Abbildung 22 zeigt exemplarisch die Startseite der Doxygen-Dokumentation.

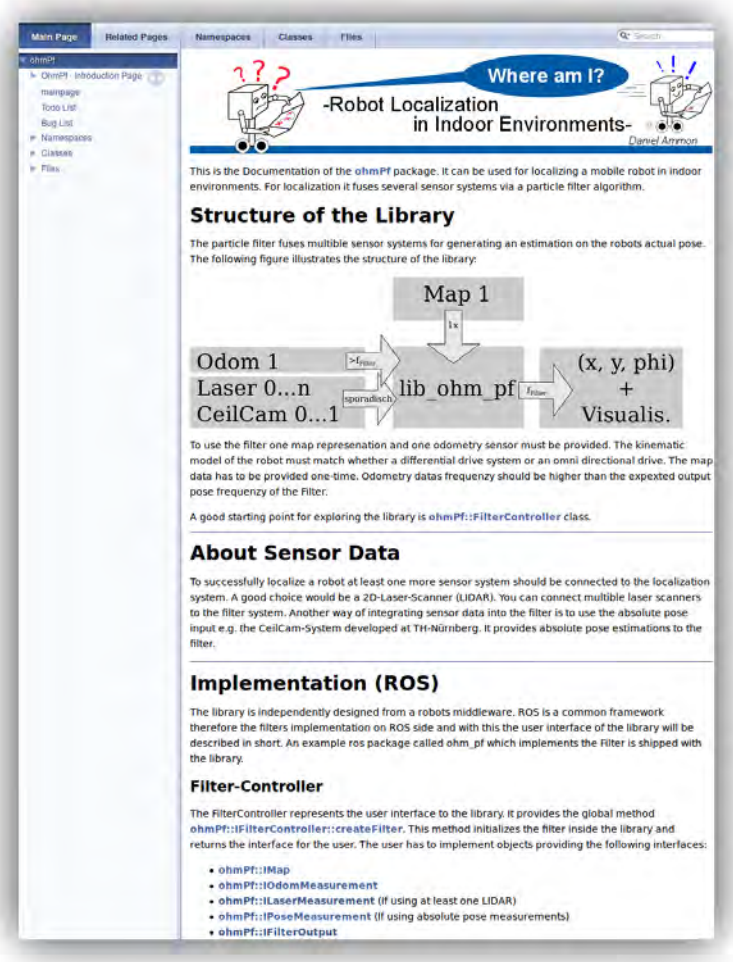


Abbildung 22: Doxygen-Dokumentation: Ein Ausschnitt der Startseite der Doxygen-Dokumentation zum entwickelten Lokalisierungssystem. [eigene Abbildung]

Das Lokalisierungssystem wird dem Benutzer über ROS zugänglich gemacht. Die Beschreibung aller Parameter des ROS-Knotens befindet sich in der Doxygen-Dokumentation und kann zusätzlich im Anhang dieser Arbeit nachgeschlagen werden (siehe Tabellen 3 und 4).

[18] `http://www.stack.nl/~dimitri/doxygen`

6 Test und Auswertung

Nach der vollständigen Implementierung des Lokalisierungssystems können Testdaten aufgenommen werden. Dieser Abschnitt beschreibt die durchgeführten Tests und deren Auswertung. Alle Testdaten wurden auf einem PC mit Intel Core i5-3570K Prozessor und einem virtuellen Ubuntu 14.04 (64-Bit) Betriebssystem generiert. Die Virtualisierung erfolgt durch die Software VMware-Workstation-Player[19] (Version 12). Die gesamten Testdaten sind auf der dieser Arbeit beigelegten CD abgelegt. Alle Sensordaten liegen im rosbag-Format[20] vor. So können alle Tests reproduziert werden.

6.1 Anwendungstest Bosch

Da das Lokalisierungssystem zur Unterstützung der Entwicklung des Bosch eigenen FTS dient, wird ein Anwendungstest in den Fertigungshallen der Firma Bosch durchgeführt (siehe Kapitel 2.1). Abbildung 23 zeigt die dort entstandene Umgebungskarte.

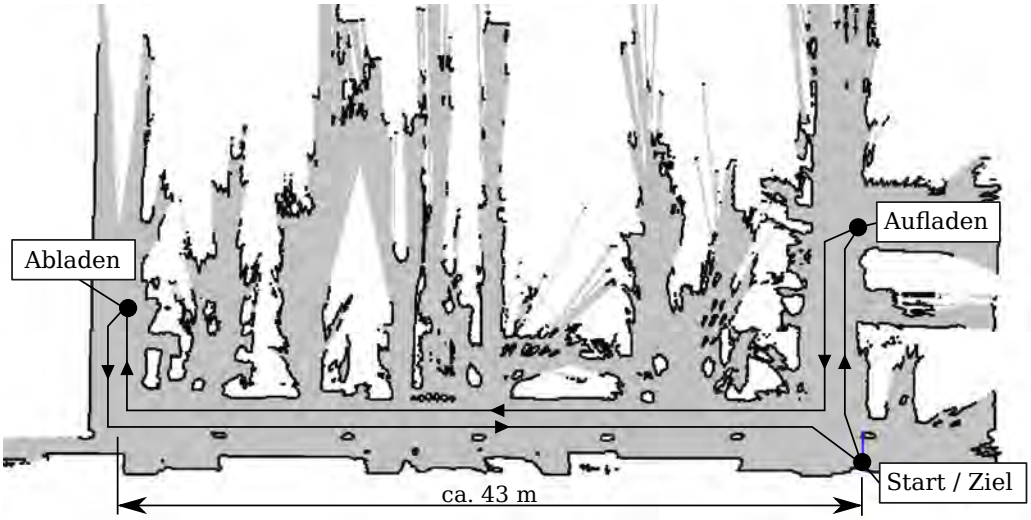


Abbildung 23: Umgebungskarte Bosch: Während des Anwendungstests bei Bosch mit einem 2D-Laserscanner und SLAM aufgenommene Umgebungskarte mit Markierungen der Start-, End-, Auflade- und Abladepose des Transportfahrzeuges. [eigene Abbildung]

Das Transportfahrzeug muss von einer Startpose aus eine Aufladepose anfahren, an der Abladepose abladen und wieder zur Startpose zurückkehren. Auflade- und Abladepose liegen zirka 43 m auseinander. Dieses Szenario gilt für beliebige Transportaufträge; das Transportgut beziehungsweise die Auflade- und Abladevorgänge werden nicht näher betrachtet.

[19] `http://www.vmware.com`
[20] `wiki.ros.org/rosbag`

In einem der Lokalisierung selbst vorangestellten Kartierungsschritt wird mit dem Software-Paket ohm-tsd-slam eine Umgebungskarte erstellt (siehe Abbildung 23). Während des SLAM-Vorgangs werden neben der Umgebungskarte auch Bilddaten der Fabrikdecke aufgenommen, um über das CeilCam-Softwarepaket[21] in der Lokalisierungsphase Posenschätzungen anhand der Deckenstruktur zu generieren. Nach Aufzeichnung der Umgebung (Fabrikdecke + SLAM) kann das Transportfahrzeug mittels einer Fernbedienung zu Start-, Auflade-, Ablade- und Endpose manövriert werden. Dabei werden folgende Datenströme aufgezeichnet:

- 2D-Laserscannerdaten (OMRON OS32C Sicherheitslaserscanner)
- Odometriedaten (IMU BNO055 + Radencoder; Fusioniert mittels EKF)
- Posenschätzung (Verwendung CeilCam-Softwarepaket)

Abbildung 24 zeigt den geschätzten Pfad des FTS resultierend aus den Odometriedaten und unter Verwendung des Lokalisierungssystems. Eingabegrößen in das Lokalisierungssystem sind hier während der Lokalisierungsphase aufgezeichnete Odometrie- und Laserscannerdaten, die Umgebungskarte und die Initialpose.

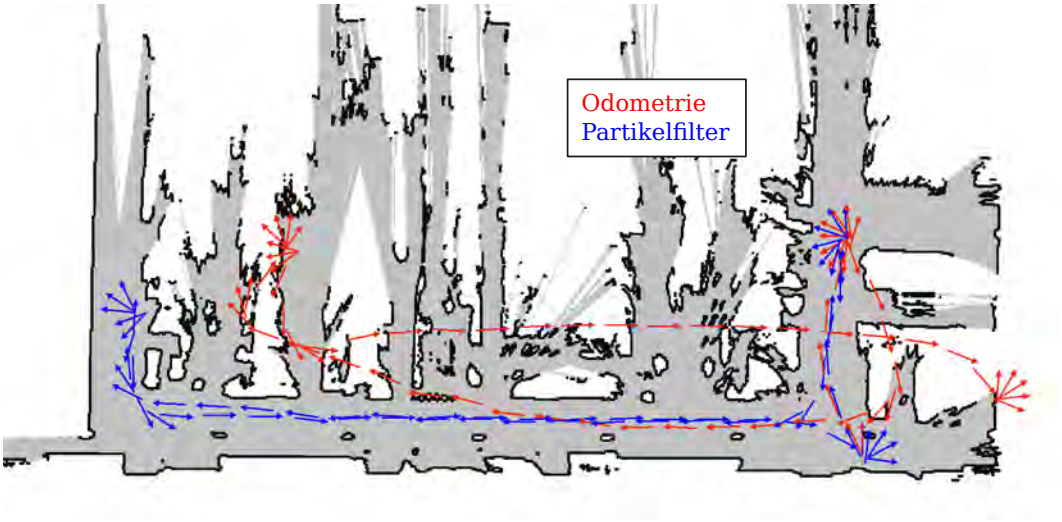


Abbildung 24: Lokalisierung Bosch: Die roten Pfeile zeigen den aus Odometriedaten entstandenen Pfad des Transportfahrzeugs. Start- und Endpose liegen nicht aufeinander. Die blauen Pfeile beschreiben den geschätzten Pfad aus dem Lokalisierungssystem (Partikelfilter). Start- und Endpose liegen hier nahe beieinander. [eigene Abbildung]

[21] Das CeilCam-Softwarepaket wurde parallel zu dieser Arbeit von Herrn Matthias Hertl im Rahmen seiner Masterarbeit mit dem Titel "Lokalisierung eines mobilen Roboters auf Basis von Deckenstrukturen mittels einer monokularen Kamera" implementiert. Es errechnet Posenschätzungen aus den Deckenstrukturen der Einsatzumgebung. Die Deckenstrukturen werden mit einer nach oben gerichteten und am Roboter befestigten 2D-Kamera aufgenommen. Die Posenschätzungen können in dem in dieser Arbeit entwickelten Lokalisierungssystem weiterverarbeitet werden.

Der Vergleich der Abbildungen 23 und 24 macht deutlich, dass die Odometriedaten hier keine zufriedenstellende Posenschätzung liefern. Sie bilden das Szenario nicht korrekt ab. Die Schätzung des Partikelfilters hingegen bildet den real gefahrenen Pfad richtig ab. Anfangs- und Endpose liegen, wie zu erwarten, auf einem Punkt und der Roboter bewegt sich nicht über Wände hinweg, sondern nur in freien Zonen der Umgebungskarte.

Die Initialpose wird dem Partikelfilter im ersten Teil vorgegeben. Wird der Roboter an einer unbekannten Pose in der Fabrik ausgesetzt und eingeschaltet, muss er sich global lokalisieren. Abbildung 25 zeigt eine globale Lokalisierungsphase anhand der aufgezeichneten Daten. In diese Fall wird dem Partikelfilter keine Initialpose vorgegeben, wodurch er anfangs die Partikel über die gesamte Umgebung gleich verteilt. Nach kurzer Zeit (zirka 5 s) kann mittels Laserscanner- und CeilCam-Messungen die wahre Pose des Roboters ermittelt werden.

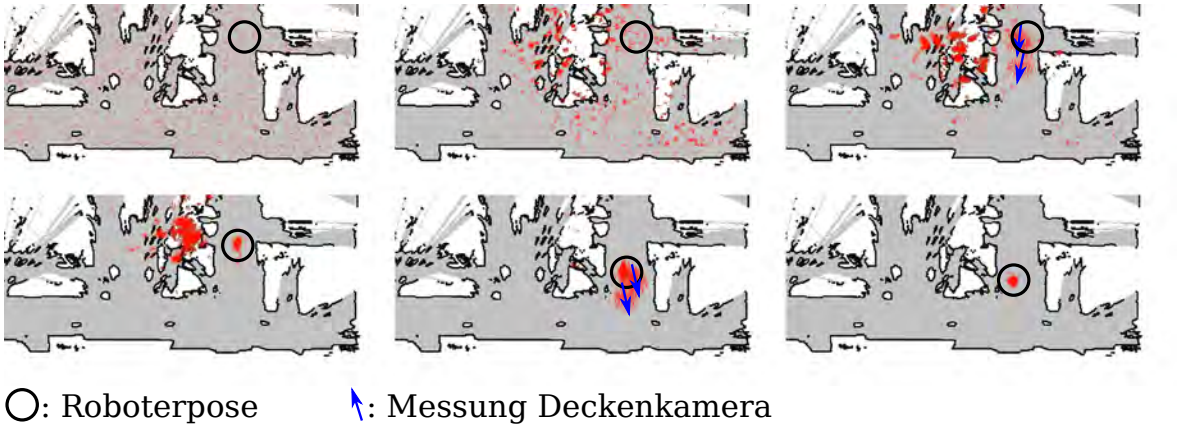


Abbildung 25: Globale Lokalisierung Bosch: Von links oben nach rechts unten; (1) Während sich das Transportfahrzeug bewegt, wird eine globale Lokalisierung eingeleitet – die Partikel werden über alle freien Bereiche in der Umgebungskarte gleich verteilt; (2) Die Partikel verdichten sich an Posen, zu denen die aktuellen 2D-Laserscannermessungen "passen"; (3) Die Deckenkamera gibt Posenschätzungen anhand der Deckenstrukturen aus – wenige Partikel werden an diesen Posen gestreut; (4) Die Wolke verdichtet sich an der wahren Roboterpose; (5) Eine weitere Messung der Deckenkamera bestätigt die Annahme und führt zur weiteren Verdichtung der Wolke; (6) Die Partikelwolke befindet sich vollständig an der wahren Pose des Roboters – die globale Lokalisierungsphase ist abgeschlossen. [eigene Abbildung]

Ist die Partikelwolke an einem Punkt verdichtet, werden andere Bereiche der Umgebung nicht mehr vom Lokalisierungssystem abgebildet. So ist es nicht möglich zu detektieren, dass die Posenschätzung falsch ist. Ein absolutes System, wie die Deckenkamera, kann in solchen Phasen der Fehllokalisierung die Relokalisierung des Filters ermöglichen. Abbildung 26 zeigt dieses Verhalten. Dem Partikelfilter wird hierbei eine falsche Roboterpose vorgegeben. Durch die absoluten Posenmessungen des CeilCam-Pakets ist das Lokalisierungssystem in der Lage, diese Fehllokalisierung zu detektieren und in kurzer Zeit ($\approx$5 s) die wahre Roboterpose zu erkennen.

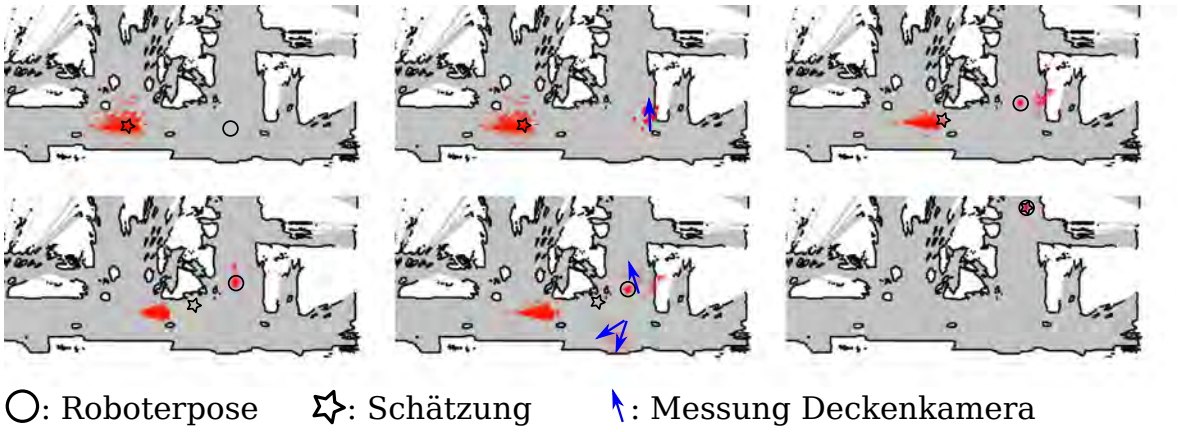


Abbildung 26: Relokalisierung Bosch: Von links oben nach rechts unten; (1) Die geschätzte Pose des Roboters ist falsch und die Streuung der Partikelwolke reicht nicht, um die wahre Pose zu erfassen – der Roboter ist falsch lokalisiert; (2) Die Deckenkamera liefert eine Annahme über die Roboterpose, woraufhin Partikel an dieser Pose gestreut werden; (3) Eine zweite Partikelwolke geht aus der Posenschätzung der Deckenkamera hervor; (4) Beide Partikelwolken existieren und die Roboterpose ist unbestimmt (Die alternative Verwendung eines Cluster-Algorithmus zum Extrahieren der Pose ergibt an dieser Stelle, dass die Posenschätzung auf einer der beiden Partikelwolken steht. Die Pose aber bleibt dennoch unbestimmt, da zwei Partikelwolken zur selben Zeit existieren.); (5) Eine weitere Messung der Deckenkamera bestätigt die Annahme der neuen Partikelwolke – der Resampling-Schritt führt zur Verlagerung der Partikel; (6) Die Partikelwolke befindet sich vollständig an der wahren Pose des Roboters – die Relokalisierungsphase ist abgeschlossen. [eigene Abbildung]

Anmerkungen zum Versuch "Anwendungstest Bosch":
Der folgende Umstand wird bei den Messungen zum Anwendungstest bei Bosch außer Acht gelassen und deshalb an dieser Stelle erwähnt: In beiden Fällen, der globalen sowie der Relokalisierung, gibt es Phasen, in denen die Pose unbekannt ist. Um die wahre Pose zu finden, muss sich das Fahrzeug bewegen, da sonst keine neuen Messwerte aus der Umgebung aufgenommen werden können. Die Bewegung bei unbekannter Pose ist gefährlich. Hier muss der Roboter in einen Explorationsmodus umschalten, bis die Posenschätzung wieder hinreichend aussagekräftig ist (siehe Kapitel 7.2). Ein möglicher Messwert zur Aussagekräftigkeit der Posenschätzung ist die Varianz der Partikelposen. Ist sie zu hoch, ist die Partikelwolke nicht verdichtet und die Posenschätzung ungenau oder falsch. Der Roboter darf sich nicht auf die Posenschätzung verlassen, sondern muss diese durch eine Explorationsphase verbessern, bevor er wieder Aufträge entgegennimmt. Dieses Verhalten ist Aufgabe einer Roboterkontrollarchitektur, beziehungsweise einer übergeordneten Softwareinstanz, wie beispielsweise eines Zustandsautomaten, der das Verhalten des Roboters steuert.

6.2 Simulation Gazebo/STDR

Das in Kapitel 6.1 gezeigte Verhalten des Lokalisierungssystems ist sehr subjektiv bewertet. Dies liegt hauptsächlich daran, dass die "wahren" Posendaten nicht vorhanden sind. Die Fabrikhalle müsste vermessen werden oder es müsste ein weiteres Lokalisierungssystem mit hoher Genauigkeit installiert werden, um die Genauigkeit des entwickelten Lokalisierungssystems beurteilen zu können. Eine praktikablere Möglichkeit zur Beurteilung der Genauigkeit stellt eine Simulation der Daten dar. Durch das Simulieren der Messdaten liegt die wahre Pose (*ground truth*) vor. Die geschätzte Pose kann damit verglichen und bewertet werden. Zur Bewertung wird eine parallel zu dieser Arbeit entwickelte Software[22] verwendet. Diese wurde ursprünglich den Vergleich verschiedener SLAM-Algorithmen (*slam benchmarking*) erstellt. Da hierbei letztendlich auch die Pose eines Roboters bewertet wird, kann das "Benchmarking-Tool" auch für diese Arbeit eingesetzt werden. Die Messungen der Genauigkeit des Lokalisierungssystems werden in drei Teile unterteilt:

- Absolute Genauigkeit
- Relative Genauigkeit
- Positioniergenauigkeit

Nachfolgend werden diese Kategorien und die darunter fallenden Messungen erläutert.

[22] Das in dieser Arbeit verwendete "Benchmarking-Tool" wurde parallel zu dieser Arbeit von Herrn Tobias Fink im Rahmen seines Masterstudiums im Robotik-Labor der TH-Nürnberg implementiert und basiert auf der Methodik eines online verfügbaren SLAM-Benchmarking-Tools [18, 3] (`http://kaspar.informatik.uni-freiburg.de/~slamEvaluation/index.php`)

Absolute Genauigkeit

Um die absolute Genauigkeit des Lokalisierungssystems zu messen wird jede vom Lokalisierungssystem ausgegebene Pose wird mit der wahren Pose aus der Simulation verglichen. So kann zu jeder geschätzten Pose ein translatorischer Fehler e_t und ein rotatorischer Fehler e_r ermittelt werden. Abbildung 27 zeigt die Struktur der Datenerhebung und Datenverarbeitung anhand eines Flussdiagramms.

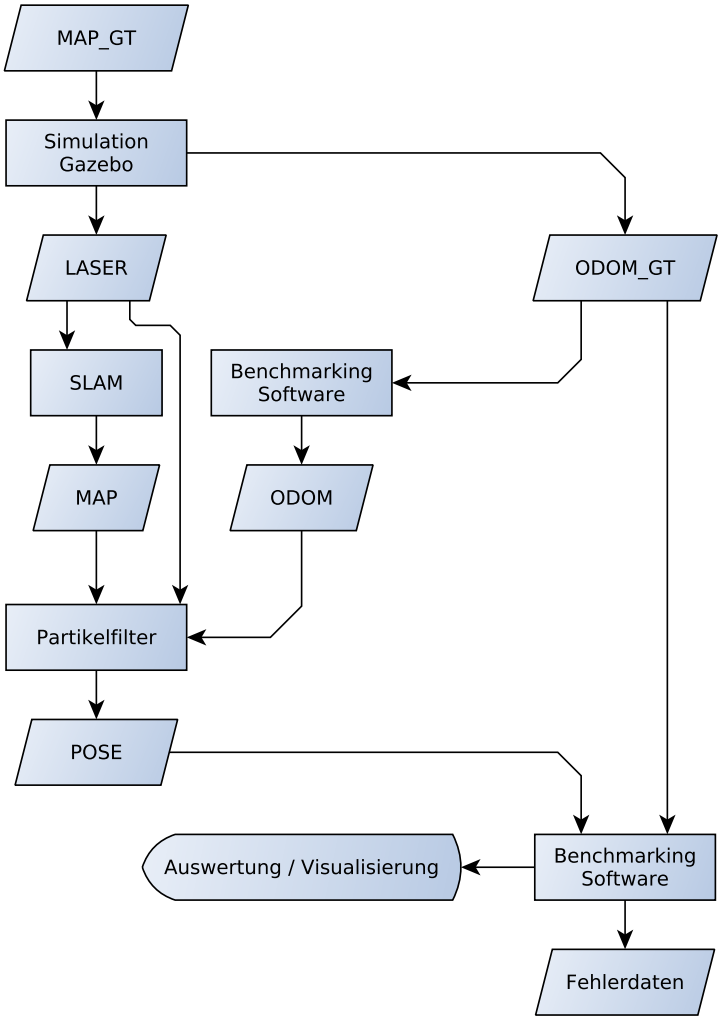


Abbildung 27: Datenerhebung mittles Simulation: Datenerhebung und Datenverarbeitung zum Testen des Lokalisierungssystems mittels Simulationsdaten. Der Partikelfilter wird mit einer Umgebungskarte aus einem vorangegangenen SLAM-Schritt, Laserdaten aus der Simulation und durch die Benchmarking-Software verfälschten Odometriedaten gespeist. Die vom Lokalisierungssystem errechnete Posenschätzung wird mithilfe der Benchmarking-Software mit den wahren Odometriedaten aus der Simulation verglichen. [eigene Abbildung]

Die Simulationssoftware Gazebo[23] liefert Odometrie- und Laserdaten. Das hierbei verwendete Robotermodell entspricht einem KUKA youBot[24] mit omnidirektionalem Antrieb. Des Weiteren ist die Simulation eines URG-04 2D-Laserscanners[25] an der Stirnseite des Roboters angebracht. Die einzelnen Entfernungsmessungen der Laserdaten werden von der Simulationssoftware durch einen GUI-Raytracing-Algorithmus auf Basis der Umgebungskarte (MAP_GT) generiert und mit einem normalverteilten Rauschen ($\mathcal{N}(0;0{,}01)$) versehen. Abbildung 28 zeigt das verwendete Robotermodell in der Umgebungskarte mitsamt der Partikelwolke, den Laserdaten und der Posenschätzung.

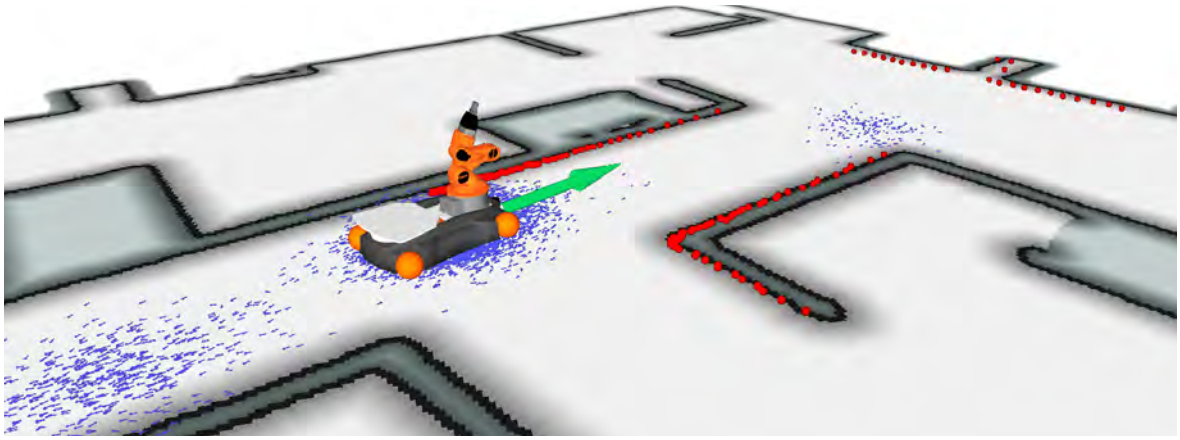


Abbildung 28: Simulation mittels Gazebo: Die Abbildung zeigt einen Ausschnitt aus einer Umgebungskarte. Darin ist die Simulation eines KUKA youBot zu sehen. Desweiteren sind die simulierten Entfernungsmessungen aus einem URG-04 2D-Laserscanners (rot), die Partikelwolke (blau) und die Posenschätzung (grün) zu erkennen. Die Szene wurde mit der Software rviz visualisiert. [eigene Abbildung]

Der Roboter wird mit einer Fernsteuerung durch die Simulationsumgebung gesteuert. Dabei erfährt der Roboter Maximalgeschwindigkeiten von 0,5 m/s beziehungsweise 30 °/s.

[23] `http://www.gazebosim.org`
[24] `http://www.youbot-store.com`
[25] `http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html`

Die Odometriedaten aus der Simulation sind nicht verfälscht und bilden die wahre Pose des Roboters (ODOM_GT) ab. In der Realität bilden die Odometriedaten die Pose des Roboters unzureichend genau ab (vgl. Abbildung 24). Mittels der Benchmarking-Software werden die Odometriedaten mit einem Messrauschen und einem Drift versehen (ODOM). In Abbildung 29 ist die Umgebungskarte mit den wahren (blau) und den verfälschten (rot) Odometriedaten zu sehen.

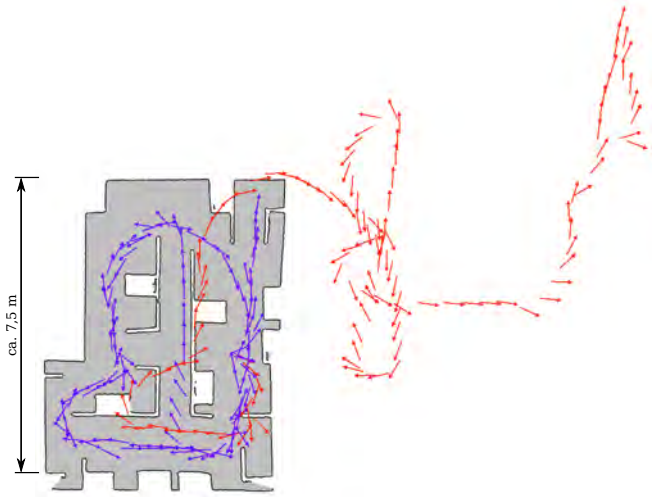
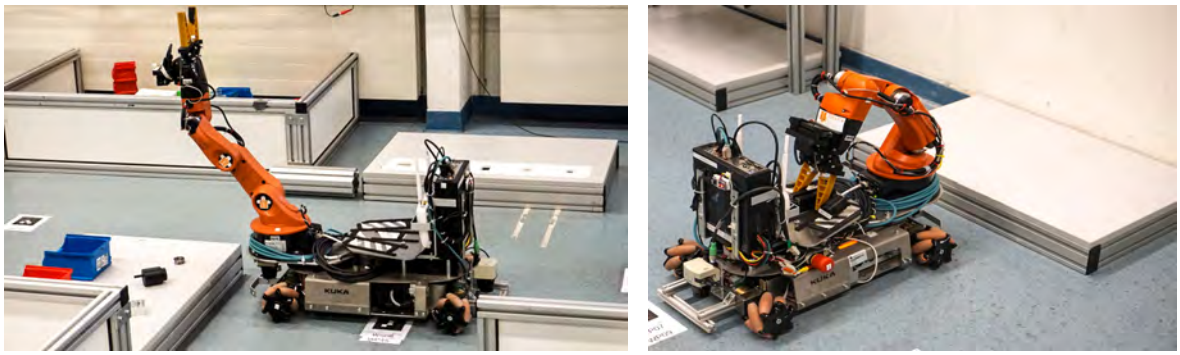


Abbildung 29: Verfälschte Odometriedaten: Die blauen Pfeile zeigen den abgefahrenen Pfad (ODOM_GT). Die roten Pfeile zeigen die mithilfe des Benchmarking-Tools verfälschten Odometriedaten (ODOM). Die Daten sind so stark verändert, dass sie nach kurzer Zeit die Grenzen der Umgebungskarte verlassen. [eigene Abbildung]

Die Umgebungskarte wird der aus dem RoboCup@Work Wettbewerb 2016[26] in Leipzig nachempfunden. Das Lokalisierungssystem wurde dort zusammen mit der youBot-Roboterplattform verwendet (siehe Abbildung 30). Abbildung 31 (1) zeigt die am Wettbewerb aufgenommene Umgebungskarte. Für die Simulation wird ein etwas vereinfachtes Kartenmodell angefertigt, welches die reale Umgebung repräsentiert (MAP_GT – Abbildung 31 (2)). So eine Karte ist in der Realität nicht vorhanden. Sie muss in einem Kartierungsschritt aufgezeichnet werden. Auch dies geschieht über die Simulation und das laboreigene SLAM-Paket ohm-tsd-slam. Abbildung 31 (3) zeigt die aus dem SLAM entstandene Umgebungskarte (MAP).

Zur Beurteilung der absoluten Genauigkeit wird der Partikelfilter mit der aus dem Kartierungsschritt aufgenommenen Karte, den Laserdaten aus der Simulation und den verfälschten Odometriedaten gespeist (siehe Abbildung 27).

[26] `http://www.robocup2016.org/de/ligen/robocup-industrial/robocup-work`



(a) (b)

Abbildung 30: Roboterplattform youBot: Beide Abbildungen zeigen die Roboterplattform youBot während eines Wettbewerbs. (a) Identifikation der Objekte auf einer Ablagefläche vor dem Roboter mittels einer am Roboterarm angebrachten 3D-Kamera. (b) Ablegen eines gegriffenen Objekts auf der robotereigenen Ablagefläche. [eigene Abbildung]

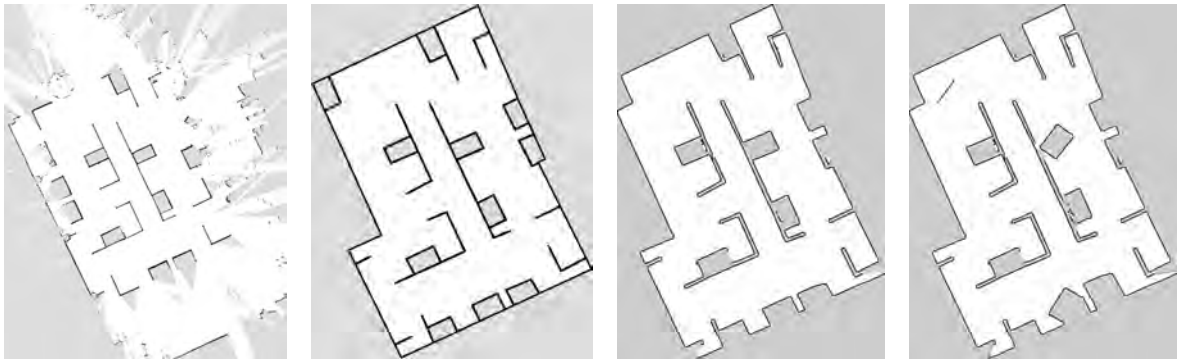


Abbildung 31: Umgebungskarten Simulation: Von links nach rechts: (1) Aufgezeichnete Umgebungskarte RoboCup@Work Wettbewerb 2016 in Leipzig; (2) Abbildung der dem Simulator zugeführten Umgebungskarte (MAP_GT); (3) Aus den simulierten Laserdaten und SLAM entstandene Umgebungskarte (MAP); (4) Für spätere Tests von Hand modifizierte Umgebungskarte (MAP_MOD; vgl. Kapitel 6.2). [eigene Abbildung]

Das Benchmarkin-Tool errechnet auf Basis der Posendaten aus dem Partikelfilter (POSE) und den wahren Posendaten aus der Simulation (ODOM_GT) die Abweichung der geschätzten Posendaten von den wahren Posendaten (vgl. Abbildung 27). Die dafür verwendete Fehlermetrik stammt aus [18, 3] und ist nachfolgend in eigenen Worten beschrieben.

Anfangs werden die Posendaten aus dem Partikelfilter $\vec{\tilde{x}}^n$, sowie die wahren Posendaten aus der Simulation $\vec{x}^n$ zu homogenen Transformationsmatrizen (bezeichnet mit $\mathbf{T}$) umgeformt:

$$\vec{x}^n = \begin{pmatrix} x \\ y \\ \phi \end{pmatrix} \Rightarrow \mathbf{T}_n = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & x \\ \sin(\phi) & \cos(\phi) & y \\ 0 & 0 & 1 \end{pmatrix}$$

Damit kann zu jeder Posenschätzung $\vec{\tilde{x}}^n$ aus dem Partikelfilter die relative Transformation $\widetilde{\mathbf{T}}_{0n}$ bezogen auf die Startpose $\vec{\tilde{x}}^0$ errechnet werden:

$$\widetilde{\mathbf{T}}_{0n} = \widetilde{\mathbf{T}}_0^{-1} \cdot \widetilde{\mathbf{T}}_n$$

Analog hierzu kann auch zu jeder wahren Pose $\vec{x}^n$ aus der Simulation eine Transformation relativ zu der Startpose $\vec{x}^0$ errechnet werden:

$$\mathbf{T}_{0n} = \mathbf{T}_0^{-1} \cdot \mathbf{T}_n$$

Die beiden Transformationen $\widetilde{\mathbf{T}}_{0n}$ und $\mathbf{T}_{0n}$ können nun unabhängig von den Startposen miteinander verglichen werden. Hierfür werden die Transformationsmatrizen wieder in x-, y- und ϕ-Wert zerlegt:

$$\widetilde{\mathbf{T}}_{0n} \Rightarrow \begin{pmatrix} \widetilde{x}_{0n} \\ \widetilde{y}_{0n} \\ \widetilde{\phi}_{0n} \end{pmatrix}; \mathbf{T}_{0n} \Rightarrow \begin{pmatrix} x_{0n} \\ y_{0n} \\ \phi_{0n} \end{pmatrix}$$

Der absolute translatorische, sowie der absolute rotatorische Fehler zu jeder Posenschätzung $\vec{\tilde{x}}^n$ kann dann wie folgt berechnet werden:

$$e_{t,n} = \sqrt{(\widetilde{x}_{0n} - x_{0n})^2 + (\widetilde{y}_{0n} - y_{0n})^2}$$
$$e_{r,n} = |\widetilde{\phi}_{0n} - \phi_{0n}|$$

Die Startpose der beiden Datensätze kann auf diese Weise unterschiedlich sein. Wichtig ist jedoch, dass die beiden Datensätze zeitlich synchronisiert sind. Jede Pose hat hierfür einen Zeitstempel. Die Synchronisierung resultiert aus der Vorgehensweise mit der das Lokalisierungssystem die Zeitstempel seiner Ausgabepose setzt. Er verwendet hierfür nicht die Uhrzeit, sondern den Zeitstempel des letzten verarbeiteten Odometriedatums. Da die Odometriedaten aus den wahren Posen (ODOM_GT) entstehen, liegt so zu jeder vom Lokalisierungssystem ausgegebenen Pose eine wahre Pose mit gleichem Zeitstempel vor.

Abbildung 32 zeigt die so berechneten translatorischen und rotatorischen Fehler aller Posenschätzungen in Form kumulativer Verteilungsfunktionen (*Cumulative Distribution Function – CDF*).

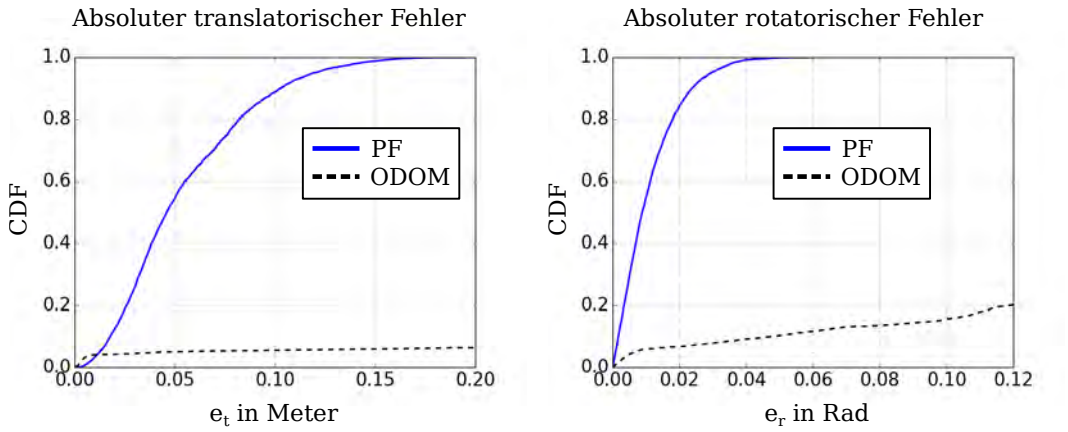


Abbildung 32: Absolute Genauigkeit: Der absolute translatorische und absolute rotatorische Fehler aller Posenmessungen des Partikelfilters zur Darstellung der absoluten Genauigkeit. Die Kurve der Posenschätzungen aus den Odometriedaten ist aus Darstellungsgründen nicht vollständig abgebildet. [eigene Abbildung]

Es zeigt sich, dass die Posenwerte des Partikelfilters um ein Vielfaches genauer sind als die der Odometriedaten. Durch die Fusion der Laserdaten, der Umgebungskarte und der Odometriedaten kann hier ein deutlich besseres Ergebnis erzielt werden. Zirka 90 % aller Messungen weisen einen translatorischen Fehler unter 10 cm und einen rotatorischen Fehler unter 1,5 ° auf. Insgesamt wurden zirka 3700 Posenschätzungen durch das Lokalisierungssystem errechnet.

Die absolute Posengenauigkeit stellt einen wichtigen Messwert dar. Sie gibt an, wie gut die reale Pose im Robotersystem abgebildet wird. Müssen Bezüge zu Posen in der Realität hergestellt werden, hilft diese Messung dabei, den zu erwartenden Fehler einschätzen zu können. Neben der absoluten Genauigkeit der Posenschätzung kann auch eine relative Posengenauigkeit ermittelt werden.

Relative Genauigkeit

Um Transportaufträge auszuführen, verwenden autonome Robotersysteme Navigationsalgorithmen. Diese steuern den Roboter anhand der Umgebungskarte und der Posenschätzung von Start- zu Zielpose. Dabei ist nicht die absolute Genauigkeit der Pose in der Realität entscheidend, sondern die relative Genauigkeit zwischen kleineren Wegabschnitten. Umgebungskarten bilden oft nur die räumliche Struktur, nicht aber die realen Verhältnisse ab. Dies gilt auch für die Posenschätzung in einer solchen Umgebungskarte. Die Autoren der Publikationen [18] und [3] verwenden zur Bewertung von SLAM-Algorithmen relative Posenänderungen (*relations*) zu verschiedenen Zeitpunkten. Diese Technik wird auch in dieser Arbeit verwendet, um die Konsistenz der Posenschätzung zu überprüfen.

Das Benchmarking-Tool kann auch für diese Fehlermetrik verwendet werden. Es werden nun nicht mehr zu allen Posenschätzungen aus dem Lokalisierungssystem absolute Transformationen gebildet und mit denen der wahren Posendaten verglichen (siehe Kapitel 6.2), sondern an verschiedenen Stellen äquidistante Transformationen aus den wahren Daten gebildet und mit den Messdaten verglichen. Die Transformationen beziehen sich nun nicht mehr alle auf die Startpose, sondern liegen aneinandergereiht. Abbildung 33 zeigt die so entstandenen Transformationen.

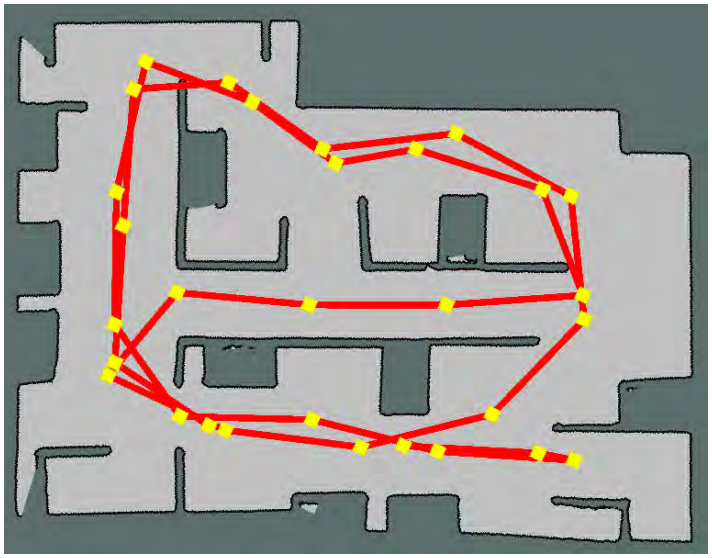


Abbildung 33: Relative Transformationen: Mit dem Benchmarking-Tool erstellte und visualisierte relative Transformationen (*relations*). Die gelben Quadrate stellen die ausgewählten Messpunkte aus den wahren Daten (ODOM_GT) dar. Die roten Linien stellen die Transformation zwischen jeweils zwei Messpunkten dar. Die Transformationen haben hier eine Länge von jeweils zwei Metern. [eigene Abbildung]

Zu jedem Messpunkt aus den Referenzdaten sucht das Benchmarking-Tool die passende Pose aus der Messung. Ist keine Pose zum gesuchten Zeitpunkt vorhanden, wird über die umliegenden Posen linearisiert. Die maximale Zeitdifferenz zwischen den zur Linearisierung verwendeten Posen darf 100 ms nicht überschreiten. Die Linearisierung ist hier nötig, da nun Messwerte aus den wahren Posen für den Vergleich ausgewählt werden. Diese liegen mit einer Datenrate von 100 Hz vor. Das Lokalisierungssystem liefert nur eine Datenrate von zirka 20 Hz. Somit könnte ohne Linearisierung nicht zu jeder wahren Pose eine passende, gemessene Pose gefunden werden. Die Fehlermetrik gleicht der aus der absoluten Genauigkeitsmessung.

Es werden insgesamt sechs Bewertungen mit jeweils verschiedenen Rasterungen der Transformationen errechnet. Als Datengrundlage dienen weiterhin die gemessenen und wahren Posendaten, wie sie bei der Messung der absoluten Genauigkeitsmessung in Kapitel 6.2 Verwendung finden. Die ausgewerteten Rasterungen sind in Meter: 2,0 (Abbildung 33); 1,0; 0,5; 0,1; 0,01; 0,001. Abbildung 34 zeigt das Messergebnis der verschiedenen, relativen Messungen analog zur absoluten Genauigkeitsmessung.

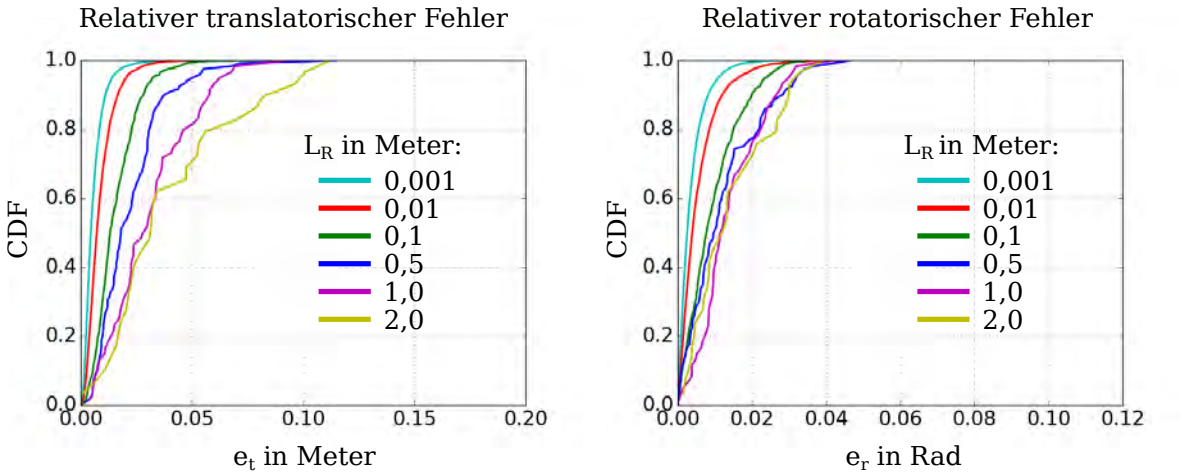


Abbildung 34: Relative Genauigkeit: Um die relative Genauigkeit herauszufinden, wird das Messergebnis unter verschiedenen Rasterungen mit den wahren Daten verglichen. Die beiden Grafen zeigen die so entstandenen kumulierten Fehlerverteilungsfunktionen des translatorischen und rotatorischen Fehlers. [eigene Abbildung]

Mit steigender Rasterlänge L_R steigt auch der translatorische beziehungsweise rotatorische Fehler. Dies ist ein zu erwartendes Ergebnis, da auch ein Driftfehler der Odometriedaten mit der zurückgelegten Wegstrecke steigt. Auch die Umgebungskarte unterliegt einem Drift-Fehler. Dieser führt bei großen Schleifen zu Schleifenfehlern im SLAM (*loop closing error*).

Eine weitaus interessantere Erkenntnis liefert die Betrachtung des relativen translatorischen Fehlers prozentual bezogen auf die Rasterlänge L_R (siehe Abbildung 35a).

Zwischen den Rasterlängen 0,5 m und 1 mm steigt der relative translatorische Fehler bezogen auf die Rasterlänge beträchtlich an. Bei einer Rasterlänge von 1 cm haben zirka 30 % der Messungen einen Fehler größer 100 % der Rasterlänge.

Durch Aufzeichnen der Mittelwerte der relativen translatorischen Fehler bezogen auf die Rasterlänge in einem doppelt logarithmischen Koordinatensystem (siehe Abbildung 35b), können folgende Beobachtungen gemacht werden:

1. Bis zu einer Rasterung von 0,5 m liegt der mittlere translatorische Messfehler unter 5 % des Rastermaßes.

2. Bis zu einer Rasterung von 10 cm liegt der mittlere translatorische Messfehler unter 20 % des Rastermaßes.

3. Ab einer Rasterung von zirka 1 cm liegt der mittlere translatorische Fehler über 100 % des Rastermaßes.

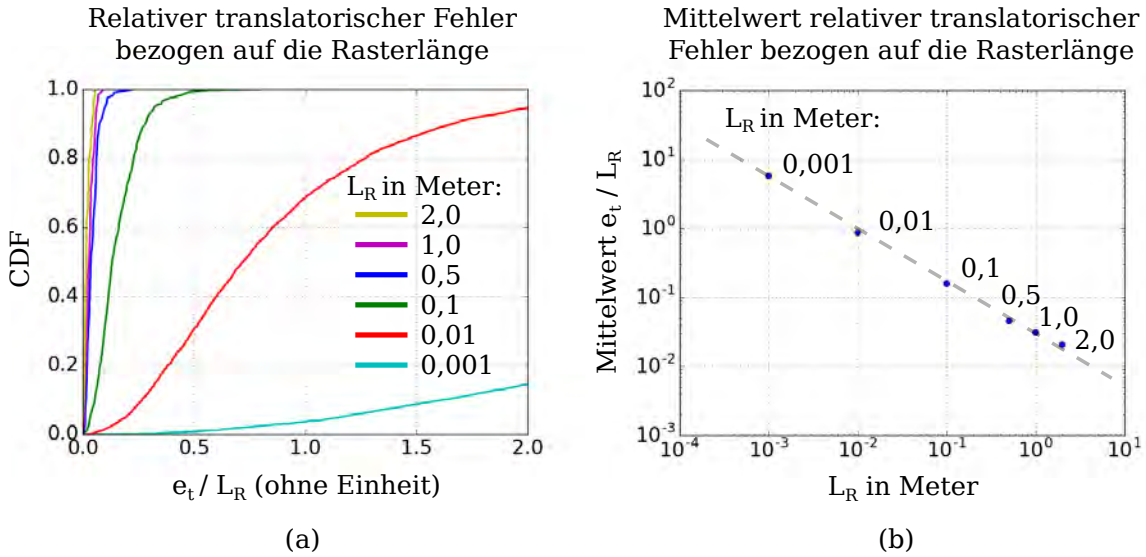


Abbildung 35: Prozentuale relative Genauigkeit: (a) Der relative translatorische Fehler bezogen auf die Rasterlänge; (b) Die Mittelwerte der relativen translatorischen Fehler bezogen auf die Rasterlänge aufgetragen in doppelt logarithmischer Skalierung (Ausgleichsgerade empirisch ermittelt). [eigene Abbildung]

Ein Fehler von 100 % des Rastermaßes bedeutet, dass der Fehler der geschätzten Pose des Lokalisierungssystems in der Größenordnung des Rastermaßes liegt. Das Lokalisierungssystem kann dieses Rastermaß somit nicht mehr auflösen. Schlussendlich ist über diese Messung zu sagen, dass die Auflösung und damit die Genauigkeit des Lokalisierungssystems die Struktur der Umgebung abzubilden im Bereich einiger Zentimeter liegt.

Positioniergenauigkeit

Für das Navigieren und Zurechtfinden in einer Umgebungskarte spielt die relative Genauigkeit eine große Rolle. Für das Ausführen von Transportaufträgen durch autonome Transportfahrzeuge gibt es eine weitere, wichtige, die Genauigkeit beschreibende Größe: Die Positioniergenauigkeit. Dieser Begriff beschreibt die Genauigkeit beim wiederholten Anfahren einer Pose (Reproduzierbarkeit). Die Start- und Zielposen der Transportaufträge können bei vorhandener Umgebungskarte eingelernt werden, um den absoluten Fehler zu kompensieren und auf das Ausmessen der Umgebung zu verzichten. Die letztendlichen Posenwerte spielen hier keine Rolle. Das Fahrzeug soll jedoch in der Lage sein, die eingelernten Posen mehrmals möglichst genau anzufahren, um dort Aufträge auszuführen. Auch für diesen Test wird ein Simulator verwendet. Es handelt sich um den in Kapitel 5.3 erwähnten STDR-Simulator, der auch auf Basis der Umgebungskarte (wieder MAP_GT) Sensordaten eines omnidirektionalen Roboters simuliert. Der Aufbau des Szenarios ist aus Abbildung 36 zu entnehmen.

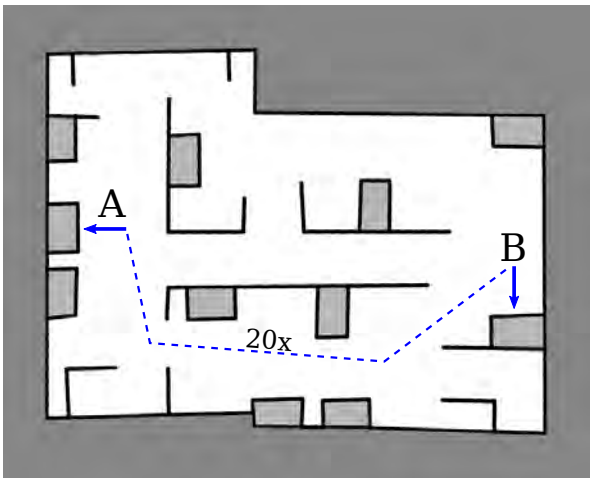


Abbildung 36: Szenario Reproduzierbarkeit: Szenario zur Positioniergenauigkeitsmessung. Der Roboter pendelt 20-mal zwischen den Posen A und B. Er benutzt dabei den gestrichelt eingezeichneten Pfad. Der Pfad ist hier nur angedeutet, der wahre Pfad variiert in jeder Iteration, da ein Navigationsalgorithmus zum Steuern des Roboters verwendet wird. [eigene Abbildung]

Der simulierte Roboter fährt mehrere Male zwischen zwei Stationen, wie sie in Abbildung 30 zu sehen sind, hin und her. Dies entspricht einem einfachen Transportauftrag im Sinne des RoboCup@Work Wettbewerbs und auch eines Auftrags im industriellen Einsatz. Gesteuert wird der Roboter hier von einem Navigationsalgorithmus, um die Bewegungsdynamik, die dabei entsteht, zu erhalten. Als Navigationssystem wird das Paket move-base[27] von ROS verwendet. Auf Basis der Posendaten des Roboters aus dem Simulator steuert dieses Paket den Roboter bis auf zwei Zentimeter genau an die Zielposen heran. Hierbei treten Maximalgeschwindigkeiten von zirka 0,5 m/s und 28 °/s auf. Nach kurzem Verharren an der Zielpose wird die jeweils andere Pose angefahren. Dabei werden die simulierten Laserdaten aus einem 2D-Laserscanner und die wahren Posendaten aufgezeichnet. Da es mit dem STDR-Simulator nicht möglich ist, Laserdaten mit Messrauschen zu erzeugen, werden die Daten nachträglich mit einem Messrauschen versehen. Auch aus den wahren Posendaten wird mittels des Benchmarking-Tools wieder ein mit starkem Drift und Messrauschen behaftetes Odometriesignal generiert (siehe Abbildung 37).

Anhand der modifizierten Laser- und Odometriedaten werden mit dem Partikelfilter Posenschätzungen generiert. Es werden zwei Messläufe unternommen. Bei dem ersten werden dem Lokalisierungssystem die Umgebungskarte (MAP), die verfälschten Odometriedaten und die verfälschten Laserdaten zugeführt. Bei dem zweiten werden dem Lokalisierungssystem die verfälschten Messdaten (Odometrie- und Laserdaten) mit zehnfacher Geschwindigkeit zugeführt. Dadurch entstehen anspruchsvollere Bedingungen für das Lokalisierungssystem. Die Maximalgeschwindigkeiten erhöhen sich auf zirka 5 m/s und 280 °/s. Die Zykluszeit des Algorithmus beträgt weiterhin 20 Hz, somit sinkt zusätzlich die Abtastrate der Messdaten.

[27] `wiki.ros.org/move_base`

Abbildung 37: Odometriesignal Positioniergenauigkeit: Die wahren Odometriedaten (rot) beschreiben den Pfad, wie im Szenario in Abbildung 36 beschrieben. Der Roboter pendelt zwischen den Posen A und B. Die verfälschten Odometriedaten (blau) sind nicht komplett abgebildet, da sie schon während der ersten Iteration die Grenzen der Umgebungskarte verlassen. [eigene Abbildung]

Zusätzlich wird im zweiten Messlauf statt der Umgebungskarte aus dem SLAM Schritt eine händisch modifizierte Karte verwendet (siehe MAP_MOD Abbildung 31). Dies Simuliert Veränderungen in der Umgebung, die das Lokalisieren anhand von Laserdaten erschweren, da die gemessenen Strukturen nicht mehr mit den in der Umgebungskarte aufgezeichneten übereinstimmen. Abbildung 38 verdeutlicht diesen Umstand.

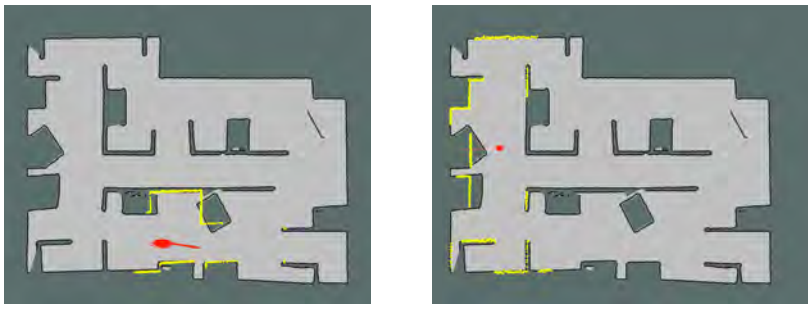


Abbildung 38: Einfluss modifizierte Umgebungskarte: Durch die Modifikationen an der Umgebungskarte passen die Laserdaten (gelb) trotz richtiger Posenschätzung (rot) nicht zu den Kartendaten. So wird die Lokalisierung in der Umgebung anhand der Laserdaten erschwert. [eigene Abbildung]

Abbildung 39 und 40 verdeutlichen die Ergebnisse aus den beiden Messläufen. Bis auf einen Ausreißer (um Messung Nr. 10000 Abbildung 39) kann um alle Posenschätzungen eine Fehlerschranke gemäß der festgelegten Genauigkeitsanforderungen aus der Anforderungsliste gelegt werden. Trotz zehnfacher Geschwindigkeit konnte das Lokalisierungssystem den Genauigkeitsanforderungen gerecht werden. Durch Veränderungen in der Umgebungskarte wird eine dynamische Umgebung simuliert. Das Einhalten der Anforderung trotz dieser Veränderungen weist auf eine hohe Robustheit des Lokalisierungssystems hin.

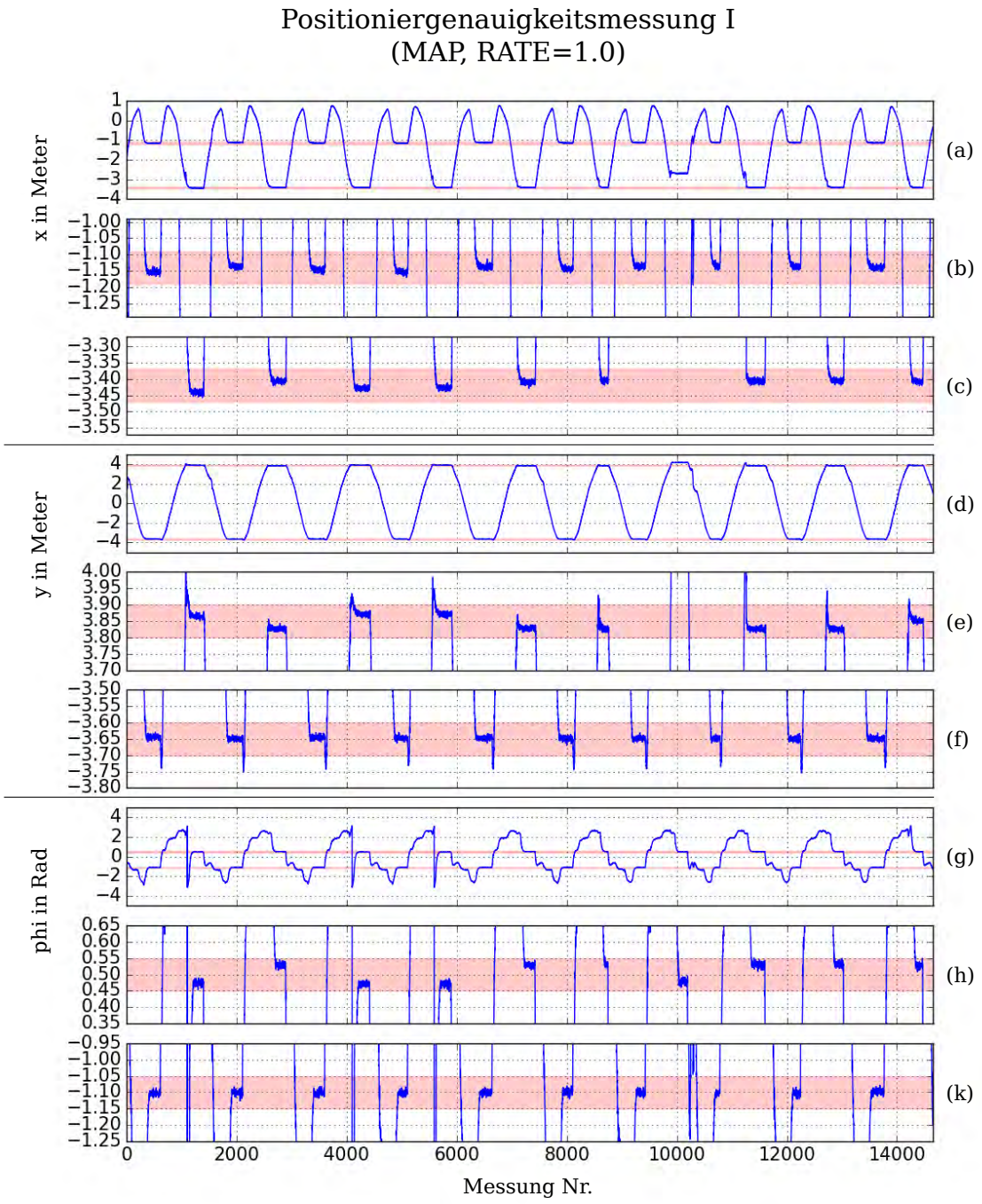


Abbildung 39: Positioniergenauigkeitsmessung I: Translatorische (Grafen a und d) und rotatorische (Graf g) Positioniergenauigkeit beim mehrmaligen Anfahren zweier Posen. Die schattierten Fehlerschranken haben eine Breite von 10 cm beziehungsweise 0,1 rad. Die Grafen b, c, e, f, h, und k sind Vergrößerungen der jeweils darüber abgebildeten Grafen a, d und g. An Stellen, an denen die Funktion ein Plateau bildet, verharrt der Roboter in seiner Pose. Dies sind die Posen A und B aus Abbildung 36. [eigene Abbildung]

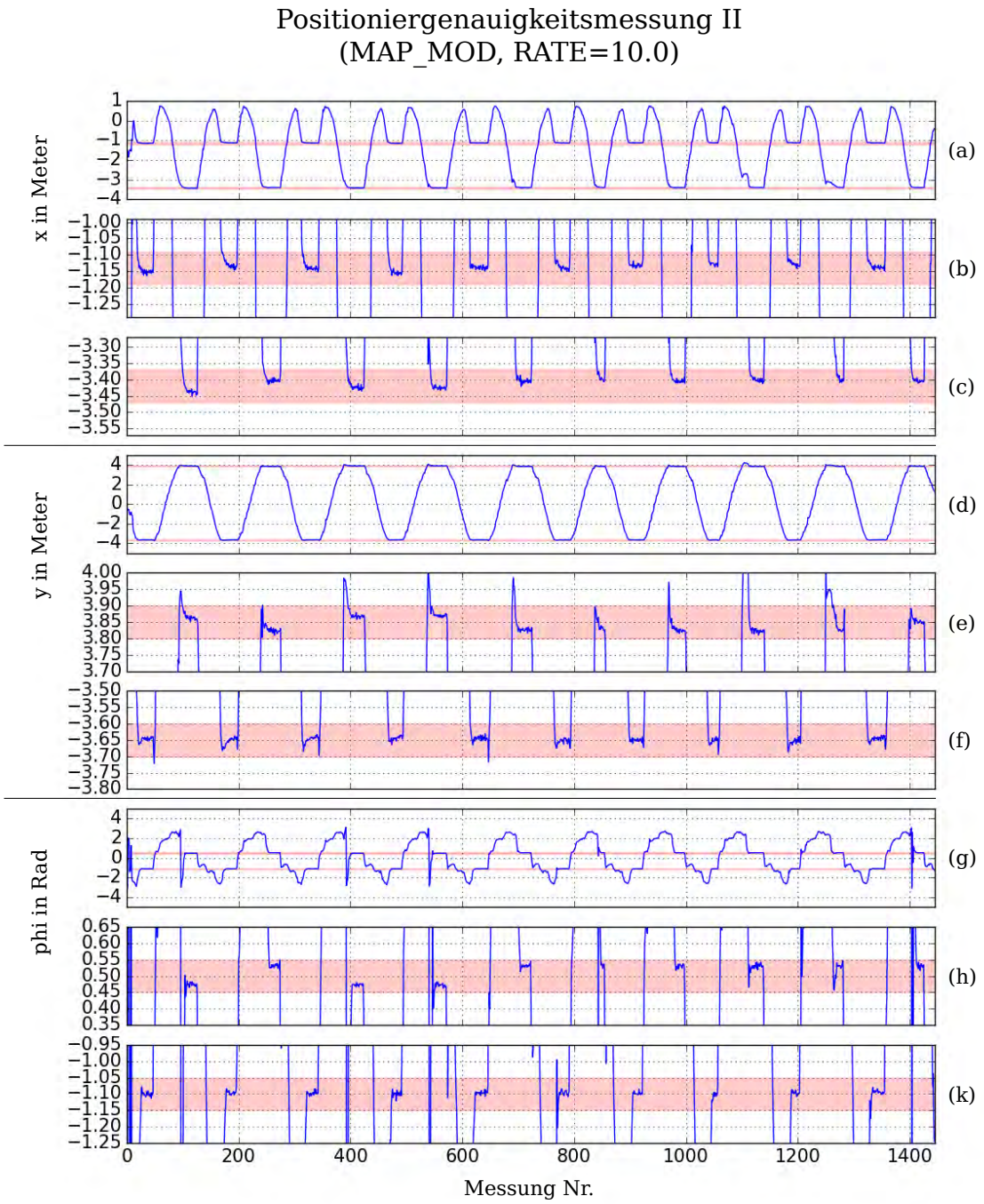


Abbildung 40: Positioniergenauigkeitsmessung II: Translatorische (Grafen a und d) und rotatorische (Graf g) Positioniergenauigkeit beim mehrmaligen Anfahren zweier Posen mit erhöhter Geschwindigkeit und modifizierter Umgebungskarte. Die schattierten Fehlerschranken haben eine Breite von 10 cm beziehungsweise 0,1 rad. Die Grafen b, c, e, f, h, und k sind Vergrößerungen der jeweils darüber abgebildeten Grafen a, d und g. An Stellen an denen die Funktion ein Plateau bildet, verharrt der Roboter in seiner Pose. Dies sind die Posen A und B aus Abbildung 36. [eigene Abbildung]

6.3 Timinganalyse

Weitere Aufschlüsse über die Funktionsweise und Qualität des Lokalisierungssystems bringt eine Timinganalyse. Die Anforderungsliste beschreibt eine Aktualisierungsrate von mindestens 20 Hz. Das Lokalisierungssystem wird für diese Analyse mit den Daten aus der Positioniergenauigkeitsmessung gespeist. Um herauszufinden, welche Bereiche der Lokalisierungssoftware die meiste Zeit in Anspruch nehmen, wird die Bearbeitungszeit folgender Filterschritte aufgezeichnet:

- OUTPUT: Ausgeben und Visualisieren der Partikelwolke beziehungsweise Berechnung und Übergabe des Filterstatus (Posenschätzung, Varianz der Partikel, ...) an die Benutzerschnittstelle.
- ODOM: Verschieben der Partikel um die Differenz aus dem letzten und dem aktuellen Odometriesignal.
- RESAMPLER: Der Resampling Algorithmus zum Umverteilen der Partikel.
- LASER: Gewichtung der Partikel auf Basis der aktuellen Laserscannermessung und der Umgebungskarte.

Abbildung 41 zeigt die Ergebnisse aus der Timinganalyse. Nicht jeder Filterschritt wird in jeder Filteriteration abgearbeitet. Sensordaten werden nur dann verarbeitet, wenn sie dem Partikelfilter vorliegen. Auch der Resampling-Schritt wird nicht in jeder Filteriteration durchlaufen (siehe Algorithmus 1). Ein Richtwert für eine maximale Durchlaufzeit des Lokalisierungssystems kann dennoch durch Addition der durchschnittlichen Bearbeitungszeiten der einzelnen Filterschritte errechnet werden. Sie beträgt in dieser Messung zirka 33 ms. Die geforderte Aktualisierungsrate von 20 Hz bedeutet eine maximale Durchlaufzeit von 50 ms. Im Durchschnitt kann die geforderte Aktualisierungsrate somit erreicht werden.

Eine weitere Erkenntnis aus dieser Messung ist der hohe Zeitbedarf des LASER-Schritts. Nach einem ersten Durchlauf der Timinganalyse kann durch Optimierungen im Quellcode[28] die mittlere Iterationszeit von 21 ms auf die in Abbildung 41 abgebildeten 16 ms verringert werden.

[28] URL: `https://github.com/amndan/ohm_pf.git`
Commit-ID: `cb21-a7c3-d608-e28d-a3ef-4036-9106-cc6b-1e27-b1ed`

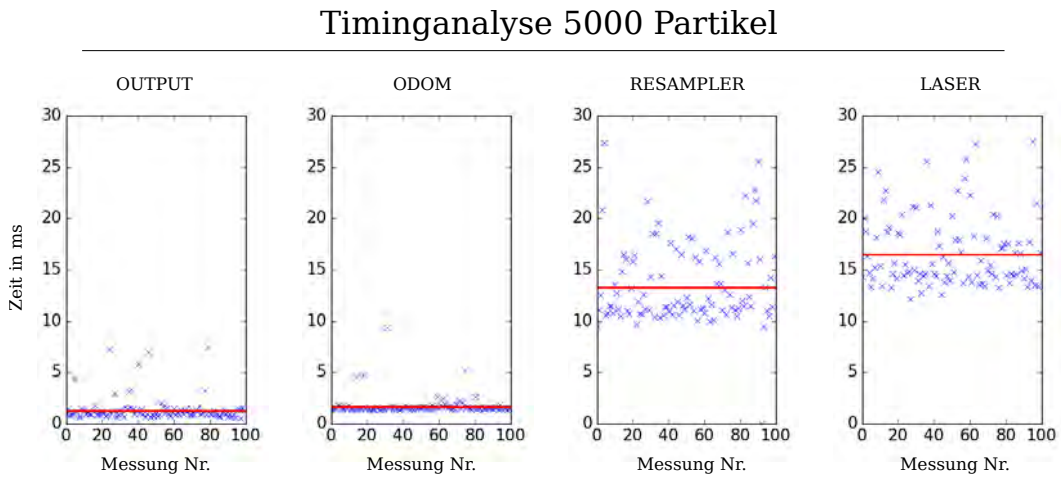


Abbildung 41: Timinganalyse 5000 Partikel: Die Bearbeitungszeit der vier hauptsächlichen Filterschritte bei einer Partikelzahl von 5000. Die rote Linie zeigt den Mittelwert der Messdaten an. Zur einfacheren Darstellung sind nur 100 Messpunkte gezeichnet. Der Mittelwert wird jeweils aus der folgenden Anzahl an Messungen gebildet: OUTPUT - 5051 Messungen, ODOM - 5053 Messungen, RESAMPLER - 478 Messungen, LASER - 2527 Messungen. [eigene Abbildung]

Die Verarbeitungszeit aller Filterschritte hängt linear von der Partikelanzahl ab. Die vorherige Messung beschreibt die Verarbeitungszeit mit 5000 Partikeln. Bei Verringerung der Partikelanzahl um die Hälfte auf 2500 Partikel, sinkt auch die mittlere Verarbeitungszeit von 33 ms auf zirka 15 ms (siehe Abbildung 42).

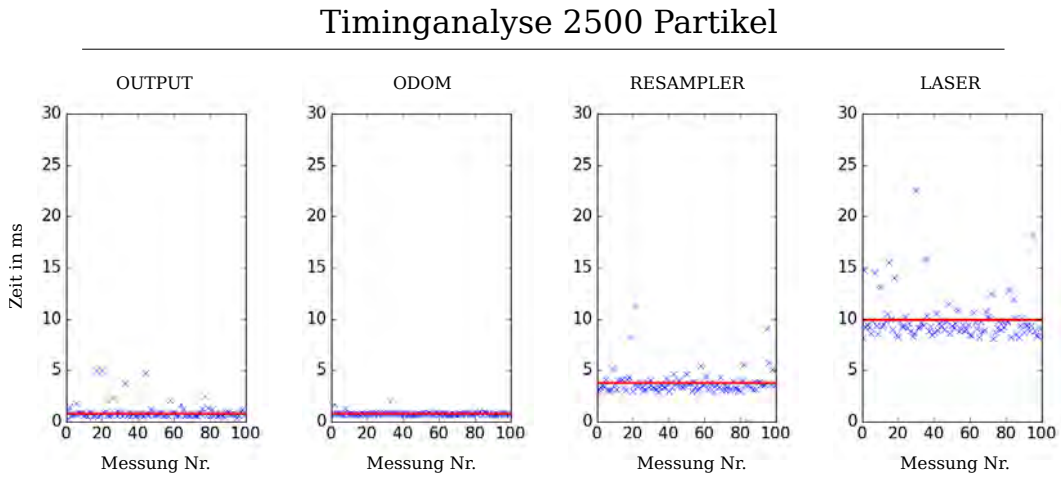


Abbildung 42: Timinganalyse 2500 Partikel: Die Bearbeitungszeit der vier hauptsächlichen Filterschritte bei einer Partikelanzahl von 2500. Die rote Linie zeigt den Mittelwert der Messdaten an. Zur einfacheren Darstellung sind nur 100 Messpunkte gezeichnet. Der Mittelwert wird jeweils aus der folgenden Anzahl an Messungen gebildet: OUTPUT - 1966 Messungen, ODOM - 1967 Messungen, RESAMPLER - 185 Messungen, LASER - 984 Messungen. [eigene Abbildung]

Die Dauer des LASER-Schritts hängt zusätzlich von der Auflösung des Laserscanners ab. Laserdaten mit zu hoher Winkelauflösung müssen ausgedünnt werden, damit die geforderte Aktualisierungsrate erreicht werden kann. Abbildung 43 zeigt den linearen Zusammenhang der zu verarbeitenden Entfernungsmessungen pro Laserscannermessung und der Verarbeitungszeit des LASER-Schritts. Hierbei werden wieder 5000 Partikel verwendet.

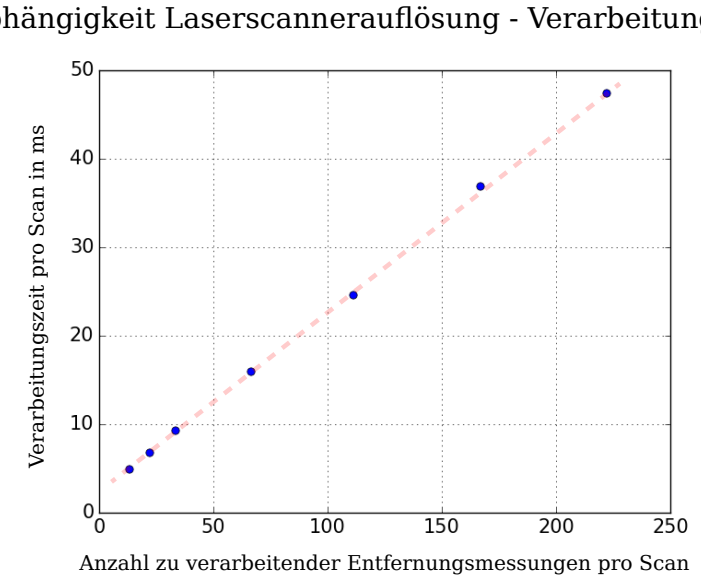


Abbildung 43: Einfluss der Scanneraufl
ösung: Die zu verarbeitenden Entfernungsmessungen pro Scannermessung aufgetragen über der Verarbeitungszeit des LASER-Schritts. (Ausgleichsgerade empirisch ermittelt) [eigene Abbildung]

Das Verwerfen von Entfernungsmessungen hat einen Informationsverlust über die Umgebung zufolge. Um eine möglichst genaue Posenschätzung zu erhalten, müssen so viele Entfernungsmessungen verarbeitet werden, wie mit dem verwendeten Prozessor möglich.

6.4 Anwendungstest Labor Mobile Robotik

Das Lokalisierungssystem findet erfolgreich Anwendung auf verschiedenen Roboterplattformen im Labor für mobile Robotik der TH-Nürnberg. Bei den auf den Seiten 69 und 70 dargestellten[29] Roboterplattformen übernimmt das in dieser Arbeit entwickelte System die Lokalisierung des jeweiligen Roboters. Die kontinuierliche Integration des Lokalisierungssystems in unterschiedliche Roboterplattformen ermöglicht Optimierungen und Verbesserungen der Software.

Die Verwendung des Systems auf verschiedenen Roboterplattformen lässt keine quantitativen Aussagen über dessen Funktion zu. Jedoch wird die hohe Flexibilität des Systems dadurch sichtbar. Die Roboterplattformen besitzen verschiedene Recheneinheiten, Sensorkonzepte und Antriebsmodelle. Nach jeweiliger Anpassung der Parametrierung des Lokalisierungssystems kann es auf allen genannten Plattformen zu deren Lokalisierung genutzt werden.

[29] Quelle: Bei den auf den Seiten 69 und 70 befindlichen Abbildungen handelt es sich um eigene Abbildungen.

LABORFÜHRUNGSROBOTER	KOBUKI-FLOTTE
	
VERWENDUNGSZWECK	
Der Laborführungsroboter soll dem Laborbesucher eine Führung durch das Labor Mobile Robotik der TH-Nürnberg ermöglichen. Der Besucher folgt der Roboterplattform auf dem Weg durch das Labor. An bestimmten Punkten hält der Roboter an und erklärt dem Besucher was dort zu sehen ist, zum Beispiel ausgestellte Forschungsprojekte.	Die Kobuki-Flotte ermöglicht Forschungsaktivitäten im Bereich Multiroboterbetrieb. Themenschwerpunkte hierbei sind beispielsweise eine kooperative Lokalisierung oder die Multiroboternavigation. Die Flotte ist in Zusammenarbeit mit der Firma Bosch entstanden.
ROBOTERPLATTFORM UND ANTRIEBSMODELL	
• Kobuki-Roboterplattform • Differenzieller Antrieb	• Kobuki-Roboterplattform • Differenzieller Antrieb
RECHENEINHEIT	
• Einplatinenrechner ODROID XU3 • Prozessorarchitektur: ARM	• Einplatinenrechner ODROID XU4 • Prozessorarchitektur: ARM
SENSORKONZEPT ZUR LOKALISIERUNG	

• 2D-Laserscanner SICK LMS100
• Odometriedaten aus Antrieb
	• 2D-Laserscanner RPLIDAR
	• Funkmodule Neocortec
	• Fusionierte Odometriedaten aus:
	  ○ IMU BNO055
	  ○ Odometriedaten aus Antrieb

BESONDERHEIT LOKALISIERUNGSSTRATEGIE	
• Verwendung des Lokalisierungssystems auf einem Einplatinenrechner	• Vorfusion von Odometrie- und IMU-Daten mittels eines EKFs. • Verwendung eines Low-Cost-Laserscanners • Integration von Funksensorik in den Lokalisierungsalgorithmus: Jeder Roboter in der Flotte trägt ein Funkmodul mit sich. Ein nicht lokalisierter Roboter kann damit, durch die Auswertung empfangener Funksignalstärken anderer bereits lokalisierter Roboter, Annahmen über seine eigene Position treffen.

YOUBOT	BODENROLLER BOSCH
	
VERWENDUNGSZWECK	
Die youBot-Roboterplattform wird als Wettkampfroboter des Teams autonohm der TH-Nürnberg im RoboCup@Work-Wettbewerb sowie in der „European Robotics League“ eingesetzt. Bei diesen Wettbewerben wird ein industrielles Umfeld generiert, in dem der Roboter autonom navigieren und mit Objekten interagieren muss.	Der Bodenroller stellt den Prototypen eines autonomen Transportsystem (ATV) der Firma Bosch dar. Er wurde von Bosch in Zusammenarbeit mit der TH-Nürnberg entwickelt. Er dient zur Entwicklung eines ATV mit dem Fertigungsgüter in der Fabrikhalle autonom transportiert werden können.
ROBOTERPLATTFORM UND ANTRIEBSMODELL	
• KUKA-youBot Roboterplattform • Omnidirektionaler Antrieb	• Eigenentwicklung • Differenzieller Antrieb
RECHENEINHEIT	
• PC (Intel i7) • Prozessorarchitektur: x86	• Industrie-PC (Intel i7) • Prozessorarchitektur: x86
SENSORKONZEPT ZUR LOKALISIERUNG	

• 2D-Laserscanner SICK TiM571 (vorne)
• 2D-Laserscanner SICK TiM571 (hinten)
• Fusionierte Odometriedaten aus:
  ○ IMU BNO055
  ○ Odometriedaten Antrieb
	• 2D-Sicherheitslaserscanner OMRON OS32C
	• RGB-Kamera Logitech C920 (1080p)
	• Fusionierte Odometriedaten aus:
	  ○ IMU BNO055
	  ○ Odometriedaten Antrieb

BESONDERHEIT LOKALISIERUNGSSTRATEGIE	
• Vorfusion von Odometrie- und IMU-Daten mittels eines EKFs. • Dual-Scanner-Betrieb: Gleichzeitige Verwendung zweier Laserscanner zur Lokalisierung. • Omnidirektionales Antriebsmodell durch Mecanum-Antrieb.	• Vorfusion von Odometrie- und IMU-Daten mittels eines EKFs. • Integration einer 2D-Deckenkamera (CeilCam) in den Lokalisierungs-algorithmus: Eine an die Decke gerichtete Kamera lässt den Roboter Annahmen über seine derzeitige, absolute Position treffen. Voraussetzung hierfür ist ein zuvor aufgenommenes und abgespeichertes Abbild der Deckenstrukturen am Einsatzort.

7 Zusammenfassung und Ausblick

Die beiden nachfolgenden Kapitel geben ein Resümee über die geleistete Arbeit und greifen weiterführende, mit der vorliegenden Arbeit zusammenhängende Themenfelder auf.

7.1 Fazit der Arbeit

In dieser Arbeit wurde die Entwicklung eines Lokalisierungssystems für fahrerlose Transport- und mobile Robotersysteme behandelt. Ziel war es, eine zuverlässige Roboterlokalisierung im Innenbereich in einem Genauigkeitsbereich weniger Zentimeter beziehungsweise Grad zu ermöglichen.

Nach der Definition der Anforderungen an ein solches System und seine Umgebung beziehungsweise einer Literaturstudie zum Thema Roboterlokalisierung, konnte ein passendes, aus mehreren Sensoren bestehendes, Sensorkonzept als Grundlage des Lokalisierungssystems ausgewählt werden. Zur Fusionierung der Sensordaten wurde der Partikelfilteralgorithmus ausgewählt. Er erfüllt die Anforderungen und bietet die gewünschte Flexibilität. Nachdem die theoretischen Grundlagen des Algorithmus nachvollzogen wurden, konnte ein Entwurf erstellt und daraufhin das Lokalisierungssystem implementiert werden.

Die durchgeführten Tests zur Verifikation der korrekten Funktion des Lokalisierungssystems zeigten, dass das System in der Lage ist, eine Roboterlokalisierung unter Erfüllung aufgestellten Anforderungen zu ermöglichen. Auch weniger verbindliche Anforderungen, wie das Realisieren einer globalen Lokalisierung, die Relokalisierung nach einer Fehllokalisierung des Roboters oder die korrekte Funktionalität des Lokalisierungssystems trotz Veränderungen in der Einsatzumgebung, können mit dem entwickelten System bedient werden. Durch Anwendungstests in realen Umgebungen konnte die korrekte Funktion des Lokalisierungssystems auf verschiedenen Hardwareplattformen und in verschiedenen Umgebungen (industrielles Umfeld, Labor-Umfeld) bestätigt werden. Wohingegen Tests mit simulierten Daten die Genauigkeit des Systems darstellen und verifizieren konnten.

Das entwickelte Lokalisierungssystem kann auf fahrerlosen Transportsystemen sowie auf anderen mobilen Robotersystemen eingesetzt werden. Durch die Verwendung eines Partikelfilters ist es nicht an ein bestimmtes Sensorkonzept gebunden und dadurch hochflexibel einsetzbar. Die modulare Auslegung der Software lässt die Integration weiterer Sensor- oder Antriebsmodelle zu, wodurch das System für das Robotik-Labor der TH-Nürnberg die Grundlage für weitere, die Roboterlokalisierung betreffende Herausforderungen bilden kann.

7.2 Zukünftige Arbeiten

Während der Erarbeitung dieser Masterarbeit sind Themen entstanden, die im Zuge dessen nicht behandelt werden konnten. Sie werden im Folgenden dokumentiert, um anschließenden Arbeiten einen Anknüpfungspunkt zu bieten.

Multiroboterlokalisierung

Roboterflotten bestehen aus mehreren Robotern. Das Zusammenwirken einzelner Roboter in einer Flotte wird als Multiroboterbetrieb bezeichnet. Multiroboterlokalisierung spezialisiert diesen Begriff auf das Lokalisierungsproblem. In Kapitel 6.4 wurde die Kobuki-Flotte vorgestellt. Durch die Integration von Funksensorik und die Verwendung der Poseninformation

eines anderen Roboters (siehe Kapitel 6.4) wurde in dieser Arbeit die Multiroboterlokalisierung angeschnitten. Diesbezüglich können weitaus effizientere und komplexere Methoden die Robustheit und Genauigkeit der Lokalisierung verbessern (siehe [6]).

Roboterkontrollarchitektur

Roboterkontrollarchitekturen bilden die Kontrollschicht für einen autonom arbeitenden Roboter. In einer solchen Architektur werden Entscheidungen über das Verhalten des Roboters getroffen. Die Integration des Lokalisierungssystems in eine solche Roboterkontrollarchitektur könnte beispielsweise die Aktivierung einer Explorationsphase beim Verlust der Lokalisierung ermöglichen. Kann die Lokalisierung wieder aufgenommen werden, schaltet die Kontrollarchitektur wieder in den Normalbetrieb um. Ein solches Verhalten kann die Ausfallzeiten eines Robotersystems drastisch verringern, da ein Verlust der Pose nicht die Inaktivität des Roboters zur Folge hätte.

TSD-Map

Der im Labor für mobile Robotik der TH-Nürnberg entwickelte Ansatz zur 2D-Kartenerstellung (SLAM) namens ohm-tsd-slam verwendet eine besondere Repräsentation der Umgebungskarte. Diese Repräsentation besteht nicht, wie bei vielen SLAM-Ansätzen, aus einer reinen *Occupancy-Grid-Map*, sondern aus *Truncated Signed Distance Functions*. Diese Repräsentation ähnelt der in Kapitel 4.3 beschriebenen Wahrscheinlichkeitskarte sehr stark. So könnte diese Art der Umgebungskartenrepräsentation direkt für das Sensormodell des 2D-Laserscanners verwendet werden.

Dynamische Karten

Die Repräsentation der Umgebung wird dem Lokalisierungssystem derzeit bei der Initialisierung mitgeteilt und ist unveränderlich. Veränderungen in der Umgebung können durch Laserscanner detektiert und in eine dynamische Umgebungskarte eingearbeitet werden. Erhält auch das Lokalisierungssystem Zugriff auf eine solche Umgebungskarte, kann sich die Qualität und Robustheit der Lokalisierung verbessern, da die Datenbasis eine aktuellere ist. In hochdynamischen Umgebungen könnte dieses Vorgehen für eine erfolgreiche Lokalisierung erforderlich sein.

Das im obenstehenden Absatz erwähnte gemeinsame Nutzen der Umgebungskartenrepräsentation von Lokalisierungsalgorithmus und SLAM-Algorithmus könnte hierbei einen Vorteil bezüglich der Recheneffizienz bringen.

Partikelanzahl

Die Partikelanzahl des Partikelfilters aus dieser Arbeit ist konstant. Um die globale Lokalisierung zu ermöglichen, wurde meist eine hohe Anzahl an Partikeln ausgewählt (≈ 5000). Nur so kann die gesamte Einsatzumgebung mit Partikeln abgedeckt werden. Eine variable Partikelanzahl kann die Genauigkeit des Partikelfilters und die Erfolgsquote bei der globalen Lokalisierung erhöhen. Während der globalen Lokalisierungsphase kann eine geringere Aktualisierungsrate des Partikelfilters zugunsten einer hohen Partikelanzahl in Kauf genommen werden, um die gesamte Umgebung mit einer hohen Partikeldichte abzutasten. Ist die Pose ermittelt, kann die Partikelanzahl reduziert werden, um eine hochfrequente Verarbeitung der Sensorsignale und damit eine hohe Genauigkeit zu gewährleisten.

Literatur

[1] D. Ammon. »Forschungsbericht I: Entwicklung eines Lokalisierungssystems für fahrerlose Transportsysteme«. Apr. 2016.

[2] D. Ammon. »Low-Cost Sensor Concept for Localization of Individual Autonomous Industrial Transportation Robots in a Robot Network«. In: *Applied Research Conference*. 2015, S. 418–422.

[3] W. Burgard u. a. »A comparison of SLAM algorithms based on a graph of relations«. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Okt. 2009, S. 2089–2095. DOI: 10.1109/IROS.2009.5354691.

[4] C. Chen u. a. »Low cost IMU based indoor mobile robot navigation with the assist of odometry and Wi-Fi using dynamic constraints«. In: *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*. Apr. 2012, S. 1274–1279. DOI: 10.1109/PLANS.2012.6236984.

[5] F. Dias u. a. »Mobile Robot Localisation for Indoor Environments Based on Ceiling Pattern Recognition«. In: *2015 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. Apr. 2015, S. 65–70. DOI: 10.1109/ICARSC.2015.32.

[6] Randolf Ebelt. *Neue Ansätze für die verteilte kooperative räumliche Lokalisierung in drahtlosen Sensornetzwerken*. Dr. Hut, 2014. ISBN: 978-3-8439-1457-4, 3-8439-1457-5.

[7] R. J. Fontana, E. Richley und J. Barney. »Commercialization of an ultra wideband precision asset location system«. In: *2003 IEEE Conference on Ultra Wideband Systems and Technologies*. Nov. 2003, S. 369–373. DOI: 10.1109/UWBST.2003.1267866.

[8] C. Forster, M. Pizzoli und D. Scaramuzza. »SVO: Fast semi-direct monocular visual odometry«. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Mai 2014, S. 15–22. DOI: 10.1109/ICRA.2014.6906584.

[9] D. Fox u. a. »Monte Carlo Localization: Efficient Position Estimation for Mobile Robots«. In: *Proc. of the National Conference on Artificial Intelligence*. 1999, S. 343–349.

[10] Erich Gamma u. a. *Design Patterns: Elements of Reusable Object-Oriented Software*. Springer, 1994. ISBN: 9780201633610.

[11] S. Gezici u. a. »Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks«. In: *IEEE Signal Processing Magazine* 22.4 (Juli 2005), S. 70–84. ISSN: 1053-5888. DOI: 10.1109/MSP.2005.1458289.

[12] G. Grisetti, C. Stachniss und W. Burgard. »Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters«. In: *IEEE Transactions on Robotics* 23.1 (Feb. 2007), S. 34–46. ISSN: 1552-3098. DOI: 10.1109/TRO.2006.889486.

[13] U. Grossmann, M. Schauch und S. Hakobyan. »RSSI based WLAN Indoor Positioning with Personal Digital Assistants«. In: *4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. Sep. 2007, S. 653–656. DOI: 10.1109/IDAACS.2007.4488503.

[14] D. Hahnel u. a. »Mapping and localization with RFID technology«. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Bd. 1. Apr. 2004, 1015–1020 Vol.1. DOI: 10.1109/ROBOT.2004.1307283.

[15] Joachim Hertzberg, Kai Lingemann und Andreas Nüchter. *Mobile Roboter: Eine Einführung aus Sicht der Informatik*. Springer, 2010. ISBN: 9783642053573.

[16] R E Kalman. »A New Approach to Linear Filtering and Prediction Problems«. In: *J Basic Eng* 82 (1960), S. 35–45.

[17] P. Koch u. a. »Multi-robot Localization and Mapping Based on Signed Distance Functions«. In: *2015 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. Apr. 2015, S. 77–82.

[18] Rainer Kümmerle u. a. »On measuring the accuracy of SLAM algorithms«. In: *Autonomous Robots* 27.4 (2009), S. 387. ISSN: 1573-7527. DOI: 10.1007/s10514-009-9155-6.

[19] A. A. Makarenko u. a. »An experiment in integrated exploration«. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Bd. 1. 2002, 534–539 vol.1. DOI: 10.1109/IRDS.2002.1041445.

[20] K. Nagatani u. a. »Improvement of odometry for omnidirectional vehicle using optical flow information«. In: *International Conference on Intelligent Robots and Systems (IROS)*. Bd. 1. 2000, S. 468–473. DOI: 10.1109/IROS.2000.894648.

[21] L. Ojeda u. a. »Current-Based Slippage Detection and Odometry Correction for Mobile Robots and Planetary Rovers«. In: *IEEE Transactions on Robotics* 22.2 (Apr. 2006), S. 366–378. ISSN: 1552-3098. DOI: 10.1109/TRO.2005.862480.

[22] Helmuth Partsch. *Requirements-Engineering systematisch: Modellbildung für softwaregestützte Systeme*. Springer, 2010. ISBN: 9783642053573.

[23] J. Seitz, T. Vaupel und J. Thielecke. »Wi-Fi azimuth and position tracking: Signal propagation, modeling and evaluation«. In: *2013 16th International Conference on Information Fusion (FUSION)*. 2013, S. 1479–1486.

[24] Jochen Seitz, Thorsten Vaupel und Jorn Thielecke. »A particle filter for Wi-Fi azimuth and position tracking with pedestrian dead reckoning«. In: *2013 Workshop on Sensor Data Fusion: Trends, Solutions, Applications, SDF 2013* (2013). DOI: 10.1109/SDF.2013.6698251.

[25] S. Thrun, W. Burgard und D. Fox. *Probablistic Robotics*. Massachusetts Institute of Technology, 2006. ISBN: 9780262201629.

[26] Sebastian Thrun. »A Probabilistic On-Line Mapping Algorithm for Teams of Mobile Robots«. In: *The International Journal of Robotics Research* 20.5 (2001), S. 335–363. DOI: 10.1177/02783640122067435.

[27] Sebastian Thrun u. a. »Robust Monte Carlo localization for mobile robots«. In: *Artificial Intelligence* 128.1 (2001), S. 99–141.

[28] Y. Wang u. a. »Bluetooth positioning using RSSI and triangulation methods«. In: *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*. Jan. 2013, S. 837–842. DOI: 10.1109/CCNC.2013.6488558.

[29] Gerhard Winkler. *Image Analysis, Random Fields and Dynamic Monte Carlo Methods A Mathematical Introduction*. Springer, 1995. ISBN: 978-3-642-97524-0.

[30] H. Zhang u. a. »Localization and navigation using QR code for mobile robot in indoor environment«. In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Dez. 2015, S. 2501–2506. DOI: 10.1109/ROBIO.2015.7419715.

Anhang

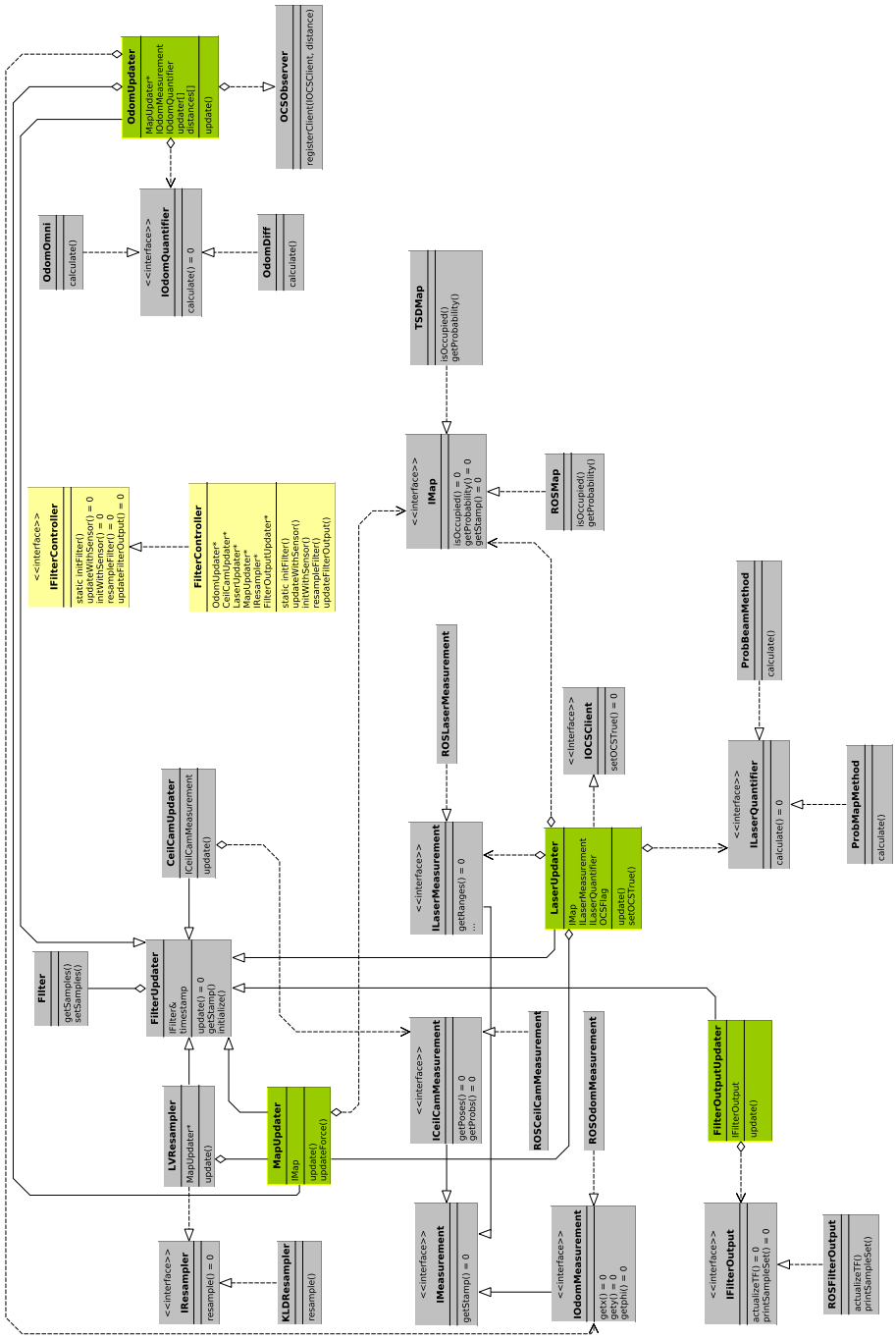


Abbildung 44: Softwareentwurf UML: Es sei darauf hingewiesen, dass dieser Entwurf nicht den tatsächlichen Softwarestand dokumentiert, sondern den Entwicklungsprozess und den dabei entstandenen Entwurf visualisiert. Die Software ist, wie in Kapitel 5.4 beschrieben, durch die dem Quellcode beiliegende Doxygen-Dokumentation dokumentiert.

Tabelle 2: Anforderungsliste: Anforderungsliste des Lokalisierungssystems. Legende der Verbindlichkeiten: P = Pflicht, W = Wunsch

Name	Beschreibung	Verb.
Funktionale Systemanforderungen		
Tracking	Das System muss die Bewegung eines zu lokalisierenden Gefährtes erkennen und die daraus resultierende Positionsänderung abbilden können. (Grundsätzlicher Lokalisierungsgedanke)	P
Passives System	Das System wirkt nicht unmittelbar auf den Betriebs- beziehungsweise Bewegungszustand des zu lokalisierenden Gefährtes ein.	P
Globale Lokalisierung	Das System soll eine globale Lokalisierung in bekannter Umgebung ermöglichen	W
Relokalisierung	Das System soll eine fehlerhafte Ortung detektieren und eine Relokalisierung einleiten können.	W
Dynamische Umgebung	Das System soll auch bei Veränderungen in der zuvor kartierten Umgebung funktionieren.	W
Anforderungen an die Systemumgebung		
Maximale translatorische Geschwindigkeit	Die maximale translatorische Geschwindigkeit des zu lokalisierenden Gefährtes muss auf 1 m/s begrenzt sein.	P
Maximale rotatorische Geschwindigkeit	Die maximale rotatorische Geschwindigkeit des zu lokalisierenden Gefährtes muss auf 82 °/s begrenzt sein.	P
Einsatzort	Das System wird ausschließlich im Innenbereich eingesetzt.	P
Kartierung	Vor dem Einsatz des Systems muss eine Umgebungskarte des Einsatzortes vorliegen. Es findet keine Exploration oder Kartierung durch das Lokalisierungssystem statt.	P
Anforderungen an die Systemstruktur (Allgemein)		
Sensordatenfusion	Das System muss die gleichzeitige Integration mehrerer Sensoren ermöglichen.	P
Partikelfilter	Das System muss als Partikelfilter ausgelegt werden.	P
Visualisierung	Das System muss den Lokalisierungszustand für den Benutzer visualisieren können.	P
Konfiguration	Das System muss durch den Benutzer konfiguriert werden können.	P
Anforderungen an die Systemstruktur (Software)		
Programmiersprache	Die Software muss in C++ erstellt werden.	P
Softwarebibliothek	Die Software muss als Bibliothek ausgelegt werden.	P
Softwaredokumentation	Die Software muss mittels Doxygen dokumentiert werden.	P
Unabhängigkeit ROS	Die Bibliothek muss unabhängig von ROS programmiert werden.	P
Einbindung ROS	Die Bibliothek muss in eine ROS-Umgebung integriert werden können.	P
Leistungsanforderungen		
Genauigkeit translatorisch	Das System muss einen translatorischen Positionierungsfehler geringer 10 cm erreichen.	P
Genauigkeit rotatorisch	Das System muss einen rotatorischen Positionierungsfehler geringer 6° erreichen.	P
Aktualisierungsrate	Das System muss eine Aktualisierungsrate von mindestens 20 Hz erreichen.	P

Tabelle 3: Beschreibung der ROS-Parameter Teil I

Name	Description	Typical Values
tfFixedFrame	Tf-Frame-Name of fixed frame. Often called "map".	map
tfOdomFrame	Tf-Frame-Name of odom frame. Often called "odom".	odom
tfBaseFootprintFrame	Tf-Frame-Name of robots base-footprint frame. Often called "base_link".	base_link
topMapSrv	Service topic for initially receiving the map.	static_map
topScan	Topic of scan data. For multiple scanners separate topics with ";". Additionally adjust count lasers according to the number of scanners to be used.	sick;hokuyo
countLasers	Amount of used scanners. Must match number of topics in topScan param.	1 - 2
topOdometry	Topic of odometry data.	odom
top2dPoseEstimate	Topic for sending an initial pose to the filter.	initialpose
topCeilCam	Topic for sending absolute pose measurements to the filter. For example measurements from the CeilCam package or GPS data.	pose_meas
topParticleCloud	Topic for visualization of the filters particlecloud.	particlecloud
topProbPose	Topic for the quality of the filters pose estimation. 0.0 indicates bad quality - 1.0 indicates good quality. The Filter is not able to detect localization failures. This value just represents an indicator for the spreading of all particles.	prob_pose
filterLoopRate	The filters loop rate in seconds. If filters loop duration is longer, a debug message is printed. If filters loop duration is under filterLoopRate the node sleeps.	0.02
outputIntervallFilter	The interval the filters actualizes its pose estimation and particle cloud visualization.	0.05
skipParticleForGui	Subsampling of particlecloud for visualization. 0 for no subsampling.	0 - 100
samplesMax	Number of particles to be used.	5000
additionalTranslationalNoise	Adds additional translational noise to each particle at resampling step. This increases spreading of particles.	0.05
additionalRotationalNoise	Adds rotational translational noise to each particle at resampling step. This increases spreading of particles.	0.01
resamplingIntervallFilter	Minimum elapsed time before resampling step is done in seconds. Higher resampling interval ensures that resampling step is based on more measurement steps. So a low resampling interval leads to resampling steps based on single measurements and therefore to low localization stability. A high resampling interval leads to bad accuracy caused by high spreading of particles due to movement.	0.1 - 1.5

Tabelle 4: Beschreibung der ROS-Parameter Teil II

Name	Description	Typical Values
resamplingMethod	Method for resampling particles. STD means standard resampling - LV means low variance resampling. Low variance resampling leads to less convergence speed.	STD
lowVarianceFactor	Amount of particles drawn in one iteration when low variance resampling is used.	3 - 10
initMode	Initialization mode. GL for global localization - POSE for initialization at given pose.	GL
initX	Initial particle cloud position x in meter if filter initMode is POSE.	0.0
initY	Initial particle cloud position y in meter if filter initMode is POSE.	0.0
initPhi	Initial particle cloud position yaw in rad if filter initMode is POSE.	0.0
initSigmaTrans	Initial translational particle cloud spreading (standard deviation in meter) if filter initMode is POSE.	0.2
initSigmaRot	Initial rotational particle cloud spreading (standard deviation in rad) if filter initMode is POSE.	0.3
subsamplingRateLaser	Linear subsampling of laser scan data before integration. Higher value leads to slower convergence speed but also affects pose accuracy. Choosing high subsampling rate leads to bad accuracy.	3 - 5
uncertaintyLaser	Additional uncertainty of laser measurements. Higher values lead to a slower convergence speed.	0.1 - 0.8
maxDistanceProbMap	Distance for inflation of occupied parts at creation of probability field in cells. Depends on map resolution. Higher value leads to slower convergence speed of filter.	5 - 30
OCSRotToTransFactor	Factor for transforming radial motion into linear motion. OCS limits are in meter - rotational movement is transformed with this factor to linear movement.	6 - 10
OCSThresholdLaser	Minimum absolute movement of robot in meter before a new laser measurement gets processed. Prevents the Filter from updating particles with redundant information.	0.005
OCSThresholdOdom	Minimum absolute movement of robot in meter before a new odometry measurement gets processed. If odom data is noisy no odometry update should be done if the robot don't moves.	0.001
OCSThresholdResampler	Minimum absolute movement of robot in meter before resampling step takes place. Can be used to delaying the resampling step according to movement of the robot. This the same effect as a higher resampling interval, but it's connected with the movement of the robot.	0.050
odomModel	Used odometry model. 0 means differential drive - 1 means omni drive.	1
odomAlpha1	DIFF: rot error from rot motion; OMNI: x error from x motion	0.2
odomAlpha2	DIFF: rot error from trans motion; OMNI: y error from y motion	0.2
odomAlpha3	DIFF: trans error from trans motion; OMNI: yaw error from yaw motion	0.2
odomAlpha4	DIFF: trans error from trot motion; OMNI: not used here	0.2