Simultaneous Localization and Mapping mit Multilevel- und Multiskalenmethoden

Leon Fiehn

Geboren am 21. August 1999 in Köln

9. Dezember 2022

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Michael Griebel

Zweitgutachter: Prof. Dr. Marc Alexander Schweitzer

Institut Für Numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Inhaltsverzeichnis

1 Einleitung

Diese Bachelorarbeit beschäftigt sich mit dem Simultaneous Localization and Mapping Problem, kurz SLAM, und einer Variante seiner Lösung. Simultaneous Localization and Mapping findet Anwendung in der autonomen Navigation von einfachen Haushaltsrobotern in Gebäuden, über Autos im Straßenverkehr, bis hin zu Drohnen in der Luft. Dafür werden verschiedene Sensoren, wie Kameras, oder Laser verwendet, mit deren Daten zum einen ein möglichst genaues Bild, bzw. eine Karte der Umgebung erstellt wird und zum anderen die Position des Roboters selbst in diesem Bild verortet wird. Da sowohl die Erfassung der Umgebung, als auch die Lokalisierung des Roboters messfehlerbehaftet sind, bedarf es spezieller Algorithmen, die beides gemeinsam approximieren, woraus sich die Problembezeichnung Simultaneous Localization and Mapping ergibt. Einige dieser Algorithmen und ihre mathematischen Hintergründe werden wir in dieser Arbeit genauer betrachten. Der Fokus wird dabei im Least Square Bereich auf dem Scan-Matching und im stochastischen Bereich auf den Partikelfiltern liegen. Außerdem werden wir ein Multilevel-Verfahren verwenden, um die Effizienz bei der Lösung des SLAM-Problems zu erhöhen.

Diese Arbeit besteht aus drei Teilen. Im ersten Teil wird das Simultaneous Localization and Mapping Problem und mögliche Herangehensweisen im Detail beschrieben. Zur Heranführung an eine vollständige Lösung des Problems werden wir uns mit einem Scan-Matching Algorithmus befassen, der auf dem Least Square Ansatz beruht. Im Zweiten Teil werden wir auf stochastische Methoden zur Lokalisierung und Kartierung eingehen. Dabei werden vor allem Partikelfilter und Multilevel-Ansätze von Bedeutung sein. Durch Kombination dieser beiden erhalten wir einen Algorithmus der das SLAM-Problem vollständig und effektiv löst. Schließlich befasst sich der dritten Teil mit zwei Datensätzen aus der echten Welt. Der Algorithmus wird an diesen getestet und analysiert. Außerdem wird auf die Eigenschaften der Implementierung eingegangen und eine Bewertung der Ergebnisse vorgenommen.

2 Einführung in SLAM

2.1 Motivation

Was braucht ein Roboter, um sich in einer unbekannten Umgebung zurecht zu finden? Zum einen die Fähigkeit seine Umgebung zu kartografieren, zum anderen gleichzeitig abzuschätzen, wo er sich befindet. Basierend darauf können dann Wege für den Roboter geplant und Hindernisse umfahren werden. Mit den enormen Verbesserungen bei der Verarbeitungsgeschwindigkeit von Computern und der Verfügbarkeit kostengünstiger Sensoren wie Kameras und Laserentfernungsmessern wird SLAM heute in immer mehr Bereichen für praktische Anwendungen eingesetzt.

Nehmen wir einen Staubsaugerroboter für Häuser. Ohne SLAM würde er sich nur zufällig in einem Raum bewegen und wäre möglicherweise nicht in der Lage, den gesamte Boden zu reinigen. Außerdem verbraucht dieser Ansatz mehr Energie, so dass die Batterie schneller leer wäre. Andererseits können Roboter mit SLAM Informationen wie die Anzahl der Radumdrehungen und Daten von Kameras und anderen bildgebenden Sensoren verwenden, um den Umfang der erforderlichen Bewegung zu bestimmen. Dies wird als Lokalisierung bezeichnet. Gleichzeitig kann der Roboter die Kamera und andere Sensoren nutzen, um eine Karte der Hindernisse in seiner Umgebung zu erstellen und so vermeiden, dass derselbe Bereich doppelt gereinigt wird. Das bezeichnet man als Mapping.

Logischer Weise hat dieses Thema in den letzten Jahren sehr viel Aufmerksamkeit bekommen, insbesondere wenn wir an die autonom fahrenden Autos denken, die Tesla, Google und co. auf die Straße schicken. Im Prinzip ist SLAM also die autonome Antwort auf die Frage: „Wo bin ich?“.

2.2 Aufgabenstellung aus praktischer Sicht

Bei der Implementierung von SLAM auf einer Fahrzeughardware stellen die Rechenkosten ein Problem dar. Die Berechnungen werden in der Regel auf kompakten und stromsparenden eingebetteten Mikroprozessoren durchgeführt, die nur über eine begrenzte Verarbeitungsleistung verfügen. Um eine genaue Lokalisierung zu erreichen, müssen Bildverarbeitung und Punktwolkenabgleich mit hoher Frequenz durchgeführt werden. Darüber hinaus sind Optimierungsberechnungen wie Loop Closures rechenintensive Prozesse. Die Herausforderung besteht darin, solche rechenintensiven Prozesse auf eingebetteten Mikrocomputern auszuführen.

Eine Gegenmaßnahme besteht darin, verschiedene Prozesse parallel laufen zu lassen. Prozesse wie die Merkmalsextraktion, die eine Vorverarbeitung des Matching-Prozesses darstellt, eignen sich relativ gut für eine Parallelisierung. Die Verwendung von Multicore-CPUs für die Verarbeitung, SIMD-Berechnungen (Single Instruction Multiple Data) und eingebettete GPUs

können die Geschwindigkeit in einigen Fällen weiter erhöhen. Da die Optimierung des Posegraphs in einem relativ langen Zyklus durchgeführt wird, kann eine Senkung der Priorität und die Durchführung dieses Prozesses in regelmäßigen Abständen die Leistung ebenfalls verbessern.

2.3 Aufgabenstellung aus mathematischer Sicht

Warum ist SLAM schwer? Im Grunde, weil die gleichen Sensordaten für beide benötigten Fähigkeiten benutzt werden müssen. Sowohl für das Mapping, als auch für die Lokalisierung. Beim Lösen dieses Problems stoßen wir auf zwei Große Quellen der Unsicherheit.

- Die stetige Unsicherheit in der Position des Roboters und der beobachteten Landschaftsmerkmale
- und die diskrete Unsicherheit in der Erkennung und Wiedererkennung der Landschaftsmerkmale. (Dieser Bereich heißt „data association“)

Jeder Lösungsansatz für das SLAM Problem, der beide Arten an Unsicherheit berücksichtigt, muss irgendwie den Raum der möglichen Landkarten durchsuchen, weil verschiedene Zuordnungen in der data association sehr unterschiedliche Landkarten erzeugen können.

Einer der wichtigsten Aspekte für eine präzise Lokalisierung und Kartierung ist die Beschreibung der Roboterbewegung und -beobachtung durch systemspezifische Modelle, die als Bewegungsmodell und Beobachtungsmodell bezeichnet werden. In diesem Abschnitt definieren wir zunächst die grundlegenden probabilistischen Definitionen und Notationen, bevor wir in Kapitel 4 weiter auf sie eingehen. Der probabilistische Ansatz verwendet verschiedene Positions x Definitionen, die die Wahrscheinlichkeit beschreiben, dass der Roboter, basierend auf gegebenen Werten, an einer bestimmten Position lokalisiert wird und das in Abhängigkeit von Parametern wie der Karte m, der Steuerung und den Beobachtungen.

$$p(x|m) \tag{2.1}$$

$$p(x_t|u_t, x_{t-1}) \tag{2.2}$$

$$p(x_t|z_t) \tag{2.3}$$

Die Beobachtungen, bzw. Laserscans z für jeden Zeitpunkt von 0 bis t werden durch folgenden Vektor definiert:

$$z_{0:t} = \{z_0\ z_1\ \ldots\ z_t\} \tag{2.4}$$

Die Steuerungen, bzw. Odometriemessungen u sind wie folgt gegeben:

$$u_{0:t} = \{u_0\ u_1\ \ldots\ u_t\} \tag{2.5}$$

Die zu bestimmende Verteilung ergibt sich insgesamt als

$$p(x_{0:t}, m \mid z_{1:t}, u_{1:t}). \tag{2.6}$$

Auf diesen Notationen werden wir im stochastischen Teil der Arbeit weiter aufbauen.

2.4 State of the Art

Nach fast 30 Jahren SLAM-Forschung gab es in der SLAM-Community einige Durchbrüche. Der Stand der Technik bei SLAM ist in drei Hauptforschungsrichtungen unterteilt, darunter Multi-Sensor-Fusion, Verwendung semantischer Informationen und Gewährleistung der Robustheit des SLAM-Systems in einer dynamischen Umgebung. Diese drei Richtungen überschneiden sich auch gegenseitig.

- Multi-Sensor-Fusion: Bei der Multi-Sensor-Fusion ist es unvermeidlich, die externen Parameter der Sensoren zu kalibrieren, und die Fusionsalgorithmen können grundsätzlich in zwei Kategorien eingeteilt werden: Filter und Graphoptimierung.

- Semantische Informationen: Es gibt zwei Hauptaspekte der Kombination von SLAM und Semantik. Zum einen hilft die Semantik SLAM. Informationen über Objekte können helfen, Karten mit Objektbeschriftungen zu erhalten, die für das menschliche Denken leichter zu verstehen sind. Darüber hinaus können die Tag-Informationen des Objekts mehr Einschränkungen für die Loop-Erkennung und BA-Optimierung mit sich bringen. Zum anderen hilft SLAM der Semantik. Mit SLAM können Ingenieure Daten über Objekte aus verschiedenen Perspektiven sammeln und ihre Position schätzen, automatisch hochwertige Beispieldaten für die semantische Erkennung erzeugen, die manuelle Kalibrierung von Daten vermeiden und den Trainingsprozess von Klassifikatoren beschleunigen.

- SLAM-System in dynamischer Umgebung: Das Hauptproblem beim dynamischen SLAM ist der Umgang mit der dynamischen Data association. Durch die Entscheidung, ob dynamische Übereinstimmungen entfernt oder zur Verfolgung von Objekten verwendet werden sollen, kann das dynamische SLAM-Problem als ein Robustheitsproblem oder als eine Erweiterung des Standard-SLAM betrachtet werden.

Die Fusion mehrerer Sensoren, die Optimierung der Data Association und der Loopback-Erkennung, die Integration mit einem heterogenen Front-End-Prozessor, die Verbesserung der Robustheit und die Genauigkeit der Neupositionierung sind die nächsten Entwicklungsrichtungen der SLAM-Technologie, die jedoch mit der Entwicklung der Verbraucheranreize und der industriellen

Kette schrittweise gelöst werden. Genau wie bei den Mobiltelefonen wird die SLAM-Technologie in naher Zukunft in das tägliche Leben der Menschen eindringen und ihr Leben verändern.

3 Heranführung an SLAM Algorithmen basierend auf Least Squares Ansätzen

In diesem Abschnitt werden wir uns mit zwei Least Square Algorithmen befassen. Zuerst mit dem direkten Scan-Matching und dann mit dem Extrahieren von Liniensegmenten, was als Erweiterung des Scan-Matching Algorithmus verwendet werden kann. Dies bietet aufgrund der hohen Anschaulichkeit und guten Verständlichkeit eine gute Heranführung an den Umgang mit Lidar-Sensordaten und die Lösung des SLAM-Problems und zeigt gleichzeitig eine Alternative zu den stochastischen Ansätzen aus Abschnitt 4 auf. Somit kann diese Bachelorarbeit einen guten Überblick über Methoden aus verschiedenen Bereichen der Herangehensweise an das SLAM-Problem vermitteln. Im Anschluss an die beiden Least Square Algorithmen wird der zugehörige mathematische Hintergrund genauer beleuchtet, um eine umfassende Behandlung des Least Square Ansatzes zu präsentieren.

3.1 Scan-Matching

Scan-Matching ist ein Verfahren, bei dem eine möglichst große Übereinstimmung von einem Scan mit einem Referenzmodell erzielt werden soll, indem der Scan entsprechend gedreht und verschoben wird. In der mobilen Robotik findet das Scan-Matching eine breite Anwendung, insbesondere beim Berechnen der Roboterposition und beim Erstellen sehr genauer Umgebungskarten. Da fast alle SLAM-Algorithmen, die mit Lidar-Sensoren arbeiten, eine Variante des Scan-Matchings verwenden, werden wir im folgenden eine davon näher betrachten.

Betrachtet man rohe Sensordaten eines Roboters allein basierend auf seinen Odometrie-Messungen, fällt einem schnell auf, dass sich bereits nach kurzer Zeit ein hoher Drift akkumuliert, weil Odometrie Daten von Robotern sehr verrauscht sind. Wäre dem nicht so, würden die vom LIDAR vermessenen Objekte statisch bleiben, während der Roboter an Ihnen vorbeifährt. Das ist jedoch in der Realität nicht der Fall und lässt sich auch an dem in dieser Arbeit betrachteten Intel Lab Datensatz feststellen. Interessant ist, dass der Drift bei Rotationen häufig größer ist, als der bei Translationen, was für das Verfahren in diesem Abschnitt von Vorteil ist.

Die Tatsache, dass vom Roboter passierte Objekte basierend auf seinen Odometrie-Messungen nicht statisch bleiben, lässt sich ausnutzen, um über die LIDAR-Scans des Roboters seine tatsächliche Bewegung abzuschätzen. Dazu werden aufeinanderfolgende Scans übereinandergelegt und so ver-

schoben, dass sich statische Objekte beider Scans nach Verschiebung an der gleichen Stelle befinden. Diese Transformation kann dann als Korrektur der Odometrie-Werte verwendet werden. Eine mögliche Methode dies umzusetzen, ist der Iterative-Closest-Point-Algorithmus (ICP), der den neuesten LIDAR-Scan mit dem vorherigen Scan oder mehreren vorangegangenen Scans abgleicht.

Der ICP-Algorithmus umfasst drei Schritte: Assoziierung, Transformation und Fehlerbewertung. Diese werden so lange wiederholt, bis die Scans zufriedenstellend aufeinander ausgerichtet sind. Das Ziel ist es also, die starre Transformation (Rotation und Translation) zu finden, die den neuen Scan am besten an den vorherigen anpasst.

Der erste Schritt bei der Schätzung dieser Transformation besteht darin, zu entscheiden, welche Punkte im neuen Scan denselben physischen Merkmalen entsprechen wie die Punkte im vorherigen Scan. Eine mögliche Herangehensweise ist die nächste Nachbarn Suche: Punkte im neuen Scan werden mit dem nächstgelegenen Punkt im vorherigen Scan verknüpft, falls dieser nicht bereits Teil einer Verknüpfung ist. Zwar entstehen dadurch vereinzelte Fehler bei der Assoziierung von Punkten aus den Scans, aber im Allgemeinen ziehen diese Assoziationen die Transformation in die richtige Richtung. Abbildung 1 veranschaulicht eine solche Assoziation.

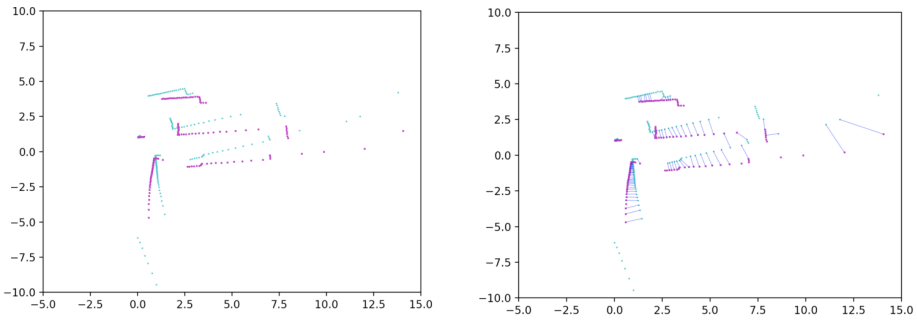

Abbildung 1: Visualisierung zweier aufeinanderfolgender Lidar-Scans mit eingezeichneter nächster Nachbarn Assoziierung im rechten Bild. Punkte der vorherigen Messung sind hellblau, Punkte der aktuellen Messung sind violett.

Der nächste Schritt in dem Algorithmus ist es, die Transformation zu bestimmen, die den mittleren quadratischen Abstand zwischen den assoziierten Punkten minimiert:

$$\tilde{T} = \underset{T}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \|v_i - T n_i\|^2$$

Wobei $\tilde{T}$ die endgültige geschätzte Transformation ist und v_i und n_i die

Punkte aus dem vorherigen bzw. neuen Scan sind, die miteinander assoziiert wurden.

Anschließend wird der Transformationsfehler e ausgewertet:

$$e = \frac{1}{N} \sum_{i=1}^{N} \|v_i - T n_i\|^2$$

Basierend darauf wird entschieden, ob eine weitere Iteration notwendig ist, um die Scans besser aneinander anzugleichen.

Bei der Implementierung bietet sich zur Suche der nächsten Nachbarn ein KD-Baum an, eine Datenstruktur aus der algorithmischen Geometrie. Dabei handelt es sich um einen Suchbaum für Punkte im zwei- und höherdimensionalen reellen Raum, der die zu durchsuchende Punktmenge mithilfe von Split-Hyperebenen rekursiv in zwei etwa gleichgroße Teilmengen aufteilt. In dem dadurch entstehenden binären Baum entspricht jeder interne Knoten des Baumes einer dieser Split-Hyperebenen und die Blätter entsprechen den zu durchsuchenden Punkten. Der Vorteil von KD-Bäumen liegt darin, dass ihre Erstellung nur $\mathcal{O}(n \log n)$ Zeit benötigt und die erwartete Laufzeit der Nächsten-Nachbarn-Suche im zweidimensionalen nach [FBF77] in $\mathcal{O}(\log n)$ liegt. Dies ist deutlich schneller als die Brute-Force Suche der nächsten Nachbarn aus den beiden Scans, die in $\mathcal{O}(n^2)$ liegen würde.

Bei der Bestimmung der Transformation $\tilde{T}$ geht es im Wesentlichen darum, die Kovarianz zwischen den beiden assoziierten Punktmengen, die Matrix M, zu ermitteln.

$$M = PQ^T$$

Dabei ist P eine Matrix, deren i-te-Spalte $n_i - \mu_n$ oder der i-te neue Punkt ist, ausgedrückt relativ zum Schwerpunkt der neuen Punktmenge μ_n. Analog dazu ist Q eine Matrix, deren i-te Spalte $v_i - \mu_v$ ist. Die Rotation R zwischen den beiden Punktwolken findet man nun mithilfe der Singulärwertzerlegung:

$$M = U \Sigma V^T$$
$$R = V U^T$$

Bleibt noch die Translation t zwischen den Punktmengen. Diese entspricht genau der Differenz zwischen den Schwerpunkten der Punktmengen:

$$t = \mu_v - \mu_n$$

Mit Rotation und Translation, lässt sich nun die optimale Transformation bezüglich der ersten Punkt-Assoziation bestimmen. Der zugehörige Ausrichtungsfehler berechnet sich als

$$e = \frac{1}{N} \sum_{i}^{N} \|v_i - R n_i + t\|^2$$

Dieser wird verwendet, um zu entscheiden, ob eine nächste Iteration notwendig ist, oder ob die Ausrichtung zwischen den beiden Punktmengen bereits gut genug ist. Für letzteres wird ein Schwellenwert herangezogen, der von der Qualität der Messdaten abhängt. Falls der Schwellenwert aufgrund von Mess-Ungenauigkeiten nicht erreicht werden kann, wird der ICP Algorithmus nach einer festen Anzahl von Iterationen beendet. Abbildung 2 veranschaulicht den Fortschritt des ICP Algorithmus nach einer Iteration zu den Messdaten aus Abbildung 1.

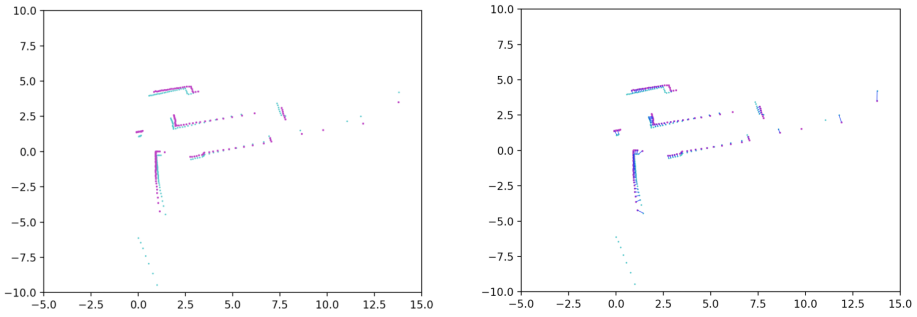

Abbildung 2: Visualisierung der Lidar-Scans aus Abbildung 1 nach der ersten Iteration des ICP Algorithmus mit eingezeichneter nächster Nachbarn Assoziierung im rechten Bild.

Die Genauigkeit des ICP Algorithmus hängt von dem Informationsgehalt der zu vergleichenden Scans ab. Wenn sich ein Roboter in der Nähe großer Wandabschnitte befindet, die nicht parallel zueinander sind, kann er seine Transformation zwischen den Scans sehr zuverlässig schätzen. Das liegt daran, dass er in allen Richtungen über gute Umgebungsinformationen zu seiner Bewegung verfügt. Befindet sich der Roboter hingegen in einem weitgehend geraden Gang mit parallelen Wänden, geben seine Messungen keinen Aufschluss darüber, wie er sich entlang des Ganges bewegt. Eine Ausrichtung ist so gut wie jede andere, solange die Wände übereinstimmen. In diesen Fällen sind die Bewegungsschätzungen des Roboters sehr schlecht. Ähnlich verhält es sich, wenn es nur wenige eindeutige, dauerhafte Merkmale im Scan gibt, was z.B. der Fall sein kann, wenn sich der Roboter Ecken nähert und sein Blickfeld damit verkleinert. Dann gibt es keine guten Anhaltspunkte für den Roboter, um seine Rotation abzuschätzen.

Der Abgleich aufeinanderfolgender LIDAR-Scans mit dem iterativen Algorithmus für nahegelegene Punkte, kann einem Roboter einige Informationen über seine eigene Bewegung liefern. Abbildung 3 verdeutlicht die Effektivität des ICP Algorithmus. Bereits nach wenigen Iterationen sind zwei aufeinanderfolgende Scans optimal aufeinander ausgerichtet.

Dies allein reicht jedoch nicht aus, um eine zuverlässige Bewegungsschätzung zu liefern. Es bedarf mindestens einem guten Startwert, der meist über

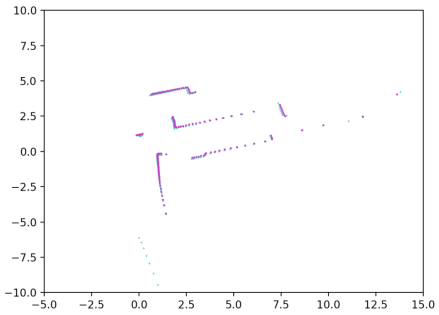

Abbildung 3: Visualisierung der Lidar-Scans aus Abbildung 1 nach fünf Iterationen des ICP Algorithmus. Die Ausrichtung der Scans ist bereits optimal.

die Odometrie-Sensoren geliefert wird, um nicht in unerwünschten lokalen Minima zu landen. Außerdem sei an dieser Stelle angemerkt, dass Rotationen mit dem ICP Algorithmus zwar recht gut geschätzt werden können, Translationen jedoch deutlich schwieriger sind. Da Odometrie-Sensoren jedoch meistens Translationen besser abschätzen als Rotationen, ergänzen sie den ICP Algorithmus gut. Eine Implementierung in Python des ICP Algorithmus findet sich auf der dieser Bachelorarbeit beigefügten CD.

3.2 Erweiterung auf Liniensegmente

Die Messungen eines Scans erfolgen für gewöhnlich in einer festen Reihe, z.B. nach aufsteigendem Winkel. Diese Eigenschaft wird im folgenden Verfahren ausgenutzt, um Liniensegmente aus den Messungen eines Scans zu extrahieren. Diese Liniensegmente können dann verwendet werden, um den vorangegangenen Scan-Matching Algorithmus zu erweitern, indem beim Schritt der Datenassoziierung nicht mehr einzelne Punkte miteinander verknüpft werden, sondern entweder Punkte aus einem Scan mit Liniensegmenten aus dem anderen Scan, oder Liniensegmente aus beiden Scans jeweils miteinander. Ein aufgenommener Scan wird also mit einem apriori Modell von Liniensegmenten überdeckt. Umgesetzt wurde dies z.B. von Cox [Cox90]. Der Ansatz berücksichtigt dann, dass gerade in Innenräumen die meisten gemessenen Punkte eigentlich Teil eines zusammenhängenden Objektes sind, beispielsweise einer Wand und kann dadurch kleinere Abweichungen zwischen zwei Scans nach einer angleichenden Transformation erreichen.

Ein weiterer Vorteil kann durch die Kartierung der Umgebung durch die Verwendung der Liniensegmente erreicht werden. Denn anstatt eine Karte mit allen gemessenen Punkten zu speichern, können die Liniensegmente zur Merkmalsextraktion dienen. Anstelle vieler einzelner Punkte müssen so lediglich ein paar Strecken gespeichert werden, was deutlich speichereffizienter ist. Sollte dies nicht genau genug für die zu kartierende Umgebung sein, können

zusätzlich weitere primitive geometrische Formen aus den Messungen, wie z.B. Kreise, extrahiert werden. So wächst die Genauigkeit einer Karte, die auf Merkmalsextraktion beruht. Schließlich bietet eine Liniensegmentkarte noch den Vorteil, aufgrund ihrer kompakten Struktur besser zur Pfandplang und Hindernisvermeidung geeignet zu sein, als eine Karte, die lediglich rohe Messpunkte abspeichert. Damit ist sie besser zur autonomen Navigation nutzbar.

Grundsätzlich funktioniert die Linienextrahierung so, dass eine feste Anzahl aufeinanderfolgender Messpunkte aus einem Scan ausgewählt wird und eine Regressionsgerade durch diese gelegt wird. Anschließend werden verschiedene Abstände zwischen den Messpunkten und ihrer Regressionsgeraden betrachtet, um die Qualität der Geraden abzuschätzen. Beschreibt die Gerade die aufeinanderfolgenden Punkte gut, wird der in Reihenfolge nächste Messpunkt zur Menge der ausgewählten Punkte hinzugenommen und es wird geprüft, ob auch diese Auswahl an Punkten noch gut durch eine Regressionsgerade beschrieben werden kann. Dies geht inkrementell so weiter, bis ein neu hinzugenommener Punkt dafür sorgt, dass keine gute Gerade mehr durch die ausgewählten Messpunkte gelegt werden kann. Dann wird die vorherige Auswahl als abgeschlossen betrachtet und ihre Regressionsgerade wird zur Menge der extrahierten Linien aufgenommen. Ab dem in Reihenfolge nächsten Messpunkt geht es nun wieder von vorne los und es wird versucht die nächste Linie zu extrahieren.

Betrachten wir nun die Implementierungsdetails zur Linienextraktion. Sei eine potentiell zu extrahierende Linie bestimmt durch $ax + by + c = 0$, mit $a, b, c \in \mathbb{R}$, sodass mindestens einer der beiden Koeffizienten a und b ungleich Null ist. Dann ist $\begin{pmatrix} a \\ b \end{pmatrix}$ der Normalenvektor der Linie. Die Distanz d von einem Punkt (x, y) aus den Messdaten zu dieser Linie ist

$$d = \frac{\left| \begin{pmatrix} a \\ b \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + c \right|}{\left\| \begin{pmatrix} a \\ b \end{pmatrix} \right\|_2}.$$

Entsprechend ist die Summe der Abstandsquadrate von n aufeinanderfolgenden Messpunkten zur Linie gegeben durch

$$f(a, b, c) = \sum_{i=1}^{n} d_i^2$$

wobei d_i der Abstand des i-ten Punktes zur Linie ist. Um $\underset{a,b,c}{\operatorname{argmin}} f(a, b, c)$, also die beste Orthogonale-Distanz-Regressiongerade zu finden, kann man entweder die partiellen Ableitungen von $f(a, b, c)$ nach a, b, c Null setzen, oder man bildet eine Matrix A aus den n Punkten durch die eine Gerade

gelegt werden soll, subtrahiert den Spaltenweisen Mittelwert und bildet die Singulärwertzerlegung. Dann erhalten wir eine Matrix $A = USV^T$ mit

$$V^T = \begin{pmatrix} b & a \\ -a & b \end{pmatrix}, \quad c = \begin{pmatrix} -a \\ b \end{pmatrix} \cdot \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$$

Der Abstand von jedem der n Punkte im sogenannten Seed-Segment (der aktuell zur Regression betrachteten Punkte) zur Regressionsgeraden sollte weniger als ein bestimmter Schwellenwert ε sein, der auf die Reichweite der Genauigkeit des Lasers abgestimmt ist. Also

$$\frac{\left| \begin{pmatrix} a \\ b \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \end{pmatrix} + c \right|}{\left\| \begin{pmatrix} a \\ b \end{pmatrix} \right\|_2} < \varepsilon$$

Außerdem sollte der Abstand von jedem Punkt im Seed-Segment zu seiner gepeilten Position, also der Position auf der Regressionsgeraden, wenn man den Punkt von der Roboterposition aus auf die Regressionsgerade projeziert, ebenfalls kleiner sein als ein vorgegebener empirischer Schwellenwert δ, wobei der Schwellenwert wie folgt festgelegt wird

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_i' \\ y_i' \end{pmatrix} \right\|_2 < \delta$$

Dabei gilt für den gepeilten Punkt (x_i', y_i'), dass er von der Roboterposition über den gemessenen Laserpunkt auf die Gerade gepeilt wird, also

$$\begin{cases} ax_i' + by_i' + c = 0 \\ \begin{pmatrix} -\sin(\theta) \\ \cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} x_i' \\ y_i' \end{pmatrix} = 0 \end{cases}$$

wobei θ dem aktuellen Winkel der Roboterposition entspricht. Es gilt also

$$\begin{pmatrix} x_i' \\ y_i' \end{pmatrix} = \frac{\begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}}{\begin{pmatrix} a \\ b \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}} \cdot (-c)$$

Abbildung 4 zeigt eine Regressionsgerade, durch sechs aufeinanderfolgende Laserpunkte, die durch einen guten Schwellenwert δ verhindert werden sollte.

Zuguterletzt sollte noch ein Schwellenwert κ verwendet werden, sodass benachbarte Laserpunkte, die Teil der gleichen Regressionsgerade sein sollen, nicht zu weit voneinander entfernt liegen.

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} \right\|_2 < \kappa$$

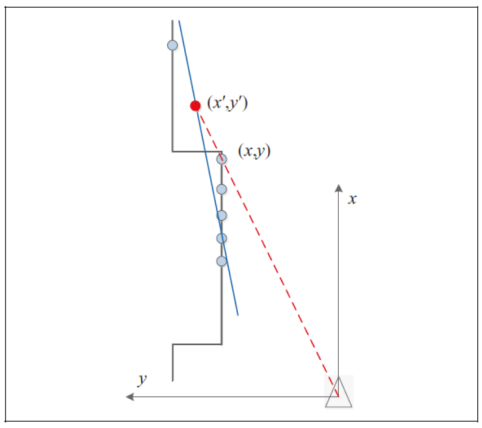

Abbildung 4: Die gepeilte Position des obersten Laserpunkts ist in rot eingezeichnet, während Regressionsgerade, die extrahiert werden soll in blau verzeichnet ist. Die rot-gestrichelte Linie entspricht der Peilung des obersten Laserpunkts. Ein passender Schwellenwet δ sollte die Extraktion dieser schrägen Geraden verhindern.

Die drei Schwellenwerte ε, δ und κ werden dabei in Abhängigkeit der Beschaffenheit der Messdaten gewählt. Im wesentlichen handelt es sich um Erfahrungswerte, die an den jeweiligen Roboter, bzw. dessen Sensoren angepasst werden. Alle drei Schwellenwerte zusammen sorgen dafür, dass nur Liniensegmente extrahiert werden, die die zugehörigen Messpunkte gut genug beschreiben. Eine Implementierung in Python zur Lininenextrahierung findet sich auf der dieser Bachelorarbeit beigefügten CD.

3.3 Mathematischer Hintergrund

In diesem Abschnitt beschäftigen wir uns mit dem mathematischen Hintergrund der beiden vorangegangenen Least Square Algorithmen. Sowohl beim einfachen Scan-Matching, als auch beim extrahieren der Liniensegmente wurde ein Optimierungsproblem gelöst. Einmal die Optimierung einer Transformation zwischen zwei Scans und einmal die Optimierung einer Orthogonalen-Distanz-Regressionsgeraden. Beide Male konnte diese Optimierung mithilfe der Singulärwertzerlegung gelöst werden. Die Mathematik dahinter werden wir nun genauer beleuchten.

Ein Optimierungsproblem wird im Allgemeinen als Suche nach einem Minimum einer Zielfunktion oder Kostenfunktion $F(x)$ definiert. Man interessiert sich nicht so sehr für den Funktionswert von $F(x)$ am Minimum, sondern vielmehr für den Wert der Variablen x, bei dem dieses Minimum auftritt. Formal kann man ein Optimierungsproblem wie folgt beschreiben

$$x^* = \underset{x}{\operatorname{argmin}}\, F(x) \tag{3.1}$$

Aus der grundlegenden mathematischen Analyse wissen wir, dass ein Extrempunkt (d. h. ein Minimum oder Maximum) einer Funktion F auftritt, wenn ihre erste Ableitung F' gleich Null ist. Tatsächlich spielen die Ableitungen der Kostenfunktion, wie wir noch sehen werden, eine wichtige Rolle wenn es darum geht, das obige Problem zu lösen. Im Allgemeinen ist die Zielfunktion eine Funktion $F : \mathbb{R}^n \to \mathbb{R}$, wobei $n \geq 1$. ist.

3.3.1 Kleinste Quadrate Optimierungsprobleme

Der Begriff der Kleinsten-Quadrate-Optimierung beschreibt eine sehr wichtige Unterklasse von allgemeinen Optimierungsproblemen. Bei der Optimierung der kleinsten Quadrate ist die Zielfunktion eine Summe über quadratische Terme, d.h.

$$F(x) = \frac{1}{2} \sum_{i}^{n} f_i(x)^2 \tag{3.2}$$

und die Funktionen f_i sind skalarwertige Funktionen $f_i : \mathbb{R}^n \to \mathbb{R}$. Diese Formulierung der Zielfunktion ist sehr gebräuchlich und viele SLAM Teilprobleme können als (nichtlineare) Probleme der kleinsten Quadrate formuliert werden. Gleichung (3.2) lässt sich mit Hilfe der Vektorschreibweise zu einer kompakteren Form umformulieren:

$$F(x) = \frac{1}{2} f(x)^T f(x) \tag{3.3}$$

Falls $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))^T$ ist. Das kleinste Quadrate Optimierungsproblem ist dann

$$x^* = \underset{x}{\operatorname{argmin}}\, F(x) = \underset{x}{\operatorname{argmin}} \frac{1}{2} f(x)^T f(x) \tag{3.4}$$

3.3.2 Lineare kleinste Quadrate Probleme

Im Falle eines linearen Problems der kleinsten Quadrate hängen alle f_i linear von x ab, d.h. der Funktionswert ist eine lineare Kombination der Variablen in x. Dann können wir $f_i(x)$ schreiben als:

$$f_i(x) = a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,n}x_n - b_i \tag{3.5}$$

Angesichts dessen können wir natürlich die einzelnen f_i übereinander stapeln, um den Spaltenvektor $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))^T$ zu erhalten. Dann wird durch Zusammenfassen aller Koeffizienten $a_{i,j}$ in der Matrix A und aller b_i im Vektor b der folgende einfache Ausdruck erreicht:

$$f(x) = Ax - b \tag{3.6}$$

3.3.3 Lösen linearer kleinste Quadrate Probleme

Um die Lage der Extrempunkte von $F(x)$ zu bestimmen, benötigen wir die Ableitung $F'(x)$:

$$F'(x) = \frac{\partial F}{\partial x} \tag{3.7}$$

$$= \frac{\partial \left(\frac{1}{2} f^T f\right)}{\partial x} \tag{3.8}$$

$$= f^T \frac{\partial f}{\partial x} \tag{3.9}$$

Die letzte Zeile wurde unter Anwendung der Kettenregel abgeleitet. Nun, da

$$\frac{\partial f}{\partial x} = \frac{\partial (Ax - b)}{\partial x} = A \tag{3.10}$$

erhalten wir schließlich

$$F'(x) = (Ax - b)^T \cdot A \tag{3.11}$$

Die Position des Extremas x^* wird ermittelt, indem die erste Ableitung auf Null gesetzt wird:

$$0 = F'(x^*) \tag{3.12}$$

$$= (Ax - b)^T \cdot A \tag{3.13}$$

$$= A^T (Ax - b) \tag{3.14}$$

$$= A^T A x^* - A^T b \tag{3.15}$$

$$A^T A x^* = A^T b \tag{3.16}$$

Die Gleichung in der letzten Zeile heißt Normalengleichung. Diese kann mit verschiedenen Methoden gelöst werden:

1. Falls $A^T A$ invertierbar ist, kann die Lösung direkt durch $x^* = (A^T A)^{-1} A^T b$ gefunden werden.

2. Eine bessere Alternative ist die QR-Zerlegung von A in eine obere Dreiecksmatrix Matrix R und eine orthogonale Matrix Q, so dass A = QR ist. Die Lösung x^* ist dann gegeben durch $x^* = R^{-1} Q^T b$. Diese Methode verhindert die Berechnung von $A^T A$ und seine Invertierung, die numerisch instabil werden kann.

3. Anstelle der QR-Zerlegung kann auch die Cholesky-Zerlegung von $A^T A$ verwendet werden. Bei dieser Methode wird $A^T A$ in LL^T zerlegt, dann wird $Ly = A^T b$ für y gelöst und x^* durch Lösen von $L^T x = y$ gefunden. L ist eine untere Dreiecksmatrix mit positiven Diagonalelementen. Verglichen mit der QR-Zerlegung ist die Cholesky-Zerlegung schneller, aber numerisch instabiler, da $A^T A$ berechnet werden muss.

4. Die Lösung des linearen Problems der kleinsten Quadrate mit $f(x) = Ax - b$ kann auch mit Hilfe der Singulärwertzerlegung (SVD) bestimmt werden. Hier würde A in $A = USV^T$ zerlegt. Nach der Berechnung eines neuen Vektors $b' = U^T b$ und Definition eines Vektors y mit $y'_i = b'/s_i$, wobei s_i der i-te Diagonaleintrag von S ist, kann die Lösung x^* durch $x^* = Vy$ gefunden werden.

Genau die letzte Methode, also die Singulärwertzerlegung, ist es schließlich, die uns beim Lösen der Least Square Probleme aus den ersten beiden Abschnitten dieses Kapitels hilft. Da sowohl das Extrahieren von Liniensegmenten, als auch das Optimieren einer Transformation zwischen zwei Scans unter die linearen kleinste Quadrate Probleme fällt, sind die Voraussetzungen zur Anwendbarkeit der Singulärwertzerlegung erfüllt.

4 Multilevel Resolution Algorithm

Ein vollständiger Lösungsansatz für SLAM setzt sich immer aus mehreren Verfahren zusammen. So gibt es verschiedene Möglichkeiten zur Lokalisierung, zur Kartierung und zur Kombination dieser beiden Bereiche. Man muss also eine Auswahl zwischen Bayes-Filtern und Least-Square Ansätzen wählen, zwischen Positionsgraphen und Belegungsrastern und zwischen online und offline Algorithmen. Dabei sind letztere rechenaufwendigere Verfahren, die erst im Nachhinein auf einen Datensatz angewendet werden, wohingegen online-Algorithmen auf den Echtzeitgebrauch ausgelegt sind. Einen Prozess haben jedoch alle LIDAR-basierten Verfahren gemeinsam und das ist der Gebrauch von Scan-Matching Algorithmen, wobei aufeinanderfolgende Aufnahmen miteinander abgeglichen werden, um die kurzzeitige Bewegung des Roboters abzuschätzen. Da LIDAR-Messungen genauere Informationen bezüglich Richtung und Entfernung von Hindernissen und Objekten mit sich bringen, als bspw. Kameras, sind Scan-Matching Algorithmen für LIDAR-Daten besonders robust.

In diesem Kapitel geht es um einen Algorithmus, der das SLAM-Problem mithilfe stochastischer Methoden löst. Nachdem im letzten Kapitel bereits die Grundlagen für den Umgang mit dem SLAM-Problem gelegt worden sind, werden nun anspruchsvollere Methoden verwendet, um eine bessere Lösung und qualitativ hochwertige Karten zu erzielen. Neben den Partikelfiltern, die nun aus dem stochastischen Bereich zum Einsatz kommen, ist das Konzept der Multilevel-Optimierung ein entscheidender Baustein des Algorithmus. Die Idee das-SLAM Problem in dieser Bachelorarbeit auf mehreren Ebenen zu optimieren stammt aus dem Paper des Multilevel Relaxation Algorithms von Frese, Larsson und Duckett [FLD05]. Allerdings unterscheidet sich die hier erläuterte Umsetzung maßgeblich von der Umsetzung des Multilevel Relaxation Algorithms, da letzterer mit Least Square Optimierung arbeitet und Positionsgraphen zur Kartierung verwendet, während hier im Folgenden sto-

chastische Optimierung mit Belegungsrastern zur Kartierung genutzt wird. Als Variante der stochastischen Optimierung mit Partikelfilter, werden wir zusätzlich Rao-Blackwell'sche Partikelfilter betrachten.

4.1 Localization

4.1.1 Partikelfilter

Partikelfilter gehören zur Familie der Markov-Lokalisierungsverfahren, die auf diskreten Modellen basieren und sind zusammen mit den Gauß-Filtern die am häufigsten verwendeten Bayes'schen Filter-Algorithmen. Der posteriore Wahrscheinlichkeitswert $Bel(x_t)$ wird hier durch eine endliche Anzahl von Parametern approximiert und der Partikelfilter konstruiert $Bel(x_t)$ rekursiv aus seinem vorherigen Zustand $Bel(x_{t-1})$. Die Bezeichnung Partikel deutet darauf hin, dass $Bel(x_t)$ durch die „Partikel" repräsentiert wird - eine Menge von M Stichproben für jeden Positionszustand $x^{(i)}$ mit dem Gewichtungsfaktor $w^{(i)}$:

$$Bel(x_t) = \{x_t^{(i)}, w_i^{(i)}\}_{i=1,\ldots,M} \tag{4.1}$$

Der Wichtigkeits-, bzw. Gewichtungsfaktor ist das Gewicht jeder Probe, das zur Auswahl der besten Partikel im Prozess verwendet wird. Im Anfangszustand, wenn die Position des Roboters ungenau ist und als gleichmäßige Verteilung dargestellt wird, wird das Gewicht aller Stichproben als der gleichmäßige Gewichtungsfaktor $\frac{1}{M}$ beschrieben.

Es gibt verschiedene Techniken der Partikelfilter-Implementierung, wir werden uns jedoch auf die sogenannte adaptive Technik konzentrieren. Dies bedeutet, dass der Filter die Anzahl der Parameter anpassen kann, um den posteriore Wahrscheinlichkeitswert online darzustellen. Einer der wichtigsten Partikelfilter-Algorithmen ist die Adaptive Monte Carlo Lokalisierung (AMCL).

Ein mögliches Beispiel für die Implementierung eines Partikelfilters ist die Lösung von

$$\underbrace{Bel(x_t)}_{neue\ Verteilung} = \alpha \cdot \underbrace{p(z_t|x_t)}_{\substack{Messdaten- \\ wahrscheinlichkeit}} \cdot \overbrace{\int \underbrace{p(x_t|x_{t-1}, u_{t-1})}_{Positionsveraenderung} \underbrace{Bel(x_{t-1})}_{vorheriges}\, dx_{t-1}}^{Vorhersage}, \tag{4.2}$$

die auch von Thrun et al. [Ta01] und Fox et al. [Fa01] verwendet wird. Die Berechnung ist in drei Phasen unterteilt: Vorhersage, Vorschlag und Korrektur von der rechten zur linken Seite der Gleichung.

- **Im Vorhersageschritt**
  wird eine Position x_{t-1} aus $Bel(x_{t-1})$ durch $x_{t-1}^{(i)}$ aus der Stichprobenmenge, die $Bel(x_{t-1})$ repräsentiert, mit dem Gewichtungsfaktor $w_{t-1}^{(i)}$ gesampelt.

- **Im Vorschlagsschritt**
  wird ein Zustand $x_t^{(i)}$ unter Verwendung der Stichprobe $x_{t-1}^{(i)}$ und der Odometriemessungen, bzw. Steuerungen u_{t-1} aus $p(x_t|x_{t-1}, u_{t-1})$ ausgewählt. Die Verteilung von $x_t^{(i)}$ und $x_{t-1}^{(i)}$ ist durch das Produkt $p(x_t|x_{t-1}, u_{t-1}) \cdot Bel(x_{t-1})$ gegeben und wird als Vorschlagsverteilung bezeichnet.

- **Im Korrekturschritt**
  wird eine Stichprobe $x_t^{(i)}$ mit einem Gewichtungsfaktor bei gegebener Beobachtung z_t gewichtet. Für jede i-te Stichprobe wird die Zielverteilung aus Gleichung (4.2) wie folgt berechnet:

$$\alpha \cdot p(z_t|x_t^{(i)}) \cdot p(x_t^{(i)}|x_{t-1}^{(i)}, u_{t-1}) \cdot Bel(x_{t-1}^{(i)}) \tag{4.3}$$

  Der Gewichtungsfaktor für jedes Sample ist definiert als:

$$w^{(i)} = p(z_t|x_t^{(i)}) \tag{4.4}$$

  Und gegeben durch die Gleichung des Quotienten:

$$\frac{\overbrace{\alpha \cdot p(z_t|x_t^{(i)}) \cdot p(x_t^{(i)}|x_{t-1}^{(i)}, u_{t-1}) \cdot Bel(x_{t-1}^{(i)})}^{Zielverteilung}}{\underbrace{p(x_t^{(i)}|x_{t-1}^{(i)}, u_{t-1}) \cdot Bel(x_{t-1}^{(i)})}_{Vorschlagsverteilung}} = \alpha \cdot p(z_t|x_t^{(i)}) \tag{4.5}$$

  Da α eine normalisierende Konstante ist, die proportional zu $w^{(i)}$ ist.

Die Approximation von $Bel(x_t)$ durch gewichtetes Sampling wird mit M Wiederholungen berechnet, bis der Gewichtungsfaktor normalisiert ist. $\left(\sum_{i=1}^{N} w^{(i)} = 1\right)$. Die erzeugten $x_t^{(i)}$ Samples definieren die diskrete Wahrscheinlichkeitsverteilung der Gleichung (4.2).

4.1.2 Rao-Blackwell'scher Partikelfilter

Nach Murphy [Mur99] besteht die Kernidee des Rao-Blackwell'schen Partikelfilters für SLAM darin, die kombinierte Verteilung $p(x_{1:t}, m|z_{1:t}, u_{1:t-1})$ über die Karte m und die Trajektorie $x_{1:t} = x_1, \ldots, x_t$ des Roboters zu schätzen. Diese Schätzung erfolgt anhand der Lidar-Messungen $z_{1:t} = z_1, \ldots, z_t$ und den Odometriemessungen $u_{1:t-1} = u_1, \ldots, u_{t-1}$ des mobilen Roboters. Der Rao-Blackwell'sche Partikelfilter für SLAM verwendet die folgende Faktorisierung

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}) \cdot p(x_{1:t}|z_{1:t}, u_{1:t-1}). \tag{4.6}$$

Diese Faktorisierung ermöglicht es uns, zunächst nur die Trajektorie des Roboters zu schätzen und dann die Karte anhand dieser Trajektorie zu berechnen. Da die Karte stark von der Positionsschätzung des Roboters abhängt, bietet dieser Ansatz eine effiziente Berechnungsmethode.

In der Regel kann Gleichung (4.6) effizient berechnet werden, da die Verteilung über die Karten $p(m|x_{1:t}, z_{1:t})$ analytisch unter Verwendung von „Mapping mit bekannten Posen“ [Mor88] berechnet werden kann, da $x_{1:t}$ und $z_{1:t}$ bekannt sind. Um die Verteilung $p(x_{1:t}|z_{1:t}, u_{1:t-1})$ über die potentiellen Trajektorien zu schätzen, kann man einen Partikelfilter anwenden. Jeder Partikel repräsentiert eine potentielle Trajektorie des Roboters. Außerdem ist eine individuelle Karte mit jedem Partikel verbunden. Die Karten werden aus den Laserscans und den Trajektorien des entsprechenden Partikels erstellt.

Einer der gebräuchlichsten Algorithmen zur Partikelfilterung ist der Sampling-Importance-Resampling-Filter (SIR). Der Rao-Blackwell'scher SIR-Filter für das Mapping verarbeitet die Sensormessungen und die Odometriedaten inkrementell, sobald sie verfügbar sind. Er aktualisiert die Menge der Partikel, die die Verteilung über die Karte und die Trajektorie des Roboters darstellt. Der Prozess lässt sich in den folgenden vier Schritten zusammenfassen:

1. *Sampling*: Die nächste Generation von Partikeln $\{x_t^{(i)}\}$ wird aus der Generation $\{x_{t-1}^{(i)}\}$ durch sampeln aus der Vorschlagsverteilung π gewonnen. Häufig wird ein probabilistisches Odometrie Bewegungsmodell als Vorschlagsverteilung verwendet.

2. *Gewichtungsfaktoren*: Jedem Partikel wird je nach Wichtigkeit ein individueller Gewichtungsfaktor $w_t^{(i)}$ zugewiesen, das dem Wichtigkeits-Sampling-Prinzip entspricht.

$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})} \tag{4.7}$$

   Die Gewichte tragen der Tatsache Rechnung, dass die Vorschlagsverteilung π im Allgemeinen nicht gleich der Zielverteilung der Nachfolgezustände ist.

3. *Resampling*: Die Partikel werden proportional zu ihrer Wichtigkeit ersetzt. Dieser Schritt ist notwendig, da nur eine endliche Anzahl von Partikeln verwendet wird, um eine kontinuierliche Verteilung zu approximieren. Darüber hinaus ermöglicht das Resampling die Anwendung eines Partikelfilters in Situationen in denen die Zielverteilung von der Vorschlagsverteilung abweicht. Nach dem Resampling haben alle Partikel das gleiche Gewicht.

4. *Kartenschätzung*: Für jeden Partikel wird die entsprechende Kartenschätzung $p(m^{(i)}|x_{1:t}^{(i)}, z_{1:t})$ berechnet auf der Grundlage der Trajektorie $x_{1:t}^{(i)}$ des Partikels und der Reihe an Laserscans $z_{1:t}$.

Nach Doucet et al. [DFG01], erhalten wir eine rekursive Formulierung zur Berechnung der Gewichtungsfaktoren durch Einschränkung der Vorschlagsverteilung π auf die Erfüllung der folgenden Annahme:

$$\pi(x_{1:t}|z_{1:t}, u_{1:t-1}) = \pi(x_t|x_{1:t-1}, z_{1:t}, u_{1:t-1}) \cdot \pi(x_{1:t-1}|z_{1:t-1}, u_{1:t-2}) \quad (4.8)$$

Berechnet man die Gewichtungsfaktoren dann entsprechend dem Beobachtungsmodell $p(z_t|m, x_t)$ und ersetzt man die Vorschlagsverteilung π durch das Bewegungsmodell $p(x_t|x_{t-1}, u_{t-1})$ erhält man:

$$\begin{aligned} w_t^{(i)} &= w_{t-1}^{(i)} \cdot p(z_t|x_{t-1}^{(i)}, m^{(i)}, u_t) && (4.9)\\ &= w_{t-1}^{(i)} \cdot \int_{x_t} \underbrace{p(z_t|x_t, m^{(i)}) \cdot p(x_t|x_{t-1}^{(i)}, u_t)}_{\tau(x_t)} \, dx_t && (4.10)\\ &\simeq w_{t-1}^{(i)} \cdot \int_{\{x_t|\tau(x_t)>\varepsilon\}} \tau(x_t) \, dx_t && (4.11)\\ &\simeq w_{t-1}^{(i)} \cdot \sum_{j=1}^{K} \tau(x_j) && (4.12) \end{aligned}$$

Dabei sind x_j die gesampelten Partikel um das Maximum der vom Scan-Matching gefundenen Wahrscheinlichkeitsverteilung.

4.1.3 Algorithmus

Jedes mal wenn ein neues Messtupel (u_{t-1}, z_t) verfügbar ist, wird der Vorschlag für jeden Partikel einzeln berechnet und dann verwendet, um diesen Partikel zu aktualisieren. Dies führt zu den folgenden Schritten:

1. Eine erste Schätzung $x_t^{'(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$ für die Position des Roboters, die durch das Partikel i repräsentiert wird, wird aus der vorherigen Position $x_{t-1}^{(i)}$ dieses Partikels und den seit der letzten Filteraktualisierung gesammelten Odometriemessungen u_{t-1} gewonnen. Der Operator $\oplus$ entspricht dabei dem Standard-Positionskompositionsoperator [LM97].

2. Ein Scan-Matching-Algorithmus wird auf der Grundlage der Karte $m_{t-1}^{(i)}$ ausgeführt, ausgehend von der Anfangsschätzung $x_t^{'(i)}$. Die vom Scan-Matcher durchgeführte Suche ist auf einen begrenzten Bereich um $x_t^{'(i)}$ beschränkt. Wenn das Scan-Matching einen Fehlschlag meldet, werden die Position und die Gewichte gemäß dem Bewegungsmodell berechnet (und die Schritte 3 und 4 werden übersprungen).

3. In einem Intervall um die vom Scan-Matcher gemeldete Position $\hat{x}_t^{(i)}$ wird ein Menge von Stichprobenpunkten ausgewählt. Auf der Grundlage dieser Punkte werden der Mittelwert und die Kovarianzmatrix des

Vorschlags durch punktweise Auswertung der Zielverteilung $p(z_t|m_{t-1}^{(i)}, x_j)p(x_j|x_{t-1}^{(i)}, u_{t-1})$ an den gesampelten Positionen x_j berechnet. In dieser Phase wird auch der Gewichtungsfaktor $\eta^{(i)}$ gemäß folgender Gleichung berechnet:

$$\eta^{(i)} = \sum_{j=1}^{K} p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1}) \tag{4.13}$$

4. Die neue Position $x_t^{(i)}$ des Partikels i wird aus der Gaußschen Approximation $\mathcal{N}(\mu_t^{(i)}, \Sigma_t^{(i)})$ der verbesserten Vorschlagsverteilung gezogen.

5. Aktualisierung der Gewichtungsfaktoren.

6. Die Karte $m^{(i)}$ des Partikels i wird entsprechend der gezogenen Position $x_t^{(i)}$ und der Beobachtung z_t aktualisiert.

Nach der Berechnung der nächsten Generation von Partikeln wird ein Resampling Schritt in Abhängigkeit vom Wert von $N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N}(\tilde{w}^{(i)})^2}$ durchgeführt, wobei sich $\tilde{w}^{(i)}$ auf das normalisierte Gewicht des Partikels i bezieht.

Der Pseudocode zum vorgestellten Algorithmus findet sich in Algorithmus 1. Die zugehörige Implementierung in Python ist auf der zu dieser Bachelorarbeit beigefügten CD enthalten.

Algorithmus 1 Rao-Blackwell'scher Partikelfilter

Input:

$\mathcal{S}_{\sqcup -\infty}$, die Sample Menge des vorangegangenen Zeitschritts
z_t, die aktuellen Laserscan-Daten
u_{t-1}, die neusten Odometrie-Messungen

Output:

$\mathcal{S}_t$, die neue Sample Menge

1: $\mathcal{S}_t = \{\}$
2: **for all** $s_{t-1}^{(i)} \in \mathcal{S}_{t-1}$ **do**
3: $\quad \langle x_{t_1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)} \rangle = s_{t-1}^{(i)}$

$\quad$ // Scan-Matching
4: $\quad x'^{(i)}_t = x_{t-1}^{(i)} \oplus u_{t-1}$
5: $\quad \hat{x}_t^{(i)} = \text{argmax}_x p(x \mid m_{t-1}^{(i)}, z_t, x'^{(i)}_t)$

6: **if** $\hat{x}_t^{(i)} = \text{failure}$ **then**
7: $x_t^{(i)} \sim p(x_t \mid x_{t-1}^{(i)}, u_{t-1})$
8: $w_t^{(i)} = w_{t-1}^{(i)} \cdot p(z_t \mid m_{t-1}^{(i)}, x_t^{(i)})$
9: **else**
// Sample um den Bereich
10: **for** $k = 1, \ldots, K$ **do**
11: $x_k \sim \{x_j \mid |x_j - \hat{x}^{(i)}| < \Delta\}$
12: **end for**

// Berechnung des Gaußschen Vorschlags
13: $\mu_t^{(i)} = (0, 0, 0)^T$
14: $\eta^{(i)} = 0$
15: **for all** $x_j \in \{x_1, \ldots, x_K\}$ **do**
16: $\mu_t^{(i)} = \mu_t^{(i)} + x_j \cdot p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_t \mid x_{t-1}^{(i)}, u_{t-1})$
17: $\eta^{(i)} = \eta^{(i)} + p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_t \mid x_{t-1}^{(i)}, u_{t-1})$
18: **end for**
19: $\mu_t^{(i)} = \mu_t^{(i)} / \eta^{(i)}$
20: $\Sigma_t^{(i)} = 0$
21: **for all** $x_j \in \{x_1, \ldots, x_K\}$ **do**
22: $\Sigma_t^{(i)} = \Sigma_t^{(i)} + (x_j - \mu^{(i)})(x_j - \mu^{(i)})^T \cdot p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_j \mid x_{t-1}^{(i)}, u_{t-1})$
23: **end for**
24: $\Sigma_t^{(i)} = \Sigma_t^{(i)} / \eta^{(i)}$
// ziehe neue Positionen aus der Verteilung
25: $x_t^{(i)} \sim \mathcal{N}(\mu_t^{(i)}, \Sigma_t^{(i)})$

// update die Gewichtungsfaktoren
26: $w_t^{(i)} = w_{t-1}^{(i)} \cdot \eta^{(i)}$
27: **end if**
// update die Karte
28: $m_t^{(i)} = \text{integrateScan}(m_{t-1}^{(i)}, x_t^{(i)}, z_t)$
// update die Sample Menge
29: $\mathcal{S}_t = \mathcal{S}_t \cup \{\langle x_t^{(i)}, w_t^{(i)}, m_t^{(i)} \rangle\}$
30: **end for**

31: $N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (\tilde{w}^{(i)})^2}$
32: **if** $N_{\text{eff}} < T$ **then**
33: $\mathcal{S}_t = \text{resample}(\mathcal{S}_t)$
34: **end if**

4.2 Mapping

Es gibt verschiedene Möglichkeiten, eine Karte der Umgebung zu erstellen. Die zwei verbreitetsten Varianten sind Pose Graphs (Positionsgraphen) und Occupancy Grids (Belegungsraster). Beide bringen ihre Vor- und Nachteile mit sich und werden je nach Anforderungen des zu vermessenden Gebiets ausgewählt. So ermöglichen Belegungsraster bspw. eine direkte Anwendung effizienter Pfadplanungsalgorithmen, die in vielen Robotern zum Einsatz kommen. Da wir uns in dieser Arbeit mit Indoor-Datensätzen beschäftigen, bieten sich letztere, also die Belegungungsraster, an. Diese vereinfachen vorallem den Schritt der Data-Association, da sie eine effektive und robuste Suche nach dem globalen Maximum beim Integrieren neuer Lidar-Scans ermöglichen. Im Folgenden werden wir erst eine allgemeine Form der Belegungsraster betrachten und dann auf die Anpassungen eingehen, die unseren Algorithmus durch einen Multilevel-Ansatz beschleunigen.

4.2.1 Allgemeine Belegungsraster

Belegungsraster teilen den zu vermessenden Raum durch ein feinmaschiges Gitter auf. Jede Zelle des Gitters wird durch eine binäre Zufallsvariable dargestellt, die die Wahrscheinlichkeit der Belegung durch ein Hindernis beschreibt - 0, wenn sie frei ist und 1, wenn sie definitiv besetzt ist. Die bislang unbeobachteten Zellen werden mit einem Standardwert belegt, zum Beispiel $0,5$. Dann ist die Karte durch ein Produkt der einzelnen Zellen gegeben:

$$p(m) = \prod_{i}^{N} p(m_{x_i,y_i}) \tag{4.14}$$

Die Berechnung der Karte des Belegungsrasters ist in der Regel ein probabilistisches Problem, bei dem der posteriore Wahrscheinlichkeitswert aus den gegebenen Beobachtungen berechnet wird, wobei die Positionsschätzung auf dem statischen Zustand des binären Bayes-Filters beruht. Daher ist die Karte gemäß der Bayes-Regel und der Markov-Vermutung gegeben als:

$$p(m_i|Z_{1:t}, X_{1:t}) = \frac{p(Z_t|m_i, Z_{t-1}, X_{1:t}) \cdot p(m_i|Z_{1:t-1}, X_{1:t})}{p(Z_t|Z_{t-1}, X_{1:t})} \tag{4.15}$$

$$\overset{Markov}{=} \frac{p(Z_t|m_i, X_t) \cdot p(m_i|Z_{1:t-1}, X_{1:t-1})}{p(Z_t|Z_{t-1}, X_{1:t})} \tag{4.16}$$

Nach jedem Teil der Transformationen und Vereinfachungen, die in in [TBF05; Mil08] ausführlich erläutert werden, haben die endgültigen Gleichungen die Form von logarithmischen Wahrscheinlichkeiten:

$$p(m_i|Z_{1:t}, X_{1:t}) = 1 - \frac{1}{1 + \exp l_{t,i}} \tag{4.17}$$

$$l_{0:t,i} = l_{0:t-1,i} + \log \frac{p(m_i|Z_{1:t}, X_{1:t})}{1 - p(m_i|Z_{1:t}, X_{1:t})} - \log \frac{p(m_i)}{1 - p(m_i)} \tag{4.18}$$

Wie wir sehen können, ist die Genauigkeit der Karte definitiv von einer genauen Positionsschätzung abhängig, genauso wie die erfolgreiche Lokalisierung von einer korrekten Karte abhängt.

In unserem Algorithmus verwenden wir für das Belegungsraster keine binären, sondern kontinuierliche Werte. Dies wird erreicht, indem mit jeder neuen Messung der Umgebung die Zellen zwischen dem Roboter und dem vom Laser Range finder detektierten Hindernis aktualisiert werden. Je häufiger eine Zelle als leer aktualisiert wird, desto mehr geht ihr Wert gegen Null und umgekehrt. Abbildung 5 zeigt, welche Zellen zwischen Roboter und gemessenem Hindernis aktualisiert werden. Der Mittelpunktswinkel des zu aktualisierenden Kreissektors hängt von der Anzahl an Lasermessungen über den vom Roboter betrachteten Bereich ab.

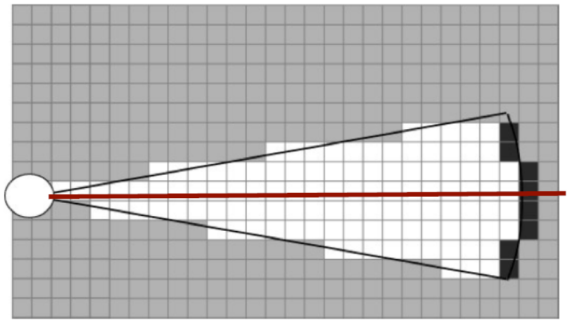

Abbildung 5: Links befindet sich der Roboter und rechts ein Hindernis, dass vom roten Laserstrahl detektiert wurde. Alle Zellen im Kreissektor werden akutalisiert.

Belegungsraster als Variante der Raumrepräsentation werden in der Regel für zweidimensionale Karten verwendet, aber sie sind nicht auf zwei Dimensionen beschränkt. Es gibt viele Techniken, die auf der Raumrepräsentation durch dreidimensionale Belegungsraster basieren [Pa12; JK99; BFP09; Ka11].

4.2.2 Multiskalen-Raster

Beim Multilevel Ansatz für Belegungsraster werden zwei, oder mehr Karten in unterschiedlichen Auflösungsskalen verwaltet. Dabei ermöglichen die gröberen Skalen eine schnelle Eingrenzung des zu durchsuchenden Gebietes beim Bestimmen des globalen Maximums zur Integration neuer Sensordaten. Ist die Zelle der Maximum-Likelihood-Lösung im groben Gitter bestimmt, wird auf die nächst feinere Ebene gewechselt, um die Genauigkeit der Lösung zu erhöhen.

Wir modellieren das Scan-Matching-Problem anhand des probabilistischen Modells, bei dem neue Sensordaten durch die Karte und die aktuelle Position mit ihrer Ausrichtung bestimmt werden, wobei letzteres von der vorherigen Position und den neuen Steuerungen des Roboters abhängt. Der Roboter bewegt sich von x_{t-1} nach x_t, entsprechend einer Steuerung u. Die Lidar-Beobachtung z ist abhängig vom Umgebungsmodell m und der Position des Roboters. Unser Ziel ist es, die posteriore Verteilung über die Position des Roboters zu finden, $p(x_t|x_{t-1}, u, m, z)$. Wir wenden die Bayes-Regel an und entfernen irrelevante Bedingungen, was:

$$p(x_t|x_{t-1}, u, m, z) \propto p(z|x_t, m) \cdot p(x_t|x_{t-1}, u) \tag{4.19}$$

ergibt. Der erste Term, $p(z|x_t, m)$ entspricht dem Beobachtungsmodell: Wie wahrscheinlich ist eine bestimmte Lidar-Messung, falls die Umgebung und die Roboterposition bekannt sind? Der zweite Term, $p(x_t|x_{t-1}, u)$ ist das Bewegungsmodell des Roboters, das beispielsweise durch die Steuerungsbefehle oder die Odometriemessungen erhalten wird.

Während das Bewegungsmodell typischerweise in Form einer multivariaten Gauß-Verteilung bekannt ist, ist das Beobachtungsmodell schwieriger zu berechnen und hat eine komplexere Struktur. Es hat in der Regel mehrere Extrema. Durch den Multilevel Ansatz wird eine effiziente Berechnung der Verteilung $p(z|x_t, m)$ möglich, so dass wir die posteriore Verteilung der Roboterposition aus Gleichung (4.11) bestimmen können. Wir nehmen an, dass die Lidar-Messungen voneinander unabhängig sind, wodurch wir schreiben können:

$$p(z|x_t, m) = \prod_j p(z_j|x_t, m) \tag{4.20}$$

Die Wahrscheinlichkeitsverteilung für eine einzelne Lidar-Messung z_j sollte im Prinzip berücksichtigen, welche Fläche der Karte m von der Position x_t entlang einer bestimmten Peilung sichtbar ist. Zur Reduktion des Rechenaufwandes vernachlässigen wir die Sichtbarkeits- und Verdeckungseffekte und approximieren die Wahrscheinlichkeit von z_j anhand seiner Entfernung von einer beliebigen Fläche in m. Für jeden beobachtbaren Punkt m_i auf der Karte, können wir die bedingte Wahrscheinlichkeit berechnen, dass der Sensor einen nahe gelegenen Punkt p beobachtet, wenn m_i die Ursache für diese Beobachtung ist. Diesen Vorgang wiederholen wir für jeden Punkt auf der Karte.

Im Prinzip müssen wir $p(z|x_t, m)$ über einem dreidimensionalen Volumen von Punkten auswerten. Die drei Dimensionen entsprechen den unbekannten Parametern der Transformation T: Δx, Δy, und θ. Eine naive Implementierung mit drei verschachtelten Schleifen wäre sehr langsam. Stattdessen verwenden wir die Karten mit unterschiedlichen Auflösungen. Bei sehr niedrigen Auflösungen ist es möglich, dass Details in der Karte verschwinden. Deswegen berechnen wir die Karte mit niedriger Auflösung so, dass jede Zelle auf

den Maximalwert der entsprechenden Zellen in der hochauflösenden Karte gesetzt wird. Dadurch wird sichergestellt, dass die von der niedrig auflösenden Karte berechneten Wahrscheinlichkeiten immer mindestens so groß sind wie die der hoch auflösenden Karte. Dadurch wird sichergestellt, dass kein Maximum übersehen wird.

Die Strategie besteht darin, die niedrig auflösende Karte zu nutzen, um schnell Gebiete zu identifizieren, die das globale Maximum enthalten könnten und Gebiete, die es wahrscheinlich nicht enthalten auszuschließen. Das Ziel ist es, das Volumen zu minimieren, das mit hoher Auflösung durchsucht wird. Die Methode besteht aus drei Schritten:

1. Auswerten der Wahrscheinlichkeit $p(z|x_t, m)$ über das gesamte dreidimensionale Suchgebiet anhand der niedrig auflösenden Karte.

2. Finden des besten Voxels im niedrig aufgelösten Raum, das noch nicht berücksichtigt wurde. Bezeichne L_i diesen Wert. Falls $L_i < H_{best}$ wird abgebrochen. H_{best} ist dann die beste Übereinstimmung der Scan-Ausrichtung.

3. Auswerten des Suchvolumens innerhalb des Voxels i anhand der hochauflösenden Karte. Angenommen die log-Wahrscheinlichkeit dieses Voxels ist H_i: Es muss beachtet werden, dass $H_i \leq L_i$, da die niedrig auflösende Karte die log-Wahrscheinlichkeiten überschätzt. Wenn $H_i > H_{best}$ ist, wird $H_{best} = H_i$ gesetzt.

Sobald der Wert der Kostenfunktion über eine Reihe von Positionen ausgewertet wurde, kann eine multivariate Gauß-Verteilung an die Daten angepasst werden. Sei $x_t^{(j)}$ die j-te Auswertung von x_t:

$$K = \sum_j x_t^{(j)} \cdot x_t^{(j)T} \cdot p(x_t^{(j)}|x_{t-1}, u, m, z) \tag{4.21}$$

$$u = \sum_j x_t^{(j)} \cdot p(x_t^{(j)}|x_{t-1}, u, m, z) \tag{4.22}$$

$$s = \sum_j p(x_t^{(j)}|x_{t-1}, u, m, z) \tag{4.23}$$

$$\Sigma_{x_t} = \frac{1}{s}K - \frac{1}{s^2}uu^T \tag{4.24}$$

Bei der Schätzung der Unsicherheit des Scan-Matchers anhand der berechneten Werte von $p(z|x_t, m)$ werden die beiden Hauptquellen der Unsicherheit berücksichtigt: das Rauschen des Sensors selbst und die Unsicherheit darüber, welche Abfragepunkte welchen Teilen des Modells zugeordnet werden sollten. Der Nachteil dieses Ansatzes ist, dass die resultierende Gaußverteilung nur an die berechneten Stichproben angepasst wird. Jegliche Bereiche mit hoher Wahrscheinlichkeit, die nicht innerhalb des ausgewerteten Volumens liegen, werden in der Verteilung nicht widergespiegelt, was zu einer

überhöhten Schätzung führt. Daher ist es wichtig, die Wahrscheinlichkeit $p(z|x_t, m)$ über große Bereiche von x_t auszuwerten, was bei diesem Ansatz getan wird.

5 Ergebnisse

In diesem Kapitel werden wir die in dieser Bachelorarbeit beschriebenen Algorithmen an echten Datensätzen testen, Implementierungsdetails betrachten und verschiedene Eigenschaften der Algorithmen untersuchen. Ziel ist dabei sowohl die Bestimmung der Anzahl notwendiger Partikel, um ausreichend konsistente Karten zu generieren, als auch die Eingrenzung geeigneter Auflösungswerte für die Karten des Multilevel Algorithmus.

Um möglichst nah an reale Experimente mit Robotern, die verschiedene Areale abfahren, heranzukommen, werden wir auf online zur Verfügung gestellte Datensätze zurückgreifen. Diese beinhalten Sensordaten inklusive Odometriemessungen und Laserscans von Robotern, die sich tatsächlich in der realen Welt fortbewegt haben. Besonders geeignet waren der von Dirk Hähnel bereitgestellte Datensatz des Intel Research Labs aus Seattle [Häh03] und der von Mike Bosse und John Leonard bereitgestelle Datensatz des MIT Killian Court aus Massachusetts [BL02], da diese einen einfachen Umang mit den Rohdaten der Robotersensoren ermöglichen.

Insgesamt zeigen die Ergebnisse eine günstige Skalierung auf eine kleine Anzahl von Partikeln und eine große Anzahl von gemessenen Positionen und Scans. Eine feste Anzahl von Partikeln (etwa $M = 100$) scheint für den Multilevel Resolution Algorithmus auch in weitläufigen Umgebungen gut zu funktionieren. Im Folgenden werden wir auf die Qualität der berechneten Karten in Abhängigkeit verschiedener Parameter eingehen.

5.1 Die Datensätze

In diesem Abschnitt werden die beiden Datensätze, an denen die Algorithmen getestet werden, kurz vorgestellt und Eigenschaften wie Länge und Komplexität der Wege, sowie Beschaffenheit der enthaltenen Sensordaten erläutert.

5.1.1 Intel Research Lab

Das Intel Research Lab in Seattle zeichnet sich durch seine relativ kompakte Struktur aus. Das Gebäude steht auf einer Grundfläche von 30 x 30 Metern und der längste Kreis, den der Roboter auf seinem Weg schließt, misst etwa 70m. Beim Durchqueren des Gebäudes fährt der Roboter zunächst einige Male entlang des Rundweges im Inneren und biegt dann nacheinander in die davon abgehenden Räume ein, bevor er erneut dem Rundweg folgt. Aufgrund der Kompaktheit des zu kartierenden Areals und der relativ kurzen Kreise, die der Roboter schließt, können SLAM Algorithmen verglichen

mit weitläufigeren Datensätzen, wie dem MIT Killian Court Datensatz, vom Zeitaufwand und Speicherbedarf her relativ kostengünstig angewendet werden. Abbildung 6 zeigt eine Karte die aus dem Intel Research Lab Datensatz generiert und gemeinsam mit den Rohdaten bereitgestellt wurde, um eine grobe Orientierung zu bieten.

Der Datensatz umfasst die Rohdaten-Einträge der Odometriesensoren bezüglich der frontalen und seitlichen Fortbewegung, sowie der Rotationswinkelmessung des Roboters. Die Daten zur Fortbewegung sind mit einer Genauigkeit von 1mm gegeben und die Daten zur Rotation mit einer Genauigkeit von etwa einem tausendstel Grad. Der Laser Range Finder deckt den frontalen Halbkreis um den Roboter, also einen Bereich von 180° ab. Dieser wird mit einem Laserscan pro Grad, also insgesamt 180 Lasermessungen, abgedeckt. Jede dieser Entfernungsmessungen hat eine Genauigkeit von 1cm. Die Strecke die der Roboter während der Aufzeichnung des Datensatzes insgesamt zurückgelegt hat, beträgt 506m.

Abbildung 6: Veranschaulichung des Intel Research Lab Gebäudes.

5.1.2 MIT Killian Court

Beim MIT Killian Court Datensatz aus Massachusetts handelt es sich um die Kartierung eines recht weitläufigen Gebäudes. Insgesamt hat die Umgebung auf der sich das Gebäude befindet eine Größe von 250 x 215 Metern und der Roboter legt beim Durchfahren eine Strecke von 1,9 Kilometern zurück. Der längste Kreis, der dabei geschlossen wird, hat eine Länge von etwa 320m, was für SLAM Algorithmen deutlich anspruchsvoller ist, als die etwa 70m, des Intel Research Lab Datensatzes. Außerdem wird unterwegs nicht in verschiedene Räume abgezweigt. Abbildung 7 zeigt eine Karte die aus dem Killian

Court Datensatz generiert und gemeinsam mit den Rohdaten bereitgestellt wurde, um eine grobe Orientierung zu bieten.

Die Rohdaten-Einträge des Datensatzes entsprechen im wesentlichen denen des Intel Research Lab Datensatzes. Dies hat den Vorteil, dass eine Vergleichbarkeit der Ergebnisse der SLAM Algorithmen auf den Datensätzen gewährleistet ist. Es gibt also wieder 180 Lasermessungen, die gleichmäßig auf den Halbkreis vor dem Roboter verteilt sind und auch die Genauigkeiten der Sensordaten stimmen überein. Zusätzlich zu den Lidar- und Odometriemessungen enthält dieser Datensatz Messungen eines Sonars, die wir für unsere Zwecke allerdings nicht benötigen.

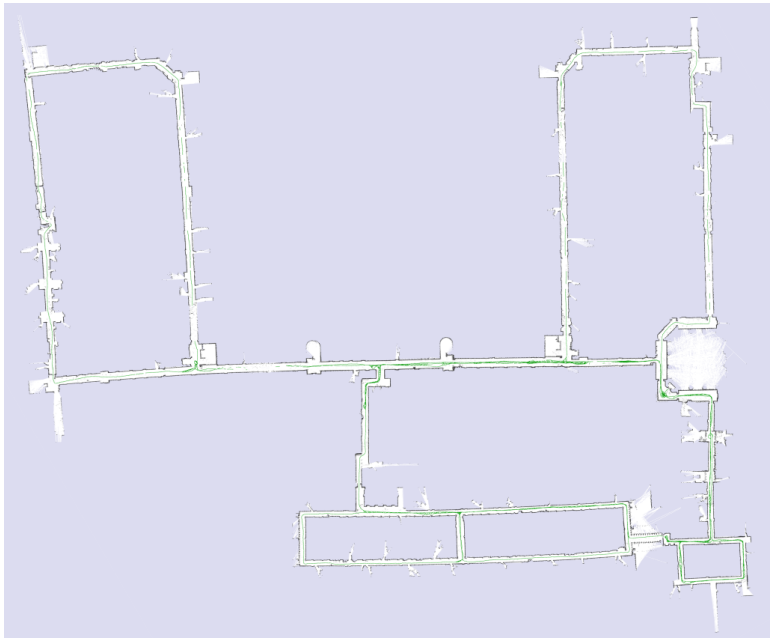

Abbildung 7: Veranschaulichung des MIT Killian Court Gebäudes.

5.2 Direkte Kartierung von Rohdaten

Betrachten wir den Intel Research Lab Datensatz zunächst ohne jegliches Scan-Matching und ohne eine Wahrscheinlichkeitsverteilung für die einzelnen Positionen. Wenn wir nur rohe Odometriedaten als Grundwahrheit für die Positionen des Roboters benutzen, also die aktuelle Position aus der Summe der bisherigen Odometriemessungen gewinnen, und die Laserscans jeweils an diesen Positionen in die Karte integrieren, erhalten wir eine Karte der Umgebung wie in Abbildung 8.

Der Drift, der bei jeder Odometriemessung entsteht und sich mit jeder neuen Messung akkumuliert, ist so stark, dass die entstehende Karte so verzerrt ist, dass sie zur Navigation von Robotern vollkommen ungeeignet ist.

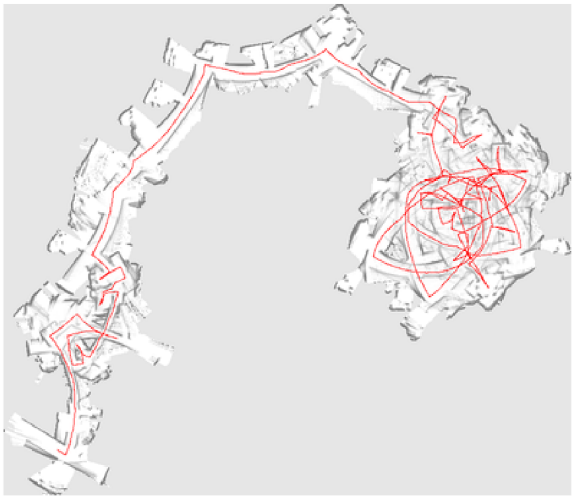

Abbildung 8: Mapping mit Positionsschätzungen, die allein auf den Odometriedaten basieren. Es ist keine sinnvolle Navigation mit der entstehenden Karte möglich. Die Trajektorie des Roboters ist in rot verzeichnet.

Man kann auf der Karte zwar noch vereinzelt Korridore erahnen und Teile der Karte lassen auf Räume schließen, aber alles was über diese vagen Schätzungen hinaus geht, wäre willkürlich geraten. Die Notwendigkeit von Scan-Matching Algorithmen und Positions-Wahrscheinlichkeitsverteilungen wird somit offensichtlich.

5.3 Anzahl verwendeter Partikel

In diesem Abschnitt werden wir auf die Anzahl der verwendeten Partikel im Partikel-Filter eingehen und dafür zunächst vom Algorithmus erzeugte Karten in Abhängigkeit einzelner Partikel betrachten. Generell repräsentiert jeder Partikel sowohl eine mögliche Schätzung der aktuellen Position des Roboters, als auch eine Karte, basierend auf dessen bisherigen Positionen. Dabei wiegen die Gewichte der einzelnen Partikel ab, wie wahrscheinlich es ist, dass der jeweilige Partikel die Grundwahrheit möglichst gut approximiert. Je mehr Partikel verwendet werden, desto besser wird also die Wahrscheinlichkeitsverteilung der bisherigen Roboterpositionen approximiert. Beispielhaft dafür sind in Abbildung 9 mehrere Partikel dargestellt, die jeweils ihre eigene Trajektorie mit sich bringen, aus der sich wiederum unterschiedliche Karten ergeben.

Insbesondere befinden sich die Partikel in Abbildung 9 gerade an einer Stelle, bei welcher der Roboter einen Kreis abschließt, nachdem sie sich einmal im Uhrzeigersinn durch das Gebäude bewegt haben. Mit ein wenig Erfahrung beim Betrachten von Belegungsraster-Karten sieht man, dass einige Hindernisse in den Karten übereinstimmen, andere aber zu Unterschieden in

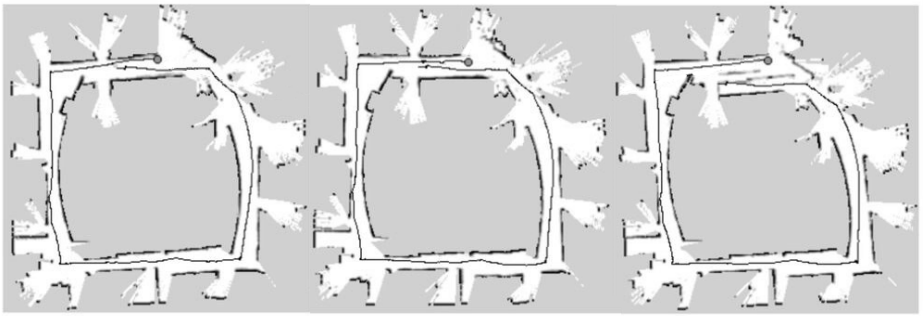

Abbildung 9: Partikel-Filter Beispiel mit unterschiedlichen Karten, die aus unterschiedlichen Partikeltrajektorien entstehen. Von links nach rechts sind die Karten den Partikeln 1 bis 3 zugeordnet.

den Karte führen. So ist bei Partikel 3 beispielsweise eine Art Geisterkorridor beim schließen des Kreises entstanden. Dies resultiert aus einer vorangegangenen falschen Positionsschätzung des Roboters, die dazu führt, dass die Karte inkonsistent wird, da es Hindernisse in der Karte gibt, die nicht mit der echten Umgebung des Roboters übereinstimmen. Bei Partikel 2 existiert zwar kein Geisterkorridor, allerdings gibt es einen leichten Versatz, also eine kleine Verschiebung zwischen den neu gemessenen und laut Karte vorhandenen Hindernissen vor dem Roboter (in der oberen rechten Ecke der Karte). Dieser Versatz ist eine geringere Abweichung von der bisherigen Karte, als bei Partikel 3, aber lediglich Partikel 1 hat eine sehr gute Übereinstimmung mit dem bisherigen Kartenmodell. Diese unterschiedlich gute Übereinstimmung mit dem bisherigen Kartenmodell schlägt sich in der Gewichtung der Partikel nieder. Partikel 3 wird also das niedrigste Gewicht erhalten, Partikel 1 das größte und das Gewicht von Partikel 2 wird irgendwo zwischen den Gewichten von Partikel 1 und 3 liegen.

Wie viele Partikel werden nun benötigt, um eine mit der wahren Umgebung des Roboters konsistente Karte zu erhalten? Das hängt von mehreren Faktoren ab:

- Zunächst ist die Genauigkeit der Robotersensoren wichtig. Je größer die Unsicherheit der Odometriemessungen und der Laserscans ist, desto flacher ist die Kurve der Positions-Wahrscheinlichkeitsverteilung und umso mehr Partikel werden benötigt, um diese Kurve gut zu approximieren.

- Auch die Qualität des Scan-Matching-Algorithmus hat einen großen Einfluss auf die Anzahl der benötigten Partikel. Je genauer dieser die Position des Roboters in der Karte anhand dessen Laserscan-Beobachtung

eingrenzen kann, desto weniger Partikel müssen zur Positionschätzung verwendet werden.

- Desweiteren spielt die Beschaffenheit der Roboterumgebung eine wichtige Rolle. Befinden sich viel markante Objekte in Reichweite des Laserscanners, kann der Scan Matching Algorithmus einen guten Rückschluss auf die aktuelle Position ziehen. Sind die Objekte jedoch weniger markant, wie beispielsweise ein einfacher Korridor, ist lediglich eine Ausrichtung in Richtung der Korridorwände möglich. In Richtung des Korridors fällt man jedoch auf die Schätzung durch die Odometriedaten zurück. Befindet sich der Roboter allerdings auf freier Fläche, können ausschließlich die ungenauen Odometriedaten zur Positionsschätzung verwendet werden, was wiederum durch eine hohe Partikelanzahl ausgeglichen werden muss. Eine Visualisierung des Einflusses der Umgebung auf die notwendige Partikelanzahl bietet Abbildung 10. Von links nach rechts sind dort eine freie Fläche, ein offener Korridor und eine markante Sackgasse mit der jeweils benötigten Partikelanzahl abgebildet.

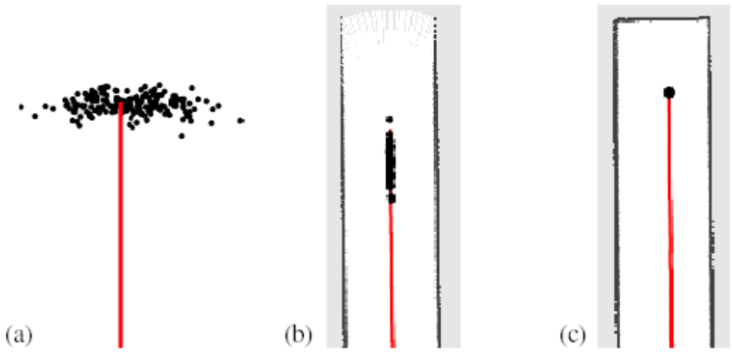

Abbildung 10: Anzahl benötigter Partikel zur Approximation der Wahrscheinlichkeitsverteilung über die Roboterposition in Abhängigkeit der Umgebung.

- Zuguterletzt spielt natürlich auch die Größe des zu kartierenden Areals und insbesondere die Wahl der Route mit der Länge von Wegen, bevor es wieder zum schließen von Kreisen kommt, eine wichtige Rolle. Falls es zu lange keinen neuen Kreisschluss gibt, werden die akkumulierten Messfehler zu groß, als dass sie rückwirkend noch sinnvoll korrigiert werden könnten. Die Wahrscheinlichkeit, dass noch ein Partikel existiert, der sehr nah an der echten Robotertrajektorie liegt, wird also immer kleiner, was nur durch eine unvertretbar hohe Anzahl von Partikeln berücksichtigt werden könnte.

Da die Laufzeitkosten und der Speicherplatzbedarf linear in der Anzahl der verwendeten Partikel steigt, sollte diese sehr gering gehalten werden. Die Wichtigkeit eines guten Scan-Matching Algorithmus zeigt sich, wenn man den Intel Lab Datensatz mit einem Partikelfilter ohne Scan-Matching kartiert. Selbst bei der Verwendung von 2000 Partikeln wird die Kurve der Wahrscheinlichkeitsverteilung über die Roboterpositionen durch die akkumulierten Fehler so flach, dass die Anzahl an Partikeln nicht mehr zur sinnvollen Approximation der Verteilung ausreicht. Entsprechend lässt sich auch die dabei entstehende Karte in Abbildung 11 nicht mehr zur Navigation des Roboters verwenden, da sie inkonsistent ist.

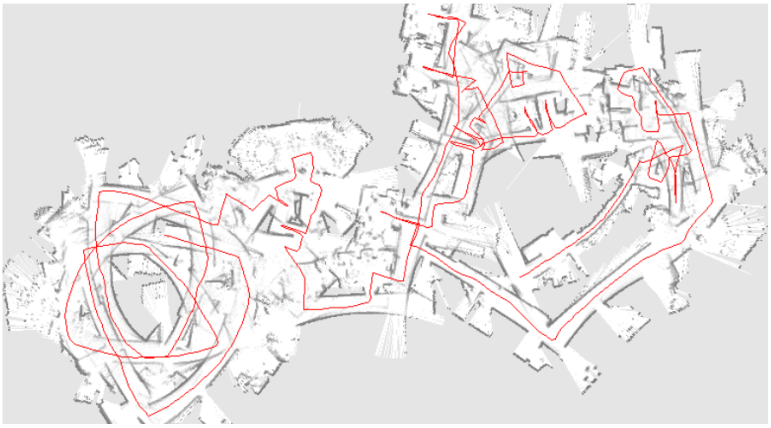

Abbildung 11: Inkonsistente Karte des Intel Research Labs, trotz hoher Partikel Zahl von 2000 Partikeln, bei ausschließlicher Verwendung eines Partikelfilters.

Mit Verwendung eines Scan-Matching Algorithmus lässt sich schließlich eine in sich konsistente Karte erstellen, während gleichzeitig die Anzahl an verwendeten Partikeln reduziert wird. Voraussetzung dafür ist in Kombination mit dem Scan-Matching natürlich ein entprechendes Resampling der Partikel in Abhängigkeit von ihrem Gewicht. Die Karte in Abbildung 12 wurde mit lediglich 20 Partikeln generiert und weist bereits keinerlei visuell offensichtliche Ungenauigkeiten mehr auf, wie sie in Abbildung 9 zu sehen waren.

Eine weitere Verbesserung erzielt schließlich das Multilevel-Verfahren aus Abschnitt 4.2.2, mit dem die Anzahl der Partikel auf zehn Stück reduziert werden kann, während sich gleichzeitig die Laufzeit der Datenassoziierung drastisch reduziert. Abbildung 13 zeigt die Karte, die mithilfe des Multilevel-Verfahrens generiert wurde. Selbst mit der geringen Anzahl von nur zehn Partikeln, weist sie genau so wenig Ungenauigkeiten auf, wie das einfache Partikel-Filter Verfahren mit Scan-Matching, das noch doppelt so viele Partikel für eine Karte gleicher Qualität benötigte.

Nun wenden wir den Multilevel Algorithmus noch auf den Datensatz an,

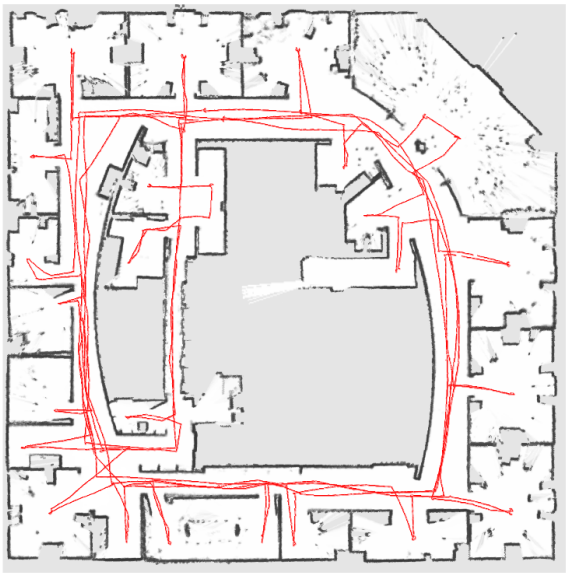

Abbildung 12: Konsistente Karte des Intel Research Labs, mit 20 Partikeln, unter Verwendung eines Partikelfilters mit Scan-Matching.

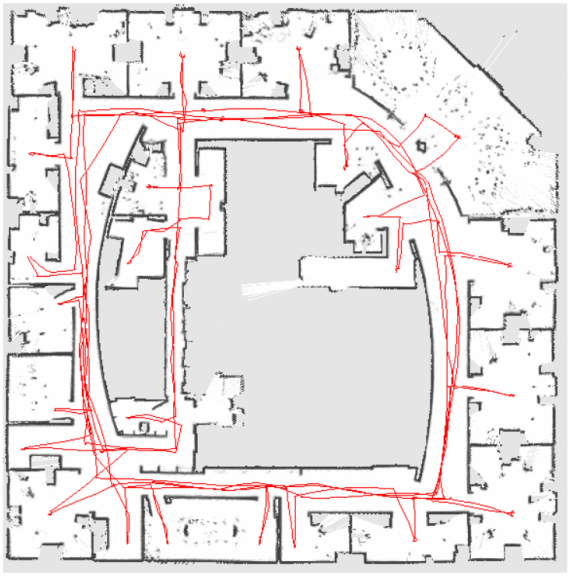

Abbildung 13: Konsistente Karte des Intel Research Labs, mit 10 Partikeln, unter Verwendung des Multilevel Algorithmus aus Abschnitt 4.2.2.

der am MIT Killian Court erfasst wurde. Dieser Datensatz ist extrem anspruchsvoll, da er mehrere verschachtelte Kreise enthält, was dazu führen kann, dass der Rao-Blackwell'sche Partikelfilter aufgrund von Partikelauslö-

schung inkonsistente Karten erzeugt. Bei der Verwendung dieses Datensatzes erwies sich das selektive Resampling Verfahren als besonders wichtig. Eine konsistente und topologisch korrekte Karte kann mit mindestens 70 Partikeln erzeugt werden. Allerdings zeigen die resultierenden Karten dann manchmal inkonsistente Scheinwände. Durch Verwendung von mindestens 90 Partikeln ist es möglich, qualitativ hochwertige Karten zu generieren. Die daraus resultierende Karte ist in Abbildung 14 zu sehen. Im Vergleich zu Abbildung 7 fällt auf, dass die Korridore genau parallel zueinander verlaufen, während die Korridore in Abbildung 7 zum Teil leichte Kurven beschreiben, die nicht der Realität entsprechen. Der Multilevel Algorithmus ist somit in der Lage selbst bei schwierigen Datensätzen noch hervorragende Ergebnisse zu erzeugen. Zwar ist die Anzahl von Partikeln, die für dieses gute Resultat erforderlich ist, mit 90 Stück deutlich höher als beim Intel Reasearch Lab Datensatz, aber die Komplexität des MIT Killian Court fällt auch deutlich höher aus, was die höhere Anzahl an Partikeln relativiert.

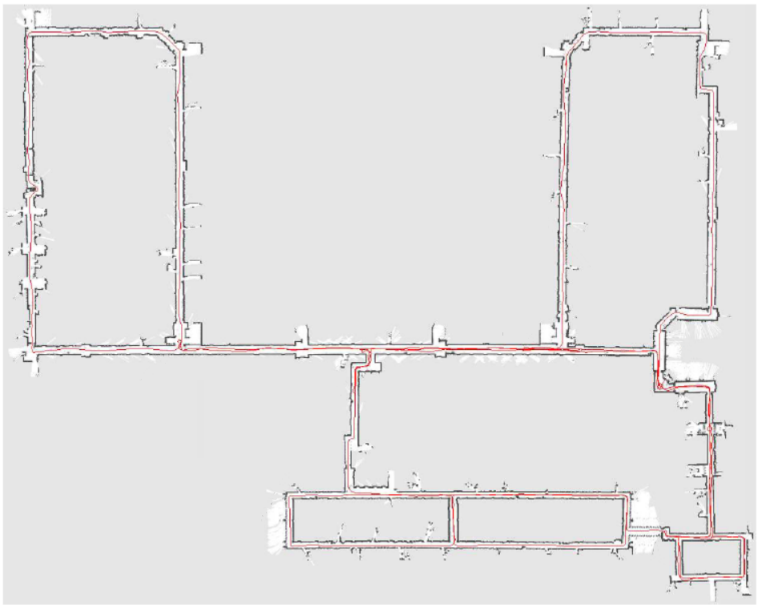

Abbildung 14: Konsistente Karte des MIT Killian Court in Massachusetts, mit 90 Partikeln, unter Verwendung des Rao-Blackwell'schen Partikelfilters.

5.4 Auflösung der Multilevel Karten

Genau wie die Anzahl der verwendeten Partikel, hat auch die Auflösung der Multilevel Karten einen Einfluss auf die Genauigkeit der generierten Karten. Für die feine Karte, der beiden in unserem Multilevel Algorithmus verwendeten Karten, gilt: Ist die Auflösung zu grob, kann die Position des Roboters

nicht genau genug bestimmt werden und es kann infolge dessen zu Inkonsistenzen in der Karte kommen. Ist die Auflösung zu fein, steigt sowohl der Speicherplatzbedarf, als auch die benötigte Zeit zum berechnen der Wahrscheinlichkeitsverteilung um die Maximum-Likelihood-Position des Roboters stark an. Für die grobe Karte gilt: Ist die Auflösung zu fein wird der Beschleunigungseffekt des Verfahrens zunichte gemacht, der durch die Vereinfachung der Karte erzielt wird. Ist die Auflösung zu grob können die Details, die zur korrekten Lokalisierung des Roboters benötigt werden, ineinander übergehen, sodass die Karte mit den dann verschwommenen Objekten keine eindeutige Lokalisierung mehr zulässt, was letztendlich zu einer inkonsistenten Karte führt. Zusätzlich muss auf den Unterschied zwischen den Auflösungen der beiden Karten geachtet werden, damit der Beschleunigungseffekt, den das Multilevel-Verfahren mit sich bringt, möglichst groß ist.

Nach einer Vielzahl von Tests mit den beiden Datensätzen aus Abschnitt 5.1 kann der optimale Bereich für die feine und grobe Auflösung der beiden Karten des Multilevel Resolution Algorithmus angegeben werden. Untersucht wurden insgesamt Auflösungen im Bereich von 0,5 cm bis 1,5 m. Dabei wurde sowohl der Speicherplatzbedarf, als auch die Laufzeit des Algorithmus und natürlich die Konsistenz der generierten Karte in Betracht gezogen. Bei feineren Auflösungen als 2 cm konnte keine Verbesserung der Ergebnisse mehr festgestellt werden, während eine Auflösung von 5,5 cm nur geringfügig schlechtere Ergebnisse erzielte. Ab Auflösungen von mehr als 8 cm nimmt die Qualität der generierten Karten so schnell ab, dass von gröberen Auflösungen abzuraten ist. Die grobe Karte sollte ihre Auflösung an der feinen Karte orientieren und etwa um den Faktor 10 gröber sein. Dementsprechend sollte ihre Auflösung zwischen 20 cm und 60 cm betragen. Als guter Kompromiss zwischen Qualität und Laufzeit, hat sich bei der Anwendung des Algorithmus eine Auflösung von 4 cm für die feine Karte und eine Auflösung von 40 cm für die grobe Karte bewährt. Dies kann somit für eine zukünftige Anwendung des Multilevel Resolution Algorithmus empfohlen werden. Die Laufzeitbeschleunigung, die durch das Multilevel-Verfahren gegenüber dem einfachen Rao-Blackwell'schen Partikelfilter erzielt wurde liegt, in etwa bei einem Faktor 100.

6 Zusammenfassung

In dieser Bachelorarbeit wird das Simultaneous Localization and Mapping Problem, zusammen mit einem Multilevel-Verfahren zur Implementierung eines Algorithmus der dieses löst, vorgestellt. Zur Hinführung an die Lösung des Problems wird dabei zunächst ein Least-Square Scan-Matching Verfahren betrachtet.

Der Algorithmus, der das Multilevel-Verfahren implementiert, kombiniert das Konzept der Optimierung auf verschiedenen Skalen aus dem Paper von

Frese, Larsson und Duckett [FLD05] mit dem stochastischen Ansatz der Partikelfilter aus dem FastSLAM 2.0 Algorithmus [Mon+03]. Als Kartengrundlage dienen dabei Belegungsraster, die gut geeignet für Umgebungen mit einer dichten beobachtbaren Struktur wie z.B. Innenräume sind und gleichzeitig Vorteile bei der Pfadplanung mit Hindernisvermeidung mit sich bringen, was für autonom fahrende Roboter unerlässlich ist. Der Multilevel-Algorithmus vereint somit eine rechnerisch effiziente Vorgehensweise, mit einer stochastisch ausgeklügelten und auf Laser Range Daten angepassten Methode zusammen mit einer kompakten und für die Navigation mobiler Roboter praktischen Kartendarstellung.

Die Ergebnisse bei Anwendung des Algorithmus auf verschiedene Datensätze, zeigen die Robustheit des Multilevel-Ansatzes. Dabei wird die Parameterwahl sowohl bezüglich der verschiedenen Skalen der Karten, als auch bezüglich der Anzahl benötigter Partikel im Partikelfilter analysiert. Insgesamt bringt die Kombination der verschiedenen Ansätze einen flexiblen und leistungsstarken SLAM-Algorithmus hervor, der qualitativ hochwertige Karten generiert.

Literatur

[BD06] T. Bailey und H. Durrant-Whyte. “Simultaneous localization and mapping (SLAM): part II”. In: *IEEE Robotics & Automation Magazine* 13.3 (2006), S. 108–117. DOI: 10.1109/MRA.2006.1678144. URL: https://ieeexplore.ieee.org/document/1678144.

[BFP09] H. Badino, U. Franke und D. Pfeiffer. *The Stixel world-a compact medium level representation of the 3D-world*. Springer, 2009, S. 51–60.

[BL02] Mike Bosse und John Leonard. *Raw log data from Odometry and a 2D SICK LMS Laser Range Finder from the Interior of the MIT Killian Court*. 2002. URL: http://ais.informatik.uni-freiburg.de/slamevaluation/datasets/mit-killian.clf.

[Cox90] Ingemar J. Cox. *Blanche: Position estimation for an autonomous robot vehilce*. 1990.

[DB06] H. Durrant-Whyte und T. Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE Robotics & Automation Magazine* 13.2 (2006), S. 99–110. DOI: 10.1109/MRA.2006.1638022. URL: https://ieeexplore.ieee.org/document/1638022.

[DFG01] A. Doucet, N. de Freitas und N. Gordan. *Sequential Monte-Carlo Methods in Practice*. Springer Verlag, 2001.

[Fa01] D. Fox und et al. *Particle filters for mobile robot localization*. Berlin: Springer, 2001, S. 401–428.

[FBF77] Jerome H. Friedman, Jon L. Bentley und Raphael A Finkel. “An algorithm for finding best matches in logarithmic expected time”. In: *ACM Transactions on Mathematical Software (TOMS)* 3 (1977), S. 209–226.

[FLD05] Udo Frese, Per Larsson und Tom Duckett. “A Multilevel Relaxation Algorithm for Simultaneous Localization and Mapping”. In: *IEEE Transactions on Robotics* (2005). DOI: 10.1109/TRO.2004.839220. URL: https://www.researchgate.net/publication/3450147.

[Gri15] Giorgio Grisetti. “Notes on Least-Squares and SLAM”. In: (2015).

[Häh03] Dirk Hähnel. *Raw log data from Odometry and a 2D SICK LMS Laser Range Finder from the Interior of the Intel Research Lab*. 2003. URL: https://www.mrpt.org/Dataset_Intel_2003.

[JK99] A.E. Johnson und S.B. Kang. *Registration and integration of textured 3D data*. 2. Aufl. Bd. 17. 1999, S. 135–147.

[Ka11] S. Kohlbrecher und et al. *A flexible and scalable SLAM system with full 3D motion estimation*. 2011, S. 155–160.

[LM97] F. Lu und E. Milios. *Globally consistent range scan alignment for environment mapping*. 1997, S. 333–349.

[Mil08] A. Milstein. *Occupancy grid maps for localization and mapping*. London: InTech, 2008, S. 382–408.

[Mon+03] Michael Montemerlo u. a. "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges". In: *Proc. IJCAI Int. Joint Conf. Artif. Intell.* (Juni 2003).

[Mor88] H.P. Moravec. *Sensor fusion in certainty grids for mobile robots*. Bd. 4. Summer 1988, S. 61–74.

[Mur99] K. Murphy. *Bayesian map learning in dynamic environments*. Denver, CO, USA, 1999, S. 1015–1021.

[Pa12] B. Peasley und et al. *Accurate on-line 3D occupancy grids using Manhattan world constraints*. 2012, S. 5283–5290.

[Sun12] Dipl.-Inf. Niko Sunderhauh. "Robust Optimization for Simultaneous Localization and Mapping". Technischen Universitat Chemnitz, 2012. URL: `https://eprints.qut.edu.au/109667/1/109667.pdf`.

[Ta01] S. Thrun und et al. *Robust Monte Carlo localization for mobile robots*. 1-2. Bd. 128. 2001, S. 99–141.

[TBF05] S. Thrun, W. Burgard und D. Fox. *Probabilistic Robotics*. Cambridge: The MIT Press, 2005.