Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2326

Entwicklung eines objektorientierten Frameworks für Simultane Lokalisierung und Kartierung

Zhen Peng

Studiengang:	Informatik
Prüfer:	Prof. Dr. Thomas Ertl
Betreuer:	Dipl.-Inf. Bernhard Schmitz, Dipl.-Inf. Bernhard Kleiner
begonnen am:	09. März 2011
beendet am:	16. Dezember 2011
CR-Klassifikation:	C.3, D.2.2, J.7

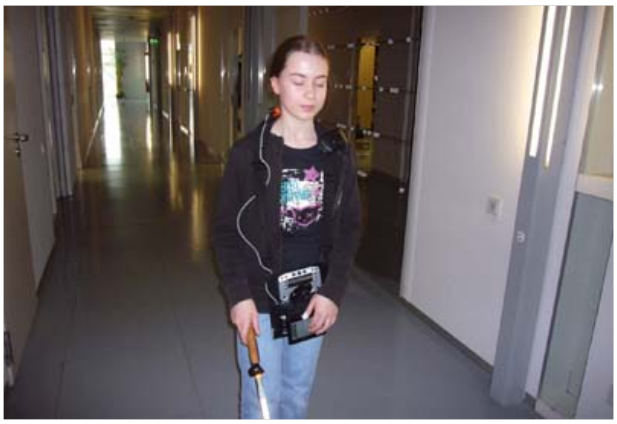
Inhaltsverzeichnis

Abbildungsverzeichnis

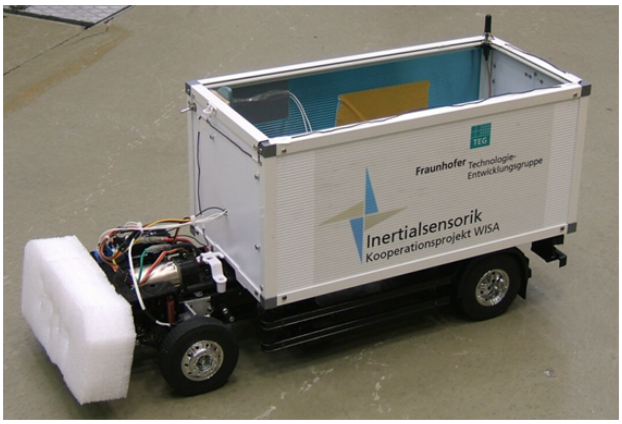
1. Einleitung

Das Projekt *Assistenz für sensorisch Behinderte an der Universität Stuttgart* (ASBUS) entwickelt ein Assistenzsystem, um Studierende mit Sehbehinderungen bei der Navigation im Universitätsgelände zu unterstützen. Das Navigationssystem ist mit GPS, RFID-Reader und einer *Inertialen Messeinheit* (IMU) ausgestattet (Abbildung 1.1). Die *Inertiale Messeinheit* (IMU) ist einer der am häufigsten verwendeten Sensortypen. Ein typisches IMU-System besteht aus drei Beschleunigungs- sowie drei Drehratensensoren, und misst damit die Beschleunigung sowie die Rotation eines Körpers relativ zu einem Bezugssystem. In diesem Navigationssystem wird ein Näherungsverfahren verwendet, um die Geschwindigkeit der Benutzer durch die Messwerte der Beschleunigungssensoren und durch die zwei Walking-Parameter zu berechnen [Komo6] . Wegen den unterschiedlichen Laufgewohnheiten und Körperlängen sind diese Parameter für jede Person sehr verschieden. Um genaue Walking-Parameter zu erhalten, müssen zahlreiche, wiederholte experimentelle Untersuchungen im Voraus durchgeführt werden. Falls der Benutzer gewechselt wird, müssen die Walking-Parameter erneut festgestellt und dann die alten Werte im Navigationssystem durch die neuen ersetzt werden, um eine fehlerhafte Schätzung der Geschwindigkeit zu vermeiden. Wenn dieser Algorithmus zur Erfassung der Bewegungsinformation des Benutzers durch Sensorfusionsverfahren ersetzt wird, erhöhen sich sowohl die Benutzerfreundlichkeit als auch die Genauigkeit des Navigationssystems erheblich. Für die Sensorfusion im Bereich der Fußgänger-Navigationssysteme wurde eine Menge an Forschungsarbeiten veröffentlicht, beispielsweise unter Anwendung von IMU-Sensoren [RAKo9] , Fusion von IMU und RFID-Signal [RGHR10] , von GPS [AHWOW04] und von Partikel-Filtern [CNo3, WKARo6] . Daher ist ein Umstieg auf Sensorfusionsverfahren notwendig.

In derselben Zeit forscht das Fraunhofer Institut IPA auf dem Gebiet der Navigation mit low-cost Sensoren und erreichte in der Vergangenheit schon große Fortschritte im Bereich der Sensorik und Sensorfusion. Rosenberg stellte eine Methode für die Zustandsschätzung eines mit IMU ausgestatteten Fahrzeuges (die Abbildung 1.1) mittels Bayes-Filtern vor [Roso6] , welche eine kurzzeitige Genauigkeit im Zentimeterbereich zulassen. Basierend auf dieser Arbeit wurden weitere Navigationssysteme entwickelt [Kleo8] , welche mittels einer Kombination von IMU und optischen Sensoren, wie z.B. Kameras, das Umfeld dreidimensional erfassen und somit eine Beobachtung der eigenen Bewegung zulassen. Mittels Sensordatenfusion werden die Raumwinkel und die Position eines Systems ohne Bezug auf äußere Referenzen ermittelt. Die Technologie der inertialen und optischen Bewegungserfassung wird in einem *Simultaneous Localization and Mapping* (SLAM) -Algorithmus kombiniert. Dieser erlaubt es, Umgebungs- und Bewegungserfassung gleichzeitig auszuführen.



(a) Das Navigationssystem im ASBUS-Projekt.



(b) Das mit IMU ausgestattete Fahrzeug in [Roso6] .

Abbildung 1.1.: Zwei Navigationssysteme.

1.1. Aufgabenstellung

Die Zielsetzung der vorliegenden Arbeit bestand darin, ein universelles C++ Software-Framework als Basis für die Entwicklung von SLAM-Applikationen zu entwickeln. Das Framework beschränkt sich nicht auf spezifische Probleme, sondern versucht, ein SLAM-System zu modularisieren und stellt einige gängige Sensorfusions-Verfahren zur Verfügung, um die spätere Entwicklung einer SLAM-Applikation zu erleichtern. Ein weiterer Schwerpunkt ist die Einbeziehung und Erstellung von Karten in das Software-Framework. Außerdem soll die Erweiterbarkeit besonders berücksichtigt werden.

1.2. Überblick über die Arbeit

Kapitel 2 beschreibt das SLAM-Problem und stellt die Charakteristik sowie die Klassifikation der SLAM-Verfahren vor.

Kapitel 3 widmet sich der Sensorfusion von verschiedenen Sensoren.

Kapitel 4 erläutert die Grundlagen von Karten in SLAM-Applikationen.

Kapitel 5 beschreibt, wie das SLAM-Framework implementiert wurde.

Kapitel 6 schließt die Arbeit mit der Diskussion zur Anwendung des Frameworks ab und bietet einen Ausblick auf weiterführende Entwicklungen und Möglichkeiten.

2. SLAM

Lokalisierung und Kartenerstellung für Anwendungen von mobilen Navigationssystemen sind aktuelle Forschungsthemen. Für die Erstellung einer Umgebungsmodellierung mithilfe von präziser Positionierung oder die Lokalisierung mittels einer vorhanden Umgebungskarte existiert bereits eine Reihe von praktischen Lösungen. Allerdings kann ein Navigationssystem in vielen Umgebungen nicht mit dem *Global-Positioning-System* (GPS) lokalisiert werden, die Erstellung einer Karte des Arbeitsumfelds im Voraus ist ebenfalls oft sehr schwierig oder gar unmöglich. In diesem Fall muss ein Navigationssystem in einer völlig unbekannten Umgebung eine Karte erstellen und gleichzeitig mit der generierten Karte sich selbst positionieren und navigieren. Dies wird SLAM-Problem (*Simultaneous Localization and Mapping*) genannt.

Eine wegweisender Beitrag auf dem Forschungsgebiet der SLAM-Probleme ist die Arbeit von Randall Smith und Peter Cheeseman über die Abschätzung der räumlichen Unsicherheit in der Robotik [SC86, SSC86] . SLAM-Systeme wurden dann in verschiedenen Umgebungen angewandt, wie zum Beispiel Indoor- [LDW91, CMNT98] , Unterwasser- [LF99, WND+00] und Outdoor-Umgebungen [GNW00, JG00] .

2.1. Klassifikation

Nach den unterschiedlichen theoretischen Grundlagen können SLAM-Algorithmen wie folgt charakterisiert werden.

2.1.1. EKF-basiertes SLAM

Der *Extended Kalman Filter* (EKF) ist eines der am häufigsten verwendeten SLAM-Verfahren zur Lösung von Schätzproblemen in nichtlinearen Systemen. Die Position des Navigationssystems wird in 2D-Koordinaten gespeichert. Das System wird in zwei nichtlineare Modelle unterteilt: das System-Modell und das Mess-Modell. Bewegungsinformationen werden durch die beiden Modelle mithilfe eines erweiterten Kalman-Filter rekursiv ermittelt. Für gewöhnlich gibt es zwei getrennte Schritte: die Schätzung und die Aktualisierung [LF99, GNW00] .

2.1.2. Wahrscheinlichkeits-basiertes SLAM

Das Problem der Unsicherheit bei der Lokalisierung mit einer Wahrscheinlichkeitsfunktion zu lösen, erscheint sowohl intuitiv als auch vernünftig. Das populärste dieser Verfahren wird als *Maximum-Likelihood-Estimation* (MLE) bezeichnet. Ein sehr effizienter Algorithmus für MLE ist der sogenannte *Baum-Welch* (auch $\alpha - \beta$)-Algorithmus [TBF98] . Dieser Algorithmus umfasst zwei Schritte: E-Step (Erwartung) und M-Step (Maximierung).

2.1.3. Partikelfilter-basiertes SLAM

Ein SLAM-Verfahren mit Partikel-Filter wird auch als *Monte-Carlo-Lokalisierung* bezeichnet [DFBT99] [YM02] . Die Grundidee ist, die unbekannte Wahrscheinlichkeitsdichte über den Zustandsraum zu schätzen, indem eine Menge an Partikeln zufällig eingesetzt werden. Jedem einzelnen Partikel wird nun mittels des stochastischen Modells der Systemdynamik eine oder mehrere Lösungskurven zugeordnet. Wenn die aus dieser Lösungskurve abgeleiteten Vorhersagen mit den tatsächlichen Messwerten übereinstimmen, erhöht sich das Gewicht der Partikel. Am Ende ergibt sich eine verbesserte Schätzung der Wahrscheinlichkeitsdichte im Zustandsraum.

2.1.4. SEIF-basiertes SLAM

Der Informationsfilter ist eine andere mathematische Beschreibung des Kalman-Filters, welcher bei den gleichen Eingangsdaten und gleichen Modellen das gleiche Ergebnis liefert. Der *Sparse Extended Information Filter* (SEIF) wurde von Sebastian Thrun und anderen vorgeschlagen [TKG+02], als Verbesserung des EKF-Algorithmus. Statt der Kovarianzmatrix wird eine räumliche Informationsmatrix verwendet, um die innere Beziehung zwischen räumliche Informationen zu repräsentieren. Die Karte wird durch lokale, Web-ähnliche Netzwerke von Merkmalen dargestellt. Bei der Aktualisierung der Informationsmatrix werden nur benachbarte Merkmale betrachtet, dadurch kann die sie in einer konstanten Zeit ausgeführt werden, unabhängig von der Anzahl der Merkmale in der Karte.

2.1.5. Klassifikation anhand Kartentypen

Außerdem können SLAM-Verfahren nach unterschiedlichen Kartenrepräsentationen unterteilt werden: Grid-basiertes SLAM [Elf87], Feature-basiertes SLAM [GNW00, JG00, TKG+02, HS08]und Topologie-basiertes SLAM [CN01]. Die Einstufung der Karten wird in Kapitel 4 im Detail vorgestellt.

2.2. Unsicherheit

Wegen der Leistungseinschränkung der Sensoren und den Störungen in der Betriebsumgebung ist eine exakte Selbstlokalisierung anhand der ungenauen wahrgenommen Information schwer zu erreichen. In der Vergangenheit konnten sehr genaue Ergebnisse durch spezifische Methoden wie Feinmechanik[Krao4], sehr präzise Sensoren oder den Einsatz von Betriebs- und Kalibrier-Punkten erhalten werden. Dadurch wird die Berechnung der Unsicherheit vermieden, weil die Ergebnisse bereits genau genug sind, die Kosten sind allerdings sehr hoch und in manchen Applikationen lohnen sie sich nicht. Ein Alternative besteht darin, mehrere kostengünstigen Sensoren mit niedrigerer Auflösung zu verwenden und die gemessenen Informationen (einschließlich der Unsicherheit) aus allen Quellen durch Sensorfusionsverfahren zu kombinieren. Um eine genügende Genauigkeit der Positionsschätzung zu erreichen, wird die Berechnung der Unsicherheit sehr wichtig.

2.2.1. Repräsentation

Die räumliche Unsicherheit in Navigationsanwendungen wird typischerweise durch die numerischen Min-Max-Grenzen für die Fehler dargestellt[Tay76]. Eine probabilistische Darstellung der Positionsunsicherheit wurde später am mobilen Roboter HILARE verwendet [CL85]. Smith, Self und Cheeseman stellten die Unsicherheit jedes Grads an Freiheit in der räumlichen Beziehungen explizit dar [SSC86]. Diese Methode ist eine der am häufigsten verwendeten Darstellungen der räumlichen Unsicherheit, sie wird nun im Detail vorgestellt.

Eine räumliche Beziehung wird durch einen Vektor der räumlichen Variablen $\mathbf{x}$ repräsentiert. Ein exemplarischer Systemzustandsvektor besteht aus drei Variablen: seine Koordinaten x und y in einem zweidimensionalen kartesischen Bezugssystem sowie die Rotation ϕ um die z-Achse. Die drei Variablen beschreiben die Lage des Navigationssystems. Im Allgemeinen könnte ein Systemzustandsvektor beliebig viele räumliche Variablen enthalten:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Eine räumliche Unsicherheit kann durch eine Wahrscheinlichkeitsverteilung über die räumlichen Variablen dargestellt werden:

$$P(\mathbf{x}) = f(\mathbf{x})d\mathbf{x}$$

Die Wahrscheinlichkeitsverteilung der Unsicherheit könnte beliebig sein. Die meisten Messgeräte bieten nur einen nominellen Wert der gemessenen Variable, detaillierte Kenntnisse

über die Wahrscheinlichkeitsverteilung sind oft nicht explizit angegeben, der mittlere Fehler kann aber durch die Spezifikation des Sensors individuell geschätzt werden. Daher wird eine räumliche Unsicherheit durch den Mittelwert $\widehat{\mathbf{x}}$ und die Kovarianz $C(\mathbf{x})$ modelliert:

$$\begin{aligned} \widehat{\mathbf{x}} &\triangleq E(\mathbf{x}) \\ \widetilde{\mathbf{x}} &\triangleq \mathbf{x} - \widehat{\mathbf{x}} \\ C(\mathbf{x}) &\triangleq E(\widetilde{\mathbf{x}}\widetilde{\mathbf{x}}^{\tau}) \end{aligned}$$

wobei E die Erwartungsfunktion und $\widetilde{\mathbf{x}}$ die Abweichung vom Mittelwert ist.

Diese Modellierung wird dann in einem Filter verwendet um die Position des Navigationssystems zu bestimmen. Im Abschnitt 3.3 wird dies im Detail diskutiert. In manchen Applikationen ist es notwendig die Unsicherheit des Systems zu visualisieren. Durch den Mittelwert und die Kovarianz kann ein konzentrisches Ellipsoid gezeichnet werden [Nah76]. Das Ellipsoid stellt die Normalverteilung der Unsicherheit dar. Es ist wichtig zu betonen,dass die wirkliche Wahrscheinlichkeitsverteilung der Unsicherheit beliebig sein könnte. Nur wenn die Kontur einer bestimmten Unsicherheit gezeichnet werden muss, wird die Normalverteilung verwendet. Bei der Entwicklung des SLAM-Frameworks wurde dies berücksichtigt, jede Unsicherheit kann als ein Ellipsoid in einer 3D-Karte visualisiert werden.

2.3. Beispiel

Nun wird ein einfaches zweidimensionales Beispiel nach [SSC86] diskutiert um das SLAM-Problem und die räumliche Unsicherheit im System zu erklären. In diesem Beispiel führt das Navigationssystem die folgende Sequenz von Aktionen aus:

- Das Navigationssystem liegt in Startposition.
- Es misst den Abstand zum Objekt #1.
- Es bewegt sich.
- Es entdeckt ein neues Objekt #2 und misst den Abstand zu #2.
- Es misst das Objekt #1 vom neuen Standort aus.
- Die räumliche Beziehung zwischen Navigationssystem, Objekt #1 sowie Objekt #2 wird bestimmt.

Abbildung 2.1(a) zeigt die räumlichen Unsicherheiten in Weltreferenzkoordinaten sowie in Systemreferenzkoordinaten nach den ersten drei Aktionen. Zwei Unsicherheiten werden hier gezeichnet: die Lage des Objekts #1 und der Standort des Navigationssystems. Das Navigationssystem liegt zunächst in der Startposition, die in der Welt-Karte als Referenzstandort (ohne Unsicherheit) markiert wird, dann misst das Objekt #1, zeichnet die Position

des Objekts #1 auf und bewegt sich einige Meter weiter. Nachdem das System sich bewegt, ändern sich die räumlichen Unsicherheiten. Die Unsicherheit der Lage des Objekts #1 in System-Karte vergrößert sich wegen der Unsicherheit der Bewegung des Navigationssystems.

Nach der Bewegung des Navigationssystems wird das Objekt #2 von diesem neuen Standort aus entdeckt (Abbildung 2.1(b)). Das Navigationssystem misst und zeichnet die Entfernung zwischen den beiden auf. Weil das Objekt #2 und das Objekt #1 unabhängig voneinander sind, verändert sich die Unsicherheit der Lage des Objekts #1 nicht.

Danach versucht das Navigationssystem das Objekt #1 wieder zu finden, misst den Abstand zum Objekt #1 (Abbildung 2.1(c)). Diese neue Messung korrigiert die alte Positionsinformation des Objekts #1. Nach der Berechnung verringert sich die Unsicherheit der Lage des Objekts #1. Die Unsicherheit der Lage des Objekts #2 verändert sich nicht.

Das Ergebnis nach allen 6 Aktionen wird in der Abbildung 2.1(d) gezeigt. Durch Vergleich mit der Abbildung 2.1(a) wird erkannt, dass nach der zweiten Messung die Unsicherheit des Objekts #1 sich deutlich verringert, d.h. die Schätzung der räumlichem Beziehung wird zunehmend genauer.

2.4. Charakteristik des SLAM-Algorithmus

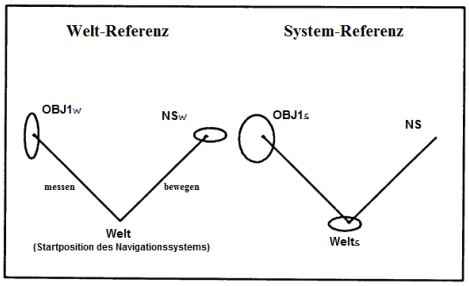
Der SLAM-Algorithmus wird im Folgenden anhand dreier Kriterien diskutiert: Konvergenz der Zustandsschätzung, Konsistenz des Schätzungsprozesses und Rechenaufwand bei der Aktualisierung der Zustandskovarianzmatrix.

2.4.1. Konvergenz

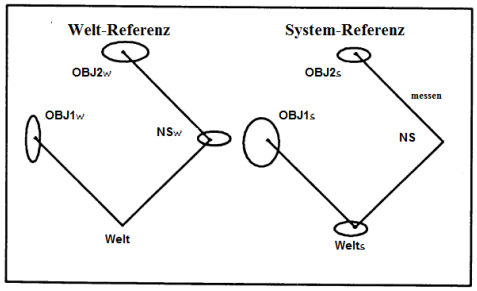
Während sich die Anzahl der Beobachtungen erhöht, wird die Unsicherheit der Karte langsam in einem begrenzten Umfang reduziert. Während sich die Abweichung verringert, wird die Schätzung des Systemzustands zunehmend genauer. Die Positionsbeziehung zwischen den in der Karte gespeicherten (Umgebungs-) Features verändert sich am Ende fast nicht mehr. Somit ist die Genauigkeit der Karte nur abhängig von der Genauigkeit der ersten Beobachtungsposition.

2.4.2. Konsistenz

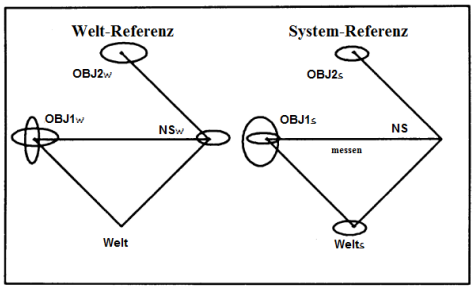
Um die Übereinstimmung der Schätzungen zu erhalten, ist die Aktualisierung der Zustandskovarianzmatrix notwendig. Die Zustandsschätzung basiert auf der Beobachtung der Umgebung, die Abweichung dieser Schätzung ist daher verknüpft mit der Abweichung der generierten Karte. Um auch ohne die absolute Positionsinformation zu vermeiden, dass die



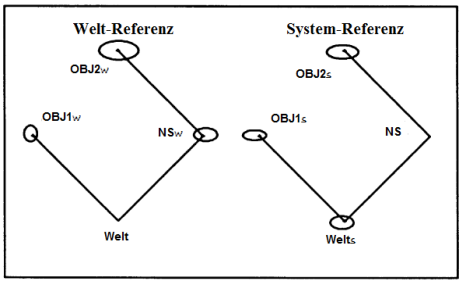
(a) Das Navigationssystem entdeckt das Objekt #1 und bewegt sich.



(b) Ein neues Objekt #2 wird entdeckt.



(c) Das Objekt #1 wird wieder gemessen.



(d) Das Navigationssystem, das Objekt #1 und das Objekt #2 werden lokalisiert werden

Abbildung 2.1.: Ein einfaches zweidimensionales Beispiel für das SLAM-Problem und die räumliche Unsicherheit. Quelle: [SSC86]

Abweichung der Karte immer größer wird, muss die Konsistenz der Zustandsschätzung gewährleistet sein.

2.4.3. Rechenaufwand

Eine wichtige Einschränkung für den SLAM-Algorithmus in der Large-Scale-Umgebung ist der große Rechenaufwand bei der Aktualisierung der Zustandskovarianzmatrix. Für ein mobiles Navigationssystem ist die Rechenleistungsfähigkeit des verwendeten Mikroprozessors begrenzt. Mit zehntausenden Features ist eine Aktualisierung der Zustandskovarianzmatrix schwierig durchzuführen. Eine Lösung zur effektiveren Berechnung muss noch gefunden werden.

3. Sensorfusion

Die traditionelle Signalerfassung wird oft durch einen einzigen Sensor vorgenommen, selbst beim Einsatz mehrerer (verschiedenartiger) Sensoren sind sämtliche Sensoren meist unabhängig voneinander. Aber in vielen Fällen müssen mehrere Signale aus verschiedenen Quellen gleichzeitig verarbeitet werden, um dadurch eine oder mehrere bestimmte Informationen zu errechnen, die direkt mittels Sensoren sehr schwierig oder unmöglich messbar sind. Multi-Sensoren sind zwar in der Lage, mehrere Signale zu erfassen, liefern allerdings teilweise auch redundante oder sogar widersprüchliche Informationen. Eine effektive und vernünftige Fusion der Signale aus mehreren verschiedenen Sensoren ist daher notwendig, um nur die mit der Umgebung konsistenten Ergebnisse zu erhalten. "Sensorfusion"stellt die Multi-Level-Verarbeitung der Multi-Sensor-Daten dar, d.h. Information aus mehreren Sensoren oder anderen Datenquellen werden kombiniert oder fusioniert, um eine bessere Schätzung zu erhalten.

In den letzten Jahren wurden die theoretische Grundlagen und Methoden der Sensorfusion erheblich verbessert. Die Sensorfusion bringt einige bedeutende Vorteile mit sich: gute Fehlertoleranz, hohe Genauigkeit, schnelle Verarbeitungsgeschwindigkeit, gute Komplementarität und große Informationsmenge.

3.1. Sensoren

In SLAM-System können Sensoren üblicherweise in zwei Kategorien unterteilt werden: interne Sensoren und externe Sensoren.

3.1.1. Interne Sensoren

Interne Sensoren nehmen die Bewegungsinformationen des Navigationssystems oder den Zustand der Komponenten innerhalb des Navigationssystem wahr. Sie sind unverzichtbar zur Regelung des Gesamtsystems. Die am häufigsten verwendeten inneren Sensoren sind Gyroskope, Beschleunigungsmesser und Odometer.

Gyroskope messen die Drehraten, Beschleunigungssensoren stellen die Beschleunigungen fest und Odometer erfassen die Anzahl der Radumdrehungen und berechnen damit die

zurückgelegten Distanzen. Die interne Sensoren schätzen die Lageposition des Navigationssystems in der Regel mithilfe der Bewegungsmodelle des Systems ab. Weil im Laufe der Zeit die Driftfehler von solchen Sensoren stetig größer werden, können sie nicht für die langfristige Positionierung eingesetzt werden. Diese Messprozesse können mit sehr hohen Abtastfrequenzen stattfinden und hängen nicht von den Umgebungsmerkmalen ab.

3.1.2. Externe Sensoren

Externe Sensoren werden verwendet, um den externen Zustand des Navigationssystems zu bestimmen. Sie ermöglichen dem Navigationssystem, die Arbeitsobjekte und Arbeitsumgebung zu erkennen und sich je nach Situation und Bedingung anzupassen. Damit werden die Anpassungsfähigkeit und die Intelligenz des Systems verbessert.

GPS

Der bekannteste externe Sensor ist das *Global-Positioning-System* (GPS). Es ist ein globales Navigationssatellitensystem zur Positionsbestimmung und Zeitmessung. Durch die bekannten Positionen von mindesten 3 GPS-Satelliten wird die aktuelle Standortinformation des GPS-Empfänger schnell berechnet. Die Genauigkeit lässt sich durch Differenzmethoden auf Werte im Zentimeterbereich steigern.

Vorteile:
Das GPS-System deckt 98% der Welt ab. Mit der Weltkarte kann das Navigationssystem schnell lokalisiert werden. Im Vergleich zu der relativen Lokalisierung mit anderen externen Sensoren ist solche absolute Lokalisierung in manchen Applikationen vorteilhaft.

Nachteile:
Die GPS-Lokalisierung kann nur in den Outdoor-Umgebungen verwendet werden, in einem Gebäude sind Signale kaum zu empfangen, der Anwendungsbereich wird damit beschränkt.

Laserentfernungsmesser

Laserentfernungsmesser (*Laser rangefinder*) erhalten die Zielentfernung durch Einsatz von Laserimpulsen. Entfernungsmessung beruht in der Regel auf der *Time of Flight*(TOF)- oder der Phasenverschiebungs- (*Phase-Shift*)-Technik. Bei der TOF-basierten Entfernungsmessung werden der Zeitpunkt zum Senden eines kurzes Laserpulses sowie der Zeitpunkt zum Empfangen des zurückkehrenden Laserpulses erfasst und aus der Länge des Zeitintervalls der zurückgelegte Weg bestimmt. Beim Phasenverschiebungsverfahren ergibt sich die Entfernung durch den Vergleich der Phasenverschiebung zwischen ausgesandtem und empfangenem Laserpuls [HKP90, BJP93].

Vorteile:
Der Messvorgang eines Laserentfernungsmessers ist schnell und das Ergebnis ist präzise. Im Vergleich zum Sonar-Sensor hat der Laserentfernungsmesser eine höhere Winkelauflösung. Der Laserstrahl ist schmal und der messbare Abstand ist deutlich länger als bei Sonar-Sensoren.

Nachteile:
Der Preis eines Laserentfernungsmessers ist vergleichsweise hoch, und aufgrund möglicher Spiegelreflexionen und diffusem Licht wird die Messung von den Materialeigenschaften der Oberfläche des Zielobjektes beeinflusst.

Millimeterwellen-Radar

Millimeterwellen-Radar-Geräte (*millimeter wave radar, MMWR*) messen die Entfernung zum Zielobjekt durch Senden und Empfangen von elektromagnetischen Wellen. Die Größe der Radar-Antenne hängt von der verwendeten Frequenz ab. Je höher die Frequenz ist, desto kleiner ist die Antenne. Meist werden kurze Pulse von elektromagnetischen Wellen verwendet.

Vorteile:
Millimeterwellen-Radar kann unter allen Wetterbedingungen verwendet werden. Der messbare Abstand ist groß und die Messung ist präzise.

Nachteile:
Ein Millimeterwellen-Radar ist vergleichsweise teuer und das Gerät ist meist größer als die anderen Sensoren. Unter den gleichen Wetterbedingungen verfällt das Signal schneller als beim Laserentfernungsmesser.

Sonar-Sensor

Sonar-Systeme werden bei der Navigation oft als Hauptsensoren verwendet [KK94, HKoo] . Die bekannte Ausbreitungsgeschwindigkeit des Ultraschalls ist die Grundlage der Ultraschall-Entfernungsmessung. Der von dem Ultraschallschwinger generierte Schallimpuls breitet sich durch die Luft in Richtung des Zielobjektes aus, nach der Reflexion am Ziel wird das Echo-Signal empfangen. Durch die Zeitdauer T und die bekannte Ausbreitungsgeschwindigkeit V des Ultraschalls in Luft kann der Abstand L zwischen dem Sensor und dem Zielobjekt berechnet werden: $L = V * T/2$.

Vorteile:
Sonar-Sensoren sind im Vergleich zum Laserentfernungsmesser und zum Millimeterwellen-Radar kostengünstiger, und verfügen trotzdem über große Erfassungsbereiche.

Nachteile:
Die geringe Winkelauflösung und das unpräzise Messergebnis von Sonar-Sensoren sind nachteilig. Je nach Umgebungsbedingungen ist es außerdem möglich, falsche oder doppelte Echo-Signale zu empfangen.

Stereo-Vision-System

Ein Stereo-Vision-System erhält die Tiefeninformation eines Objekts mit Hilfe einer einzelnen (mobilen Kamera mit verschieden Betrachtungswinkeln) oder mehreren Kameras. Die Qualität der gewonnenen Tiefkarte hängt von der Kalibrierung der Kamera und des Lichtverhältnisses ab. Solche visuellen Sensoren wurden bereits in Navigationssystemen, insbesondere bei Robotern, z.B. in [TK95] und [MJ97] eingesetzt.

Vorteile:
Ein Stereo-Vision-System liefert ausgeprägte und reiche visuelle Informationen über die Umgebung, welche die andere Sensoren schwierig oder unmöglich erfassen können.

Nachteile:
Die Menge der Verarbeitungsdaten ist relativ groß, ebenso ist die Lichtsituation relevant für Bildverarbeitung, unter Einfluss einer schlechten Beleuchtung können fehlerhafte Messergebnisse erhalten werden.

Im Allgemeinen ist die Zuverlässigkeit eines einzelnen Sensors schlechter als die Kombination von mehreren Sensoren. Bei der Auswahl der Sensoren sollten sowohl Vor- als auch Nachteile der einzelnen Sensoren in Betracht gezogen werden.

3.2. Architektur

Die Architektur der Sensorfusion stellt die logische und formale Kombination der Daten aus verschieden Sensoren dar. Waltz und Llinas [WL90] beschreiben eine Reihe von Architekturen, von denen im Folgenden die parallele und die sequentielle Architektur genauer beschrieben werden.

3.2.1. Parallele Fusion

Bei der parallelen Fusion werden alle Messwerte gleichzeitig in einem Schritt verarbeitet. Die Abbildung 3.1 stellt eine parallele Architektur dar. Die parallele Fusion hat niedrige Komplexität und ermöglicht somit die Anwendung auch auf schwach leistungsfähigen Systemen. Die Synchronisierung und Verfügbarkeit aller Sensoren wird aber vorausgesetzt.

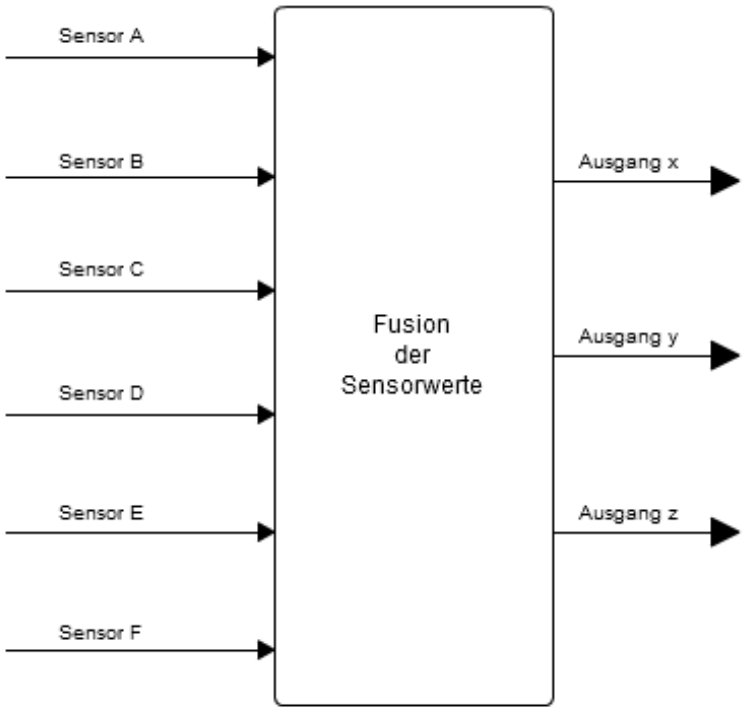


Abbildung 3.1.: Parallele Fusion

3.2.2. Sequentielle Fusion

Bei der sequentiellen Fusion werden Sensordaten in mehreren Schritten verarbeitet. Die Messwerte von einigen Sensoren werden zuerst in einem Schritt fusioniert, daraus erhaltene Ergebnisse werden dann mit weiteren Messwerten kombiniert und im nächsten Schritt erneut fusioniert usw., bis schließlich alle Sensordaten verarbeitet und die Zielergebnisse erhalten wurden. Die Abbildung 3.2 stellt eine sequentielle Architektur dar. Ein erheblicher Vorteil dieser Architektur ist, dass die Synchronisierung aller Sensoren nicht erforderlich ist. So wird ermöglicht, dass die niedrigen Fusionsschichten bei geringer Komplexität mit hohen Signal- und Zyklusraten arbeiten können, während die höheren Fusionsschichten, die in der Regel höhere Komplexität aufweisen, bei geringeren Geschwindigkeiten arbeiten. Hierdurch

kann die Integration von Sensoren verschiedener Signalraten deutlich vereinfacht werden [Roso6].

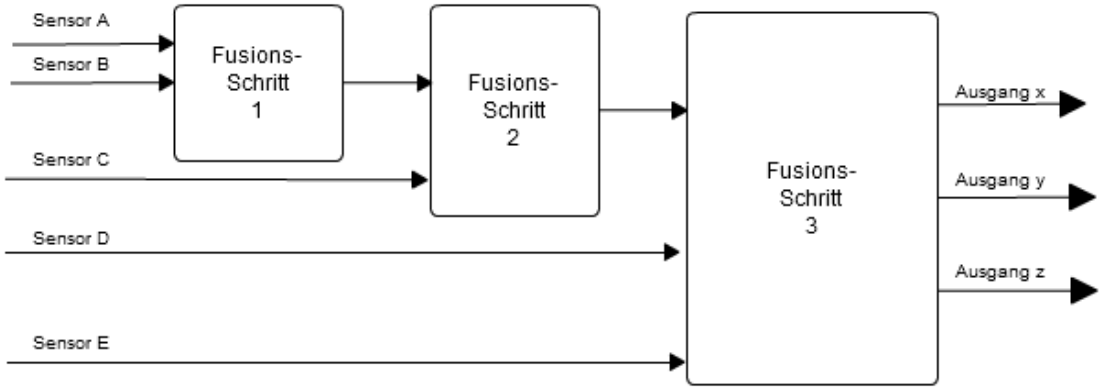


Abbildung 3.2.: Sequentielle Fusion

3.2.3. Mischformen

Basierend auf der parallelen und sequentiellen Fusion ergeben sich verschiedene Mischformen, die beide Konzepte vereinen. Die Abbildung 3.3 stellt eine exemplarische Architektur dar. Diese Modularisierung bringt große Freiheit, ein Modul einzeln einzustellen und anzupassen. Beim Austauschen oder Verändern eines Sensors muss nicht das Gesamtsystem, sonder nur der entsprechende Funktionsblock abgeändert werden.

Aufgrund der objektorientierten Implementierung können alle drei obigen Fusionsarchitekturen mit dem in dieser Arbeit entwickelten SLAM-Framework realisiert werden.

3.3. Filter

Im Jahre 1960 wurde eine berühmte Arbeit von Kalman veröffentlicht [Kal6o]. Diese Arbeit beschreibt eine rekursive Methode zur Lösung eines linearen Filtering-Problems in diskreten Daten. Das war der Prototyp des Kalman-Filters. Kurz gesagt, der Kalman-Filter besteht aus einer Reihe von mathematischen Gleichungen, durch welche Kalman eine effektive Lösung für das Problem der Schätzung des Systemzustandes fand. Ein Kalman-Filter kann nicht nur den vergangenen und gegenwärtigen Zustand schätzen, sondern auch den zukünftigen vorhersagen. Es spielt eine sehr wichtige Rolle im Bereich der autonomen Navigation [KB61] [RBoo]. Aufgrund der kontinuierlichen Weiterentwicklung, insbesondere nach Einführung des

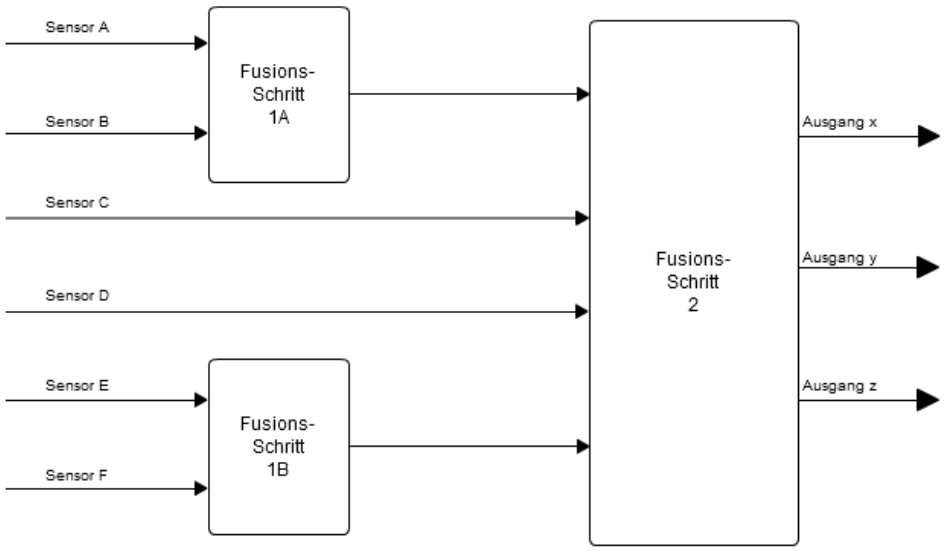


Abbildung 3.3.: Kombination aus paralleler und sequentieller Sensorfusion

erweiterten Kalman-Filters zur Lösung von nichtlinearen Problemen, finden Kalman-Filter in verschiedenen Bereich zunehmend mehr Anwendungen.

3.3.1. Linearer Kalman-Filter

Ein lineares System wird durch eine Systemgleichung und eine Messgleichung beschrieben [WBo6] :

$$\begin{aligned} x_k &= Ax_{k-1} + Bu_{k-1} + w_{k-1},\ x \in \mathbb{R}^n \\ z_k &= Hx_k + v_k,\ z \in \mathbb{R}^m \end{aligned}$$

Das Systemrauschen w und das Messrauschen v sind stets unabhängig voneinander. Sie folgen dabei einer Normalverteilung mit dem Mittelwert 0 und der Kovarianz Q_k und R_k, in üblicher Kurznotation:

$$\begin{aligned} p(w_k) &\sim N(0, Q_k) \\ p(v_k) &\sim N(0, R_k) \end{aligned}$$

Der Zustand des Systems ist durch x gekennzeichnet, der Eingang durch u und die Übergänge zwischen zeitlich aufeinanderfolgenden Zuständen werden durch die Matrix A beschrieben. Der Einfluss des Einganges u wirkt durch die Matrix B. Der Ausgang wird

durch z gekennzeichnet. Die Matrix H beschreibt die Beziehung zwischen Systemzustand und Beobachtung.

Es gibt zwei sich wiederholende Arbeitsschritte im Kalman Filter:

1. Schritt: "Kalman-Time-Update"

$$\begin{aligned} \widehat{x}_{k|k-1} &= A\widehat{x}_{k-1} + Bu_{k-1} \\ P_{k|k-1} &= AP_{k-1}A^{\tau} + Q_{k-1} \end{aligned}$$

2. Schritt: "Kalman-Measurement-Update"

$$\begin{aligned} K_k &= P_{k|k-1}H^{\tau}(HP_{k|k-1}H^{\tau} + R)^{-1} \\ \widehat{x}_k &= \widehat{x}_{k|k-1} + K_k(z_k - H\widehat{x}_{k|k-1}) \\ P_k &= (I - K_kH)P_{k|k-1} \end{aligned}$$

In der Phase der Prädiktion schätzt der Kalman-Filter den aktuellen Systemzustand anhand des vorherigen Zustands. Beim zweiten Schritt des Filtervorgangs wird die Vorhersage schließlich mit den neuen Informationen des aktuellen Messwerts korrigiert. Die zwei Schritte werden wiederholend durchgeführt bis die gesuchte Schätzung geliefert wird. Die Abbildung 3.4 zeigt die Struktur des Kalman-Filters.

Für den Kalman-Filter und die Matrizen Q und R, die die Intensität des Systemrauschens und des Messrauchens angeben, gilt:

$$\begin{aligned} E[w_k] &= 0 \\ E[w_k w_k^{\tau}] &= Q_k \\ E[v_k] &= 0 \\ E[v_k v_k^{\tau}] &= R_k \\ E[w_k v_k] &= 0 \\ P &= E[(x_k - \widehat{x}_k)(x_k - \widehat{x}_k)^{\tau}] \end{aligned}$$

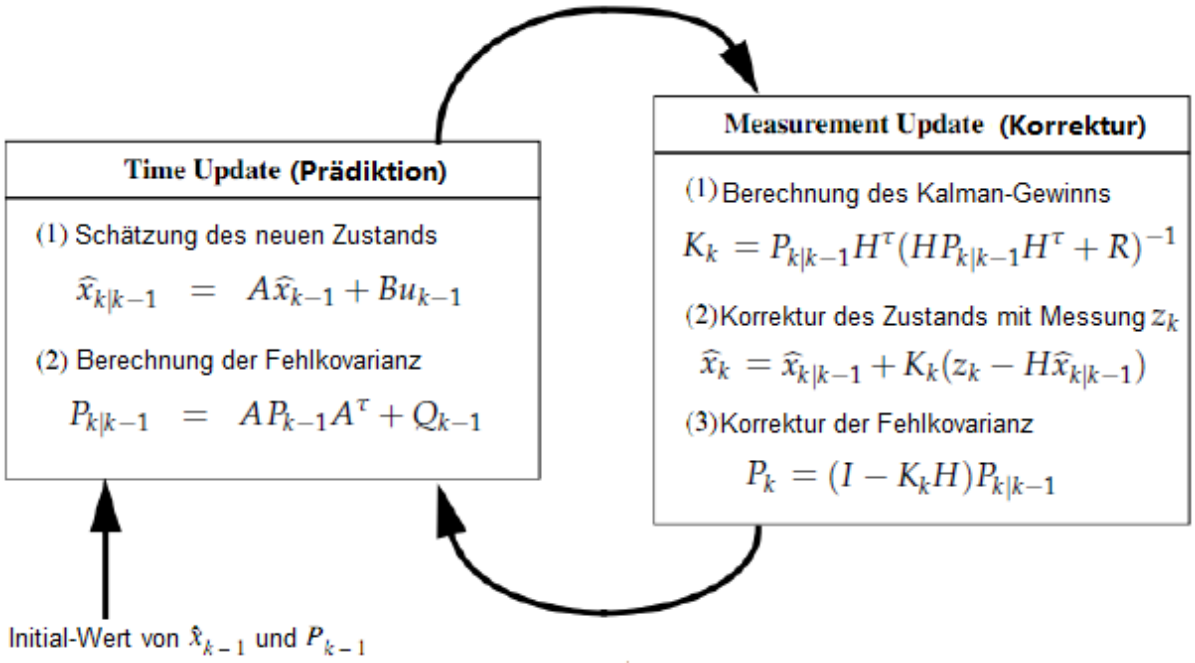


Abbildung 3.4.: Der laufende Zyklus eines diskreten Kalman-Filters. Quelle:[WBo6]

3.3.2. Erweiterter Kalman-Filter

Lineare Kalman-Filter lösen lineare Probleme. In der realen Welt werden aber meist nichtlineare Systeme angetroffen. Für diese Anwendungen wurde der erweiterte Kalman-Filter entwickelt. Wie in einem linearen Kalman-Filter wird angenommen, dass der Zustandsvektor $x \in \mathbb{R}^n$ und der Ausgangsvektor $z \in \mathbb{R}^m$ sind, jedoch werden die Systemgleichung und Messgleichung in nichtlineare Ausdrücke abgeändert:

$$\begin{aligned} x_k &= f(x_{k-1}, u_{k-1}, w_{k-1}) \\ z_k &= h(x_k, v_k) \end{aligned}$$

wobei w_k und v_k keine weißen Rauschsignale mehr sind. Das Rauschen ist schwierig durch Messung zu erfassen. Wird das Rauschen zunächst ignoriert, ergeben sich die folgenden Annäherungen:

$$\begin{aligned} \widetilde{x}_k &= f(x_{k-1}, u_{k-1}, 0) \\ \widetilde{z}_k &= h(x_k, 0) \end{aligned}$$

Somit können die beide Modelle umgeschrieben werden:

$$\begin{aligned} x_k &\approx \widetilde{x}_k + A(x_{k-1} - \widehat{x}_{k-1}) + Ww_{k-1} \\ z_k &\approx \widetilde{z}_k + H(x_k - \widetilde{x}_k) + Vv_{k-1} \end{aligned}$$

wobei A, W, H, W Jacobi-Matrizen sind:

$$\begin{aligned} A_{[i,j]} &= \frac{\partial f_{[i]}}{\partial x_{[j]}}(\widehat{x}_{k-1}, u_{k-1}, 0) \\ W_{[i,j]} &= \frac{\partial f_{[i]}}{\partial w_{[j]}}(\widehat{x}_{k-1}, u_{k-1}, 0) \\ H_{[i,j]} &= \frac{\partial h_{[i]}}{\partial x_{[j]}}(\widetilde{x}_k, 0) \\ V_{[i,j]} &= \frac{\partial f_{[i]}}{\partial v_{[j]}}(\widetilde{x}_k, 0) \end{aligned}$$

Analog zum linearen Kalman-Filter existieren auch hier zwei Arbeitsschritte:

1. Schritt: ”Kalman-Time-Update”

$$\begin{aligned} \widehat{x}_{k|k-1} &= f(x_{k-1} + u_k, 0) \\ P_{k|k-1} &= F_k P_{k-1} F_k^\tau + Q_k \end{aligned}$$

2. Schritt: ”Kalman-Measurement-Update”

$$\begin{aligned} K_k &= P_{k-1} H^\tau (H P_{k|k-1} H^\tau + R)^{-1} \\ \widehat{x}_k &= \widehat{x}_{k|k-1} + K_k(z_k - h(\widehat{x}_{k|k-1}, 0)) \\ P_k &= (I - K_k H) P_{k|k-1} \end{aligned}$$

Nun werden alle nichtlinearen Terme linearisiert, um den Kalman-Filter auf nichtlineare Probleme angewenden zu können. Eine weitere Verbesserung der Zustandsschätzung kann nach [HMo4] erreicht werden, indem auf den Filter eine Vorwärts- Rückwärtsfilterung bzw. Smoothing angewendet wird. In der vorliegenden Arbeit wird diese Methode nicht im Detail erläutert.

4. Karten

Die Karte ist eine wichtiger Bestandteil eines SLAM-Systems. Durch Verarbeitung der Informationen über die Umgebung aus den Sensoren wird die reale Welt modelliert und eine oder mehrere Karten automatisch erstellt. Die resultierenden Karten werden dann wiederum verwendet, um die Umgebung für die Wegplanung und Navigation darzustellen und die Position in der Umgebung zu ermitteln.

4.1. Kartenrepräsentation

Je nach Darstellungsweise können Karten hauptsächlich in drei Kategorien unterteilt werden: Rasterkarten [Elf90, Thro1] , Feature-Karten [CK92, LF99] sowie topologische Karten [BNRW99, KB91] .

4.1.1. Rasterkarten

Der Begriff der Rasterkarte wurde zuerst von Elfes [Elf89] und Moravec [ME85] vorgeschlagen.

In einer 2D-Rasterkarte wird die Realwelt gleichmäßig in uniformen Rasterzellen abgebildet. Jede Zelle speichert bestimmte Informationen über eine Position. Im einfachsten Fall wird ein Wert von 0 bis 1 zugeordnet: 0 für frei und 1 für besetzt. Ein weiterer häufig benutzter Wert ist die Wahrscheinlichkeit, dass diese Position belegt ist. Die Abbildung 4.1 stellt eine exemplarische 2D-Rasterkarte dar. Für eine 3D-Umgebung werden zusätzlich entweder die Höheninformation notiert, oder aber direkt 3D-Raster verwendet.

4.1.2. Feature-Karten

Eine Feature-Karte beschreibt die Umgebung als eine Reihe von Features. Durch Verarbeitung der Sensordaten werden Features wie Punkte, Linien und Flächen detektiert und dann in der Karte gespeichert. Zum Erhalten der visuellen Information über die Umgebung werden normalerweise ein oder mehrere Kameras verwendet, um mithilfe von Bildverarbeitungsprozessen Features zu finden. Außerdem werden oft Laserentfernungsmesser eingesetzt, um in Punktwolken mithilfe der Regressionsanalyse Features zu finden. Für eine strukturierte Umgebung ist die Verwendung einer Feature-Karte vorteilhaft. Zum Beispiel können für

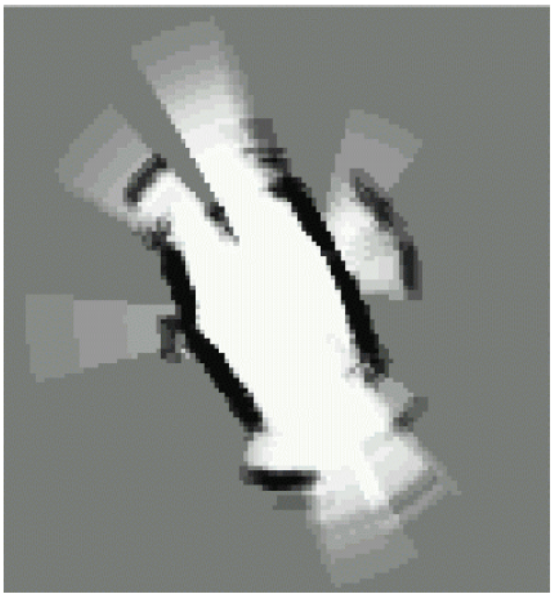


Abbildung 4.1.: Eine exemplarische 2D-Rasterkarte. Quelle: [Thro1]

eine Indoor-Umgebung die Innenwände als Linien und die Ecken als Punkte in der Karte gespeichert werden. Die Abbildung 4.2 stellt eine exemplarische Feature-Karte dar.

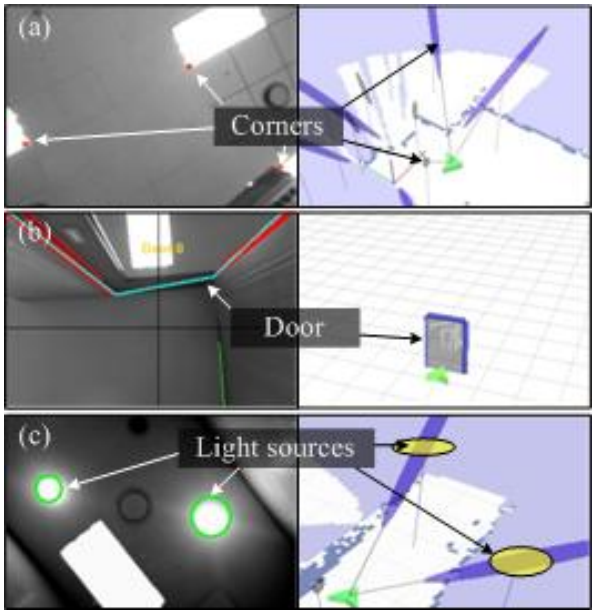


Abbildung 4.2.: Eine exemplarische Feature-Karte. Quelle: [HSo8]

4.1.3. Topologische Karten

Eine topologischen Karte betrachtet nur die topologische Information der Umgebung. Sie ist in der Regel in Form eines Graphen dargestellt. Die Knoten des Graphen bezeichnen bestimmte Orte, die Kanten dazwischen bezeichnen die Pfadinformation zwischen beiden Orten. Die Abbildung 4.3 stellt eine exemplarische topologische Karte dar. Für eine strukturierte Umgebung ist die topologische Karte sehr effektiv. Jedoch ist in einer unstrukturierten Umgebung die Identifizierung eines Standortes komplex, da in diesen Fall die topologischen Informationen allein nicht für Lokalisierung und Navigation ausreichen [KW94] .

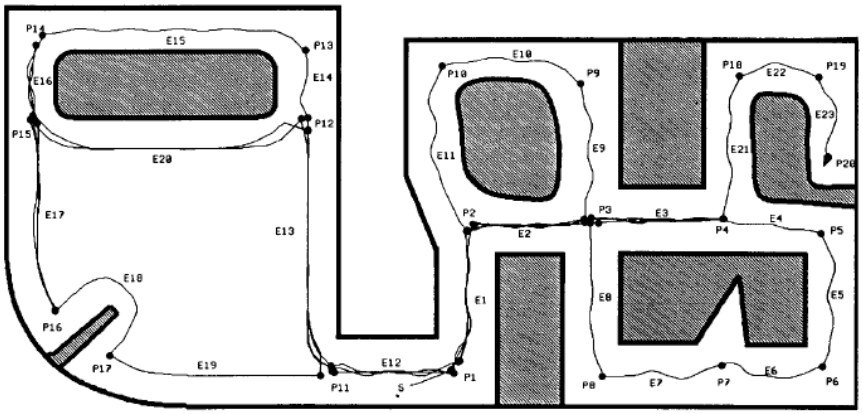


Abbildung 4.3.: Eine exemplarische topologische Karte. Quelle: [KB91]

4.1.4. Vergleich

Rasterkarten sind robust und leicht zu implementieren, allerdings nicht gut skalierbar. Die Rasterstruktur benötigt ziemlich viel Speicherplatz, der Umfang der Karte ist daher begrenzt.

Feature-Karten sind robust und kompakt. Sie stellen eine große Anforderung an den nötigen Datenverarbeitungsprozesse und eignen sich nicht für eine zu komplexe Umgebung.

Topologie-Karten hängen nur von den bestimmten Orten in der Umgebung ab, sie sind ebenfalls kompakt und effektiv. Aber wie Feature-Karten eignen sie sich nur für eine strukturierte Umgebung.

Um eine bessere Modellierung der Umgebung zu erreichen, können gemischte Darstellungen eingesetzt werden [SD98] [Thr98] .

4.2. Unsicherheit in Karte

Wie in dem Abschnitt 2.2 diskutiert wird, ist die Berechnung der Unsicherheit im SLAM-Problem sehr wichtig. Beim Implementieren der Karte in SLAM-Framework wurde berücksichtigt, wie die Information über Unsicherheit in der Karte zu speichern und zu repräsentieren.

Am üblichsten besitzen die gemessene Punkte die Positionsunsicherheit. Obwohl die Rasterkarte intuitiv und leicht zu implementieren ist, kann sie die Positionsunsicherheit nicht direkt speichern. In einer Rasterkarte ist die Position jeder Zelle bestimmt, die gespeicherte Wahrscheinlichkeitsfunktion in einer Zelle zeigt die Wahrscheinlichkeit, dass diese Position als Hindernis belegt ist. Aus diesem Grund ist die Rasterkarte nicht geeignet für das implementierte SLAM-Framework.

In einer Karten-basierten SLAM-Applikation wird nicht nur die Positionsunsicherheit eines Punktes sondern auch die Unsicherheit der Lage eines Objekts häufig diskutiert. Für ein 3D-Objekt kann die Unsicherheit der Lage in verschiedene Freiheitsgrade unterteilt werden. Grundlegende Unsicherheiten sind die Unsicherheit der Rotation, die Unsicherheit der Translation sowie die Unsicherheit der Skalierung. Die topologische Karte kennt kein Konzept für Objekte in der Umgebung, während dies in einer Feature-Karte leicht realisiert werden kann. Daher wurde für die Entwicklung des SLAM-Frameworks eine Feature-Karte verwendet.

4.3. Szenengraph

Um diese Anforderung an die Darstellung der Unsicherheit der Lage in verschiedenen Freiheitsgraden zu erfüllen wurde hier ein Szenengraph verwendet.

Ein Szenengraph ist eine objektorientierte Datenstruktur, mit der die logische und die räumliche Anordnung der darzustellenden zwei- oder dreidimensionalen Szene beschrieben wird. Er wird häufig bei der Entwicklung computergrafischer Anwendungen eingesetzt.

Im Szenengraph kann ein Knoten viele Kinder aber oft nur einen einzigen Mutterknoten haben. Ein Effekt am Mutterknoten wirkt automatisch an alle untergeordneten Knoten. Jeder Knoten des Szenengraphen hat üblicherweise eine Transformationsmatrix um die Operationen (Translation, Rotation, Spiegelung, Skalierung und Scherung) in einer effizienten Weise zu realisieren. Die entsprechende Unsicherheit kann dabei im gleichen Knoten gespeichert werden. Diese hierarchische Modellierung vereinfacht den Aufbau und das Manipulieren einer Szene deutlich. Man muss nicht jedes Einzelteil eines Objektes einzeln transformieren, sondern transformiert einfach die Gesamtheit aller Einzelteile. Aus diesen Gründen wurde im implementierten SLAM-Framework ein Szenengraph verwendet. Die Implementierung und Erweiterung werden im Abschnitt 5.4 vorgestellt.

5. Implementierung

5.1. Architektur

Die modulare Architektur erhöht die Erweiterbarkeit des SLAM-Frameworks. Mit den gewonnenen Erkenntnissen aus Kapitel 2 wird das implementierte SLAM-Frameworks in 5 Module unterteilt:

Sensoren sammeln die Information über das Navigationssystem und die Umgebung.

Modelle beschreiben die Systemgleichung und Messgleichung.

Filter schätzt und korrigiert den Systemzustand.

Karte speichert die Information über die Umgebung.

SLAM-Kern steuert alle Module.

Die Abbildung 5.1 zeigt die Architektur des implementierten SLAM-Frameworks.

Aufgrund der Vielfalt der Sensoren und der individuellen Hardware-Treiber der einzelnen Sensoren wurde das Sensor-Modul bisher noch nicht implementiert. Im Framework existiert keine Basisklasse, die alle benötigten Grundfunktionen der Sensoren definiert. In den Demo-Programmen wird das Eingeben der Sensordaten durch Simulation realisiert.

Für unterschiedliche Probleme muss der SLAM-Kern entsprechend angepasst werden. Es gibt keine einheitliche Vorlage. Dieser Teil wird erst implementiert, wenn eine praktische SLAM-Anwendung basierend auf diesem SLAM-Framework zu entwickeln ist.

Alle andere drei übrigen Module wurden jedoch im SLAM-Framework implementiert.

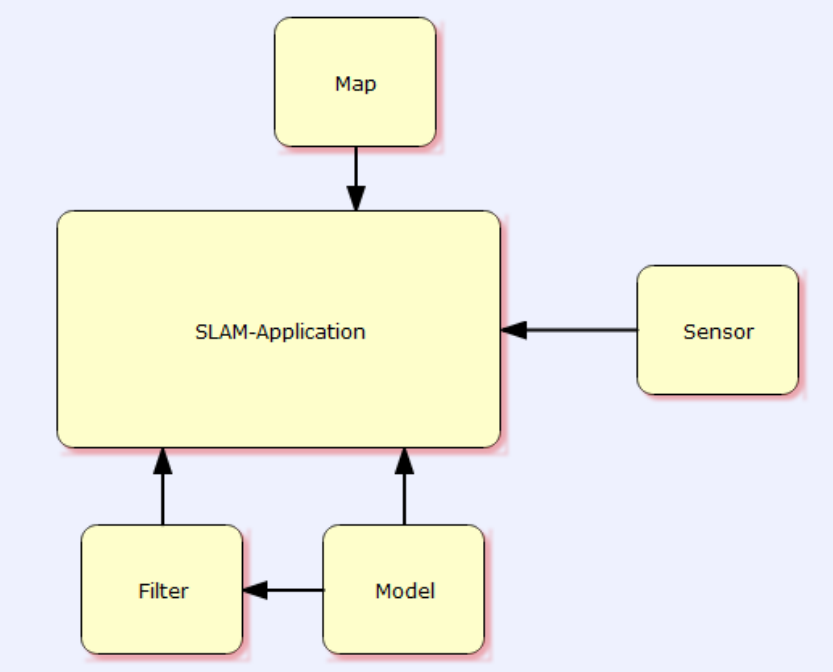


Abbildung 5.1.: Blockschaltbild eines SLAM-Systems

5.2. CMake-configure

Für ein Framework ist die Cross-Plattform-Tauglichkeit eine grundlegende und wichtige Eigenschaft. Bei dieser Entwicklung wurde die Software ”CMake” eingesetzt. CMake ist ein quelloffenes, plattformunabhängiges Build-System. Mit CMake wird die Make-Datei für das entsprechenden Betriebssystem automatisch generiert. Die Datei CMakeLists.txt und alle *.cmake-Dateien unter dem Verzeichnis ”config” dienen zur Konfiguration des CMake-Systems. Bisher wurde das SLAM-Framework bereits unter Linux und Windows getestet, es funktioniert reibungslos auf beiden Plattformen. Eine detaillierte Installationsanleitung für das SLAM-Framework wird als Anhang A beigefügt.

5.3. Filter

Im Filter-Modul sollen alle Filter zur Sensorfusion zur Verfügung gestellt werden, beispielsweise die Kalmanfilter, die im Kapitel 3 vorgestellt werden. Filter dienen zur Schätzung und Korrektur des Systemzustands. Dies wurde durch den Einsatz der externen Bibliothek BFL realisiert.

5.3.1. BFL

Bayesian Filtering Library (BFL) is eine quelloffene C++-Bibliothek und bietet eine große Palette an Filtern. Diese verarbeiten Daten rekursiv und ihr Schätzalgorithmus basiert auf der Bayes-Regel, wie beispielsweise beim (erweiterten) Kalman-Filter und Partikel-Filter. In BFL stehen mehr als 10 Bayes-Filter zur Verfügung, alle Filter sind aus einer Basisklasse abgeleitet und nutzen einheitliche Schnittstellen bei der Aktualisierung des Systemzustands.

5.3.2. Filter-Generator

Wegen der hohen Komplexität der Schnittstellen wird die BFL-Bibliothek nicht direkt, sonder über einen Wrapper im SLAM-Framework verwendet. Eine wichtige Klasse im Verzeichnis "src\filter" ist der Filter-Generator. Diese Klasse beinhaltet nur die Create**Filter()-Funktionen, dient zur Zentralverwaltung aller Filter und bietet dem Benutzer einen Überblick, welche Filter zur Verfügung stehen.

5.3.3. Modell

Wie in dem Abschnitt 3.3 diskutiert wird, wird ein Navigationssystem durch eine Systemgleichung und eine Messgleichung beschrieben. Insgesamt wurden im implementierten SLAM-Framework fünf verschiedenen Systemmodelle definiert, zwei davon sind lineare Modelle und andere drei sind nichtlinear:

$$
\begin{aligned}
x_k &= Ax_{k-1} + Bu_{k-1} + w_{k-1},\ p(w_k) \sim N(0, Q_k) \\
x_k &= Ax_{k-1} + Bu_{k-1} + w_{k-1},\ \text{w beliebig} \\
x_k &= f(x_{k-1}, u_{k-1}) + w_{k-1},\ p(w_k) \sim N(0, Q_k) \\
x_k &= f(x_{k-1}, u_{k-1}) + w_{k-1},\ \text{w beliebig} \\
x_k &= f(x_{k-1}, u_{k-1}, w_{k-1}),\ \text{w beliebig}
\end{aligned}
$$

Analog existieren fünf Messmodelle:

$$
\begin{aligned}
z_k &= Hx_k + v_k,\ p(v_k) \sim N(0, R_k) \\
z_k &= Hx_k + v_k,\ \text{v beliebig} \\
z_k &= h(x_k) + v_k,\ p(v_k) \sim N(0, R_k) \\
z_k &= h(x_k) + v_k,\ \text{v beliebig} \\
z_k &= h(x_k, v_k),\ \text{v beliebig}
\end{aligned}
$$

Model-Generator

Unter dem Verzeichnis "src\model" steht weiterhin ein Model-Generator zur Verfügung. Analog zum Filter-Generator enthält er nur die Create**Model()-Funktionen und dient zur Zentralverwaltung aller System- und Mess-Modelle.

5.4. Karte

Wie in dem Abschnitt 4.2 diskutiert wird, wurde eine Feature-Karte mithilfe der externen Bibliothek OSG im SLAM-Framework realisiert.

5.4.1. OSG

OpenSceneGraph (OSG) ist ein quelloffenes Szenengraphsystem. Es findet in Flugzeugsimulation, Spielen, virtueller Realität sowie wissenschaftlicher Visualisierung Anwendung. OSG basiert auf dem Konzept des Szenengraphen und bietet ein objektorientiertes Grafik-Framework, basierend auf der OpenGL-Bibliothek. OSG kann unter verschiedenen Betriebssystemen eingesetzt werden. Das SLAM-Framework verwendet einen Teil von OSG, um die Umgebungsinformation in einem Szenengraph zu speichern.

5.4.2. Klasse "map"

Die Klasse "map" ist die Hauptklasse in diesem Modul. Alle Operationen im Zusammenhang mit der Handhabung der Karte wurden in dieser Klasse implementiert, wie z.B.
das Einfügen von Objekten in die Karte:
registerPoint() - einen Punkt in die Karte einfügen;
registerLine() - eine Linie in die Karte einfügen;
registerPolygon() - ein Polygon in die Karte einfügen;
...

das Auffinden von Objekten in der Karte:
getObjectList() - alle Objekte in der Karte zurückgeben
findObjectById() - das Objekt mit dem spezifischen Name finden.
findObjectByName() - das Objekt mit dem spezifischen ID finden.
getNumberOfObjects() - Objekte zählen
...

Import und Export zu Datendateien:
readRootFromFile() und exportMapToFile()
...

5.4.3. Erweiterung

Wie in dem Abschnitt 4.2 diskutiert wird, sollen zwei Type von Unsicherheiten in der Karte implementiert werden: die Positionsunsicherheit eines Punktes sowie die Unsicherheit der Lage eines Objekts. In OSG wird die Unsicherheit überhaupt nicht berücksichtigt. Die Klassen müssen erweitert werden. Um die Positionsunsicherheit eines Punktes zu speichern wird eine Klasse SLAM::Vec3Extended von der Klasse osg::Vec3 abgeleitet, Unsicherheiten der drei Koordinaten x, y, z werden getrennt gespeichert. Um die Unsicherheit der Lage eines Objekts darzustellen wird eine neue Klasse SLAM::TransformUncertrainty implementiert. In dieser Klasse werden Unsicherheiten der Transformationen - Rotation, Translation sowie Skalierung gespeichert. Dies wird durch ein UML-Diagramm (Abbildung 5.2) veranschaulicht.

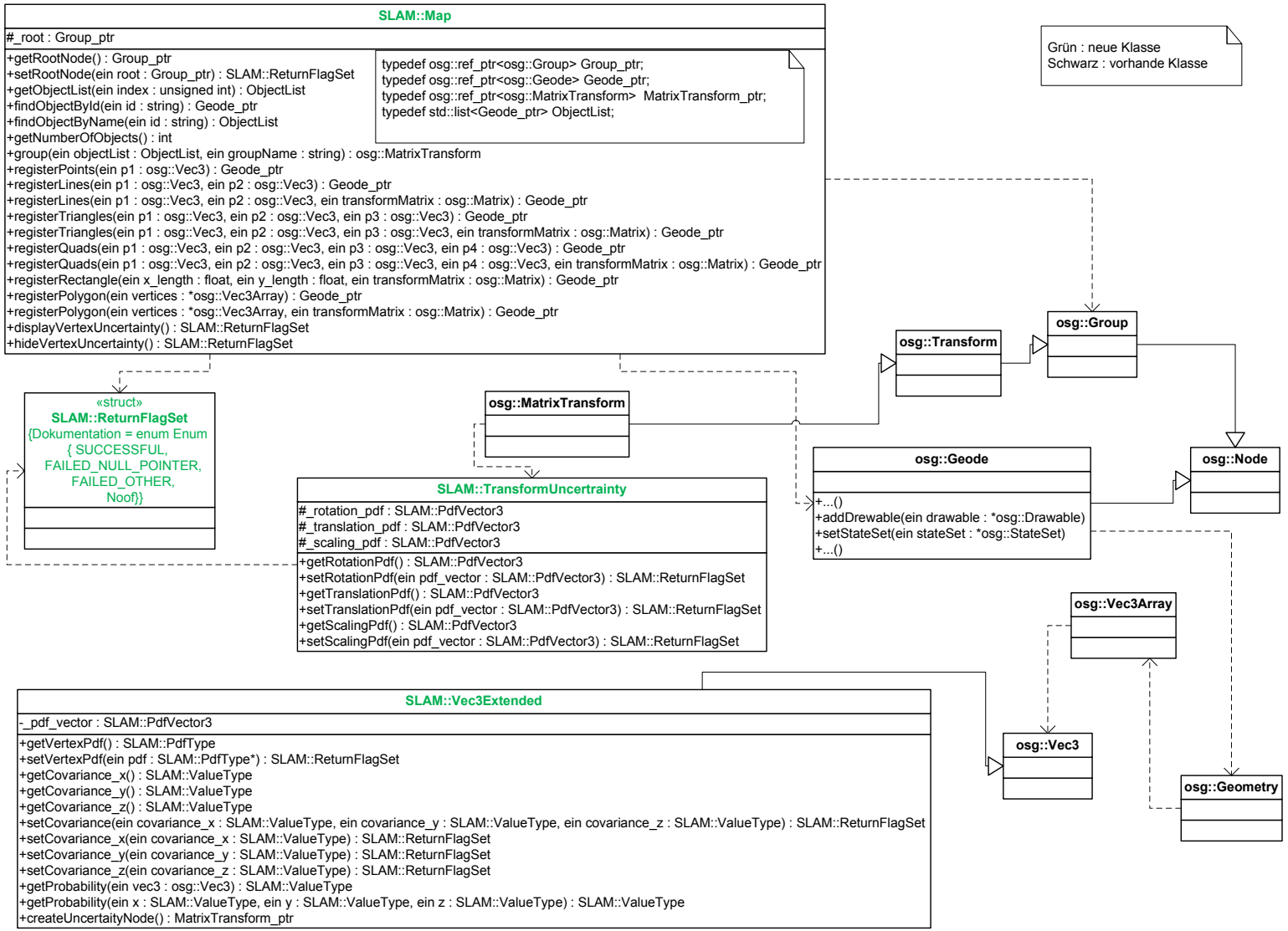


Abbildung 5.2.: Das UML-Diagramm für Klassen bezüglich der Karte

5.5. Demo-Programme

Um zu demonstrieren, wie das Framework funktioniert, wurden vier Demo-Programme als Beispiele implementiert.

5.5.1. Generator-Test

Diese Demo zeigt, wie die Modelle und Filter durch die beiden Generatoren angelegt werden. Der Code basiert auf einem ursprünglichen Beispiel aus der BFL-Bibliothek. In diesem Beispiel wird ein mit einem Ultraschall-Sensor ausgestatteter, autonomer mobiler Roboter simuliert. Der Ultraschall-Sensor misst die Entfernung zu einer Wand. Mithilfe eines Kalman-Filters wird die Position des Roboters ermittelt.

5.5.2. Map-Test

Diese Demo veranschaulicht, wie eine Karte benutzt wird. Eine leere Karte wird zuerst angelegt, dann werden Punkte, Linien, Dreiecke, Rechtecke sowie Polygone nacheinander in die Karte eingefügt. Zuletzt wird die Karte durch einen Viewer visualisiert (Abbildung 5.3).

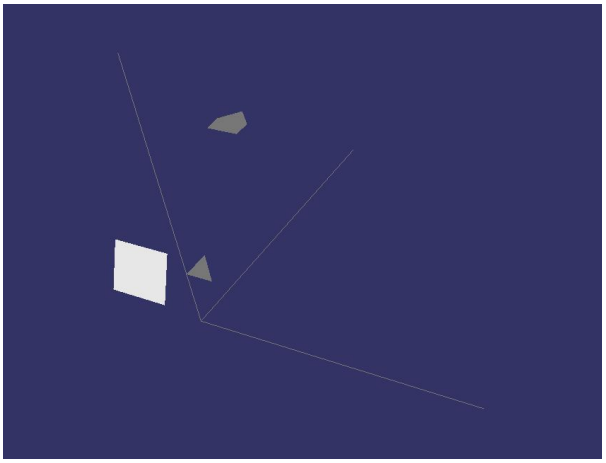
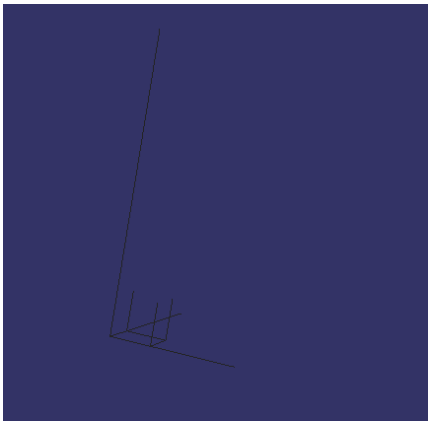


Abbildung 5.3.: Das visualisierte Ergebnis des Map-Tests

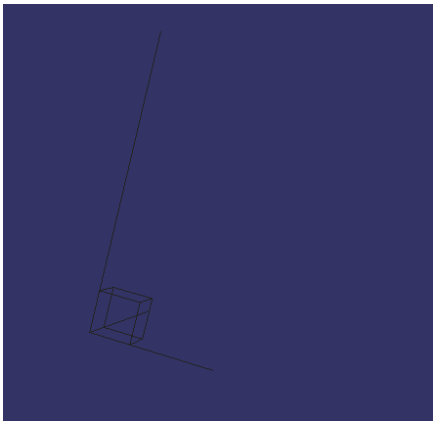
5.5.3. Thread-Test

In praktischen Anwendungen sollen Sensoren, Filter und Karten als unabhängige Threads gleichzeitig ausgeführt werden. Daher wird ein Multi-Thread-Modus benötigt. Diese Demo realisiert das Multi-Threading mithilfe der Openthreads-Bibliothek. Im Hauptprogramm

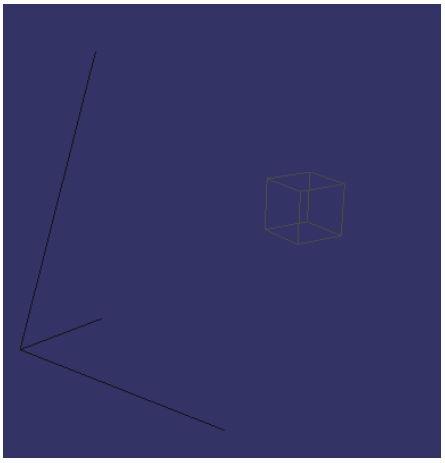
laufen zwei Threads gleichzeitig nebeneinander, einer für die simulierten Sensordatenströme, ein für den Viewer. Es wird angenommen, dass die Kamera einen Würfel erkennt. Jede Sekunde wird der Karte eine Kante hinzugefügt, bis alle 12 Kanten des Würfels in der Karte liegen. Anschließend werden sie gemeinsam gruppiert und dann als ein ganzes Objekt in der Karte bewegt und rotiert (Abbildung 5.4).



(a) Kanten werden nacheinander in die Karte eingetragen.



(b) Alle 12 Kanten werden gemeinsam gruppiert.



(c) Der Würfel wird bewegt und rotiert.

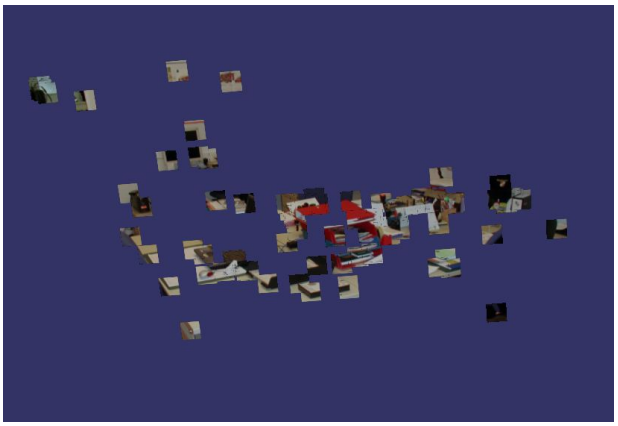
Abbildung 5.4.: Das visualisierte Ergebnis des Thread-Tests

5.5.4. OpenCV-Test

OpenCV ist eine freie C++-Bibliothek mit Algorithmen für die Bildverarbeitung und maschinelles Sehen. Diese Demo zeigt eine Anwendungsmöglichkeit mit visuellen Sensoren. Ein Testbild wird in das Hauptprogramm eingelesen, durch Bildverarbeitung in OpenCV

werden alle robusten Features im Bild detektiert und dann in die Karte eingetragen. Hier sind die Features die Eckpunkte. Schließlich wird durch den Viewer das Ergebnis visualisiert (Abbildung 5.5).

(a) Das originale Bild.



(b) In der Karte gespeicherte Features.

Abbildung 5.5.: Das visualisierte Ergebnis des OpenCV-Tests

6. Diskussion

6.1. Zusammenfassung

Das *Simultaneous Localization and Mapping* (SLAM) Problem ist ein heißes Forschungsthema im Bereich Navigationssysteme. SLAM-Systeme wurden in verschiedenen Umgebungen angewandt, wie zum Beispiel Indoor- [LDW91, CMNT98] , Unterwasser- [LF99, WND+00] und Outdoor-Umgebungen [GNW00, JG00] . Aktuelle Navigationssysteme werden oft mit mehreren Sensoren ausgestattet, die meist kostengünstig sind und ungenaue Messwerte zurückgeben. Wegen der Leistungseinschränkungen der Sensoren und Störungen in der Betriebsumgebung ist eine exakte Lokalisierung anhand der ungenauen wahrgenommen Informationen schwer zu erreichen.

Durch den Ansatz von Sensorfusionsverfahren kann die Unsicherheit des Systems effizient berechnet und damit eine genügende Genauigkeit der Schätzung der Position erhalten werden. Das beliebteste Sensorfusionsverfahren ist der Kalman-Filter, der den Systemzustand rekursiv schätzt und mit den Messwerten aus Sensoren korrigiert. Durch die Linearisierung aller nichtlinearen Terme kann ein erweiterter Kalman-Filter nichtlineare Probleme lösen.

Die Karte ist eine wichtiger Bestandteil eines SLAM-Systems. Je nach Darstellungsweise können Karten hauptsächlich in drei Kategorien unterteilt werden: Rasterkarten [Elf90, Thr01] , Feature-Karten [CK92, LF99] sowie topologische Karten [BNRW99, KB91] . Nach Berücksichtigung der räumlichen Unsicherheit im SLAM-System wurde im SLAM-Framework eine Feature-Karte implementiert.

Das implementierte SLAM-Framework wird in 5 Module unterteilt: Sensoren, Modelle, Filter, Karte sowie SLAM-Kern. Drei quelloffene C++-Bibliotheken wurden bei der Entwicklung verwendet: *Bayesian Filtering Library* (BFL) für das Filter-Modul, *OpenSceneGraph* (OSG) für das Karte-Modul und *OpenCV* für ein Demo-Programm. Mit dem in dieser Arbeit implementierten SLAM-Framework kann eine SLAM-Applikation schnell entwickelt werden.

6.2. Ausblick

Weiterführende Arbeiten könnten das in dieser Arbeit implementierte SLAM-Framework ergänzen und erweitern.

Die Implementierung einer Basisklasse für Sensoren ist notwendig, um die Grundfunktionen eines Sensors einheitlich zu definieren.

Im SLAM-Framework funktionieren bisher nur lineare Kalman-Filter und erweiterte Kalman-Filter. In der BFL-Bibliothek stehen jedoch mehr als zehn Bayes-Filters zur Verfügung, inklusive Partikel-Filter. Die BFL-Bibliothek wird im SLAM-Framework somit noch nicht vollständig ausgenutzt.

Das in [Roso6] beschriebene kontinuierliche Systemmodell kann nicht im diskreten EKF verwendet werden. Um diese Problem zu lösen, kann entweder der Algorithmus von EKF in der BFL-Bibliothek modifiziert werden, oder das kontinuierliche EKF neu implementiert werden.

Neben dem Bayes-Filter gibt es noch zahlreiche andere Sensorfusionsverfahren, wie z.B. Informations-Filter, Fuzzy-Methoden, Neuronale Netze usw. Aufgrund der Zeitbeschränkung wurden bei der Entwicklung des Frameworks nur Bayes-Filter betrachtet. Die anderen Sensorfusionsverfahren könnten in Zukunft auch im SLAM-Framework hinzugefügt werden.

Im Abschnitt 4.1 wurden drei Kartentypen vorgestellt. Neben den Feature-Karten werden auch häufig Rasterkarten in SLAM-System verwendet. Das Framework könnte später um diesen Kartentyp erweitert werden.

A. Installation

Das SLAM-Frawework verwendet die Software 'CMake' als Build-System und drei quelleoffene c++-Bibliotheken: *Bayesian Filtering Library* (BFL) für das Filter-Modul, *OpenSceneGraph* (OSG) für das Karten-Modul und *OpenCV* für ein Demo-Programm.

A.1. CMake

CMake als Build-System muss zuerst installiert werden. Es kann von der Homepage von CMake[1] heruntergeladen werden.

A.2. BFL

Die BFL-Bibliothek verwendet ein andere quelloffene Bibliothek 'Boost'. Die Boost-Bibliothek muss vor Installation der BFL-Bibliothek zur Verfügung gestellt werden.

1. Lade Source-Code der Boost-Bibliothek[2] herunter.
2. Füge den Pfad von dem BOOST-Source-Code-Verzeichnis in den System-Umgebungsvariablen unter dem 'Path'-Eintrag hinzu.
3. Lade Source-Code von der BFL-Bibliothek (Version 0.8.0)[3] herunter.
4. Kompiliere und installiere die BFL-Bibliothek (siehe [4]). Die Fehlermeldung von 'CppUnit' kann ignoriert werden, weil die Testprogramme der BFL-Bibliothek nicht ausgeführt werden müssen.

[1] http://www.cmake.org/cmake/resources/software.html
[2] http://www.boost.org/users/download
[3] http://www.orocos.org/bfl/source
[4] http://people.mech.kuleuven.be/~tdelaet/bfl-doc/installation-guide

A.3. OSG

1. Lade Source-Code der OSG-Bibliothek[5] herunter, oder checke das SVN-Repository[6] aus.
2. Kompiliere und installiere die OSG-Bibliothek (siehe [7]). Alle abhängigen Bibliothek ('dependencies') müssen auch heruntergeladen werden. Bei der CMake-Konfiguration unter 'ACTUAL-3RDPARTY-DIR' muss der entsprechende Pfad eingegeben werden.
3. Teste den OSGViewer, um zu prüfen, ob die OSG-Bibliothek richtig installiert ist.

A.4. OpenCV

1. Installiere die OpenCV-Bibliothek (siehe [8]).
2. Das SLAM-Framework verwendet die OpenCV 2.3.0 . Falls die Version der installierten OpenCV-Bibliothek nicht 2.3.0. ist, muss die Datei 'FindOpenCV.cmake' (unter SLAM-Framework/config) abgeändert werden: alle Nummern '230' durch entsprechende Versionsnummer ersetzen.

A.5. Systempfad

Füge den Pfad der BFL-Installation, der OSG-Installation-Verzeichnis sowie der OpenCV-Installation in System-Umgebungsvariable unter dem 'Path'-Eintrag hinzu.

A.6. CMake-Konfiguration

1. Starte die Software 'CMake'.
2. Wähle bei 'Source Code' das Root-Verzeichnis (nicht '/src' !) des SLAM-Frameworks aus.
3. Trage bei 'Binaries' den Pfad 'SLAM-Framework-Verzeichnis/build' ein.
4. Klicke den Button 'Configure'. Wähle die spezifische IDE und Compiler aus dann klicke den Button 'finish'.

[5] http://www.openscenegraph.org/projects/osg/wiki/Downloads/CurrentRelease
[6] http://www.openscenegraph.org/projects/osg/wiki/Downloads/SVN
[7] http://www.openscenegraph.org/projects/osg/wiki/Support/GettingStarted
[8] http://opencv.willowgarage.com/wiki/InstallGuide

5. Prüfe nach, ob BOOST-, BFL-, OSG- und OpenCV-Bibliothek richtig gefunden werden. Falls eine oder mehrere Bibliotheken nicht gefunden werden, entweder manuell eintragen oder Systempfad nachprüfen, dann CMake-GUI erneut starten. Falls der Pfad von allen Bibliotheken richtig angezeigt wird, klicke noch einmal den Button 'Configure' und dann den Button 'Generate'.
6. Beende CMake-GUI, nachdem die Projekt-Datei generiert wurde.

A.7. IDE

1. Öffne das Projekt durch der Projektdatei 'SLAM-Framework-Verzeichnis/build/SLAM-Framework.*' (Die Endung * ist unterschiedlich für unterschiedliche IDE)
2. Compiliere das SLAM-Framework.
3. Compiliere und starte die Demo-Prgramme 'test-generator' 'test-map' 'test-thread' und 'test-opencv'. Die OSG-Bibliothek und die BFL-Bibliothek sind dynamische Bibliotheken. Für eine dynamische Bibliothek muss die entsprechende *.dll Datei unter '/build/examples/test-verzeichnis' mit kopiert werden, damit sie beim Testen richtig funktioniert. Die Abhägigkeiten der Demo-Programme und Bibliotheken sind wie folgt:
   - test-generator : BFL
   - test-map : OSG, OSGViewer, OSGGA, OSGDB, OSGUtil, OSGText und OpenThreads (von OSG)
   - test-thread : OSG, OSGViewer, OSGGA, OSGDB, OSGUtil, OSGText und OpenThreads (von OSG)
   - test-opencv : OpenCV-core, OSG, OSGViewer, OSGGA, OSGDB, OSGUtil, OSGText und OpenThreads (von OSG)

Literaturverzeichnis

[AHWOW04] G. Abwerzger, B. Hofmann-Wellenhof, B. Ott, E. Wasle. GPS/SBAS and Additional Sensor Integration for Pedestrian Applications in Difficult Environments. *Proceedings of the 17th International Technical Meeting of the Satellite Division of The Institute of Navigation*, pp. 766–774, 2004. (Zitiert auf Seite 7)

[BJP93] M. Buchberger, K.-W. Jörg, E. von Puttkamer. Laserradar and sonar based world modeling and motion control for fast obstacle avoidance of the autonomous mobile robot MOBOT-IV. *Proceedings of the IEEE International Conference on Robotics and Automation*, 1:534–540, 1993. (Zitiert auf Seite 18)

[BNRW99] T. Bailey, E. M. Nebot, J. K. Rosenblatt, H. F. D. Whyte. Robust Distinctive Place Recognition for Topological Maps. *Proceedings of the IEEE International Conference on Field and Service Robotics*, pp. 347–352, 1999. (Zitiert auf den Seiten 27 und 41)

[CK92] K. Chong, L. Kleeman. Mobile robot map building from an advanced sonar array and accurate demetry. *IEEE Journal of Robotics Research*, 11(4):286–298, 1992. (Zitiert auf den Seiten 27 und 41)

[CL85] R. Chatila, J.-P. Laumond. Position Referencing and Consistent World Modeling for Mobile Robots. *Proceedings of International Conference on Robotics and Automation*, pp. 138–145, 1985. (Zitiert auf Seite 13)

[CMNT98] J. A. Castellanos, J. M. Martinez, J. Neira, J. D. Tardos. Simultaneous map building and localization for mobile robots: a multisensor fusion approach. *Proceedings of the IEEE International Conference on Robotics and Automation*, 2:1244–1249, 1998. (Zitiert auf den Seiten 11 und 41)

[CN01] H. Choset, K. Nagatani. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Transactions onRobotics and Automation*, 17:125–137, 2001. (Zitiert auf Seite 12)

[CN03] S. Ceranka, M. Niedzwiecki. Application of Particle Filtering in Navigation System for Blind. *Proceedings of the Seventh International Symposium on Signal Processing and Its Applications*, 2:495–498, 2003. (Zitiert auf Seite 7)

[DFBT99] F. Dellaert, D. Fox, W. Burgard, S. Thrun. Monte Carlo Localization for Mobile Robots. *IEEE International Conference on Robotics and Automation (ICRA99)*, 2:1322–1329, 1999. (Zitiert auf Seite 12)

[Elf87] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3(3):249–265, 1987. (Zitiert auf Seite 12)

[Elf89] A. Elfes. *Occupancy Grids: A probabilistics framework for robot perception and navigation*. Ph.D. thesis, Carnegie Mellon University, 1989. (Zitiert auf Seite 27)

[Elf90] A. Elfes. Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception. *Proceeings of the Sixth Conference on Uncertainy in Artificial Intelligence*, pp. 136–146, 1990. (Zitiert auf den Seiten 27 und 41)

[GNW00] J. Guivant, E. Nebot, H. D. Whyte. Simultaneous Localization and Map Building Using Natural features in Outdoor Environments. *Proceedings of the IEEE International Conference on Intelligent Autonomous Systems*, pp. 316–321, 2000. (Zitiert auf den Seiten 11, 12 und 41)

[HK00] A. Heale, L. Kleeman. Laser-radar based mapping and navigation for an autonomous mobile robot. *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2:948–952, 2000. (Zitiert auf Seite 19)

[HKP90] P. Hoppen, T. Knieriemen, E. von Puttkamer. Laser-radar based mapping and navigation for an autonomous mobile robot. *Proceedings of the IEEE International Conference on Robotics and Automation*, 2:948–953, 1990. (Zitiert auf Seite 18)

[HM04] C. Hide, T. Moore. Low Cost Sensors, High Quality Integration. *Proceedings of NAV04, Location and Timing Applications*, pp. 40.1–4.10, 2004. (Zitiert auf Seite 26)

[HS08] S.-Y. Hwang, J.-B. Song. Upward Monocular Camera based SLAM Using Corner and Door Features. *Preoceedings of the 17th IFAC World Congress*, pp. 1663–1668, 2008. (Zitiert auf den Seiten 5, 12 und 28)

[JG00] S. B. Jose Guivant, Eduardo Nebot. Localization and map building using laser range sensors in outdoor applications. *Journal of Robotic Systems*, 17:565–583, 2000. (Zitiert auf den Seiten 11, 12 und 41)

[Kal60] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME – Journal of Basic Engineering*, 82:35–45, 1960. (Zitiert auf Seite 22)

[KB61] R. E. Kalman, R. S. Bucy. New results in linear filtering and prediction theory. *Transactions of the ASME - Journal of Basic Engineering*, 83:95–107, 1961. (Zitiert auf Seite 22)

[KB91] B. Kuipers, Y.-T. Byun. A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991. (Zitiert auf den Seiten 5, 27, 29 und 41)

[KK94] L. Kleeman, R. Kuc. An Optimal Sonar Array for Target Localization and Classification. *IEEE International Conference on Robotics and Automation*, 4:3130–3135, 1994. (Zitiert auf Seite 19)

[Kleo8] B. Kleiner. *Generierung von Stützinformationen aus optischen Systemen für inertiale Navigationssysteme*. Master's thesis, Technische Universität Dresden, 2008. (Zitiert auf Seite 7)

[Komo6] S. Kombrink. Master's thesis, Universität Stuttgart, 2006. (Zitiert auf Seite 7)

[Krao4] W. Krause. *Konstruktionselemente der Feinmechanik*. München: Hanser, 2004. (Zitiert auf Seite 13)

[KW94] D. Kortenkamp, T. Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. *Proceedings of the Twelfth National Conference on Artifical Intelligence*, pp. 979–984, 1994. (Zitiert auf Seite 29)

[LDW91] J. J. Leonard, H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*, p. 1442–1447, 1991. (Zitiert auf den Seiten 11 und 41)

[LF99] J. J. Leonard, H. J. S. Feder. A Computationally Efficient Method for Large-Scale Concurrent Mapping and Localization. *Proceedings of the Ninth International Symposium on Robotics Research*, 1:316–321, 1999. (Zitiert auf den Seiten 11, 27 und 41)

[ME85] H. Moravec, A. Elfes. High resolution maps from wide angle sonar. *Proceedings of International Conference on Robotics and Automation*, pp. 116–121, 1985. (Zitiert auf Seite 27)

[MJ97] D. Murray, C. Jennings. Stereo vision based mapping and navigation for mobile robots. *Proceedings of International Conference on Robotics and Automation*, 2:1694–1699, 1997. (Zitiert auf Seite 20)

[Nah76] N. E. Nahi. Estimation Theory and Applications. *New York: R.E. Krieger.*, 1976. (Zitiert auf Seite 14)

[RAK09] P. Robertson, M. Angermann, B. Krach. Simultaneous Localization and Mapping for Pedestrians using only Foot-Mounted Inertial Sensors. In *Ubicomp 2009*. 2009. (Zitiert auf Seite 7)

[RBoo] S. I. Roumeliotis, G. A. Bekey. Bayesian estimation and Kalman filtering: A unified framework for Mobile Robot Localization. *Proceedings of International Conference on Robotics and Automation*, pp. 2985–2992, 2000. (Zitiert auf Seite 22)

[RGHR10] A. R. J. Ruiz, F. S. Granja, J. C. P. Honorato, J. I. G. Rosas. Pedestrian Indoor Navigation by aiding a foot-mounted IMU with RFID signal Strength Measurement. *2010 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–7, 2010. (Zitiert auf Seite 7)

[Roso6] H. von Rosenberg. *Sensorfusion zur Navigation eines Fahrzeugs mit low-cost Inertialsensorik*. Master's thesis, Universität Stuttgart, 2006. (Zitiert auf den Seiten 7, 8, 22 und 42)

[SC86] R. C. Smith, P. Cheeseman. On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986. (Zitiert auf Seite 11)

[SD98] S. Simhon, G. Dudek. A Global Topological Map Formed by Local Metric Maps. *Proceedings of IEEE International Conference on Intelligent Robotics and Systems*, pp. 1708–1717, 1998. (Zitiert auf Seite 29)

[SSC86] R. C. Smith, M. Self, P. Cheeseman. Estimating Uncertain Spatial Relationships in Robotics. *Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence*, p. 435–461, 1986. (Zitiert auf den Seiten 5, 11, 13, 14 und 16)

[Tay76] R. Taylor. A Synthesis of Manipulator Control Programs from Task-Level Specifications. *AIM-282*, 1976. Stanford University Artificial Intelligence Laboratory. (Zitiert auf Seite 13)

[TBF98] S. Thrun, W. Burgard, D. Fox. A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots. *Machine Lerning*, 31(1-3):29–53, 1998. (Zitiert auf Seite 12)

[Thr98] S. Thrun. Learning Metric-Topological Maps for Indoor Mobile Robot Nativation. *Artificial Intelligence*, 99(1):21–71, 1998. (Zitiert auf Seite 29)

[Thr01] S. Thrun. Learning Occupancy Grids with Forward Models. In *Proceedings of IEEE International Conference on Intelligent Robotics and Systems*, pp. 1676–1681. 2001. (Zitiert auf den Seiten 5, 27, 28 und 41)

[TK95] C. Taylor, D. Kriegman. Vision-Based Motion Planning and Exploration Algorithms for Mobile Robots. In *Workshop on the Algorithmic Foundations of Robotics*, volume 14, pp. 417–426. 1995. (Zitiert auf Seite 20)

[TKG+02] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, A. Y. Ng. Simultaneous Mapping and Localization With Sparse Extended Information Filters. In *Proceedings of the Fifth International Workshop on Algorithmic Foundations of Robotics*, volume 23, pp. 693–716. Nice, France, 2002. (Zitiert auf Seite 12)

[WB06] G. Welch, G. Bishop. An Introduction to the Kalman Filter. Technical report, University of North Carolina at Chapel Hill, 2006. (Zitiert auf den Seiten 5, 23 und 25)

[WKAR06] K. Wendlandt, M. Khider, M. Angermann, P. Robertson. Continuous location and direction estimation with multiple sensors using particle filtering. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pp. 92–97. 2006. (Zitiert auf Seite 7)

[WL90] E. Waltz, J. Llinas. *Multisensor Data Fusion*. Artech House, Inc. Norwood, MA, USE, 1990. (Zitiert auf Seite 20)

[WND+00] S. B. Williams, P. Newman, G. Dissanayake, J. Rosenblatt, H. Durrant-whyte. A decoupled, distributed AUV control architecture. In *Proceedings of 31st International Symposium on Robotics*, volume 1, pp. 246–251. 2000. (Zitiert auf den Seiten 11 und 41)

[YM02] D. Yuen, B. A. MacDonald. A comparison between extended Kalman filtering and sequential Mente Carlo techniques for simultaneous localisation and map building. In *Proceeings of Australian Conference on Robotics and Automation*, pp. 335–340. 2002. (Zitiert auf Seite 12)

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Zhen Peng)