

Praktikum

Forschungsprojekt

*Anthropomatik praktisch
erfahren*

10. Februar 2017

Fabian Peller, Christoph Roth, Benedikt Haas, Shi
Qiyao, Matthias Börsig
Simultane Lokalisierung und Kartografierung mit
dem ISAS-Crawler

Simultane Lokalisierung und Kartografierung mit dem ISAS-Crawler

– Praktikum: Forschungsprojekt Anthropomatik praktisch erfahren –

Fabian Peller, Christoph Roth, Benedikt Haas, Shi Qiyao, Matthias Börsig

10. Februar 2017

Zusammenfassung

Simultane Lokalisierung und Kartografierung (SLAM) ist eine Problemstellung der mobilen Robotertechnik, die aus zwei Teilen besteht. Zum einen muss ein Roboter eine Karte seiner Umgebung erstellen, zur selben Zeit muss er seine Pose, also Position und Orientierung, in ebenjener Karte schätzen. Um eine Karte seiner Umgebung erstellen zu können, muss die eigene Position bekannt sein und um seine Position zu schätzen, muss eine Umgebungskarte vorhanden sein, somit handelt es sich hierbei um ein Henne-Ei-Problem. Ziel des Praktikums war es SLAM mit Hilfe des ISAS-Crawler und dessen Ultraschallsensoren umzusetzen und parallel dazu in einer Simulationsumgebung den Crawler und das SLAM Verfahren zu simulieren. Es gibt viele Möglichkeiten dies umzusetzen. Im Rahmen des Praktikums wurden Occupancy Grid-Maps verwendet, um die Umgebungskarte abzubilden. Um die Pose zu schätzen wurden Partikelfilter genutzt. Das SLAM Verfahren wurde mit Grid-based FastSLAM implementiert. Ein Teil der Funktionen der Software werden durch die Nonlinear-Estimation-Toolbox bereitgestellt, darunter der eingesetzte Filter. Um das Ergebnis zu evaluieren wurde die Position des Roboters von einer Deckenkamera bestimmt und mit der geschätzten Position verglichen, um so den Fehler der Schätzung messbar zu machen.

Inhaltsverzeichnis

Abkürzungsverzeichnis	4
Abbildungsverzeichnis	5
1 Einleitung	7
2 Projektplan	8
3 ISAS-Crawler	9
3.1 Aufbau	9
3.1.1 Hardware	9
3.1.2 Fortbewegung	10
3.1.3 Energieversorgung	10
3.2 Hardwarefehler	11
3.3 Software	11
3.3.1 Top-Level	12
3.3.2 Mikrocontroller	13
4 Simultane Lokalisierung und Kartografierung	15
4.1 Occupancy Grid-Maps	16
4.2 Partikelfilter	18
4.3 Grid-based FastSLAM	19
5 Simulationsumgebung	20
5.1 Simulationsaufbau	20
5.2 Benutzung der grafischen Benutzeroberfläche	22
6 Implementierung	24
6.1 NLE-Toolbox	24
6.1.1 Änderungen an der Toolbox	24
6.1.2 Funktionen der Toolbox	25
6.2 Systemmodell	25

6.3	Messmodell	27
6.3.1	Sensorrauschen	27
6.3.2	Bestimmung des Fehlers	27
6.3.3	Ansteuerung der Sensoren	30
6.3.4	Bezug zur Toolbox	31
6.4	CSLAM	32
7	Evaluation	34
7.1	Aufnahmesysteme	34
7.2	Evaluierung	36
7.2.1	Vorgehen	36
7.2.2	Ergebnis simulierter Crawler	36
7.2.3	Ergebnis realer Crawler	38
8	Zusammenfassung & Ausblick	40

Abkürzungsverzeichnis

EKF	Extended Kalman Filter
ESS	Estimated Sample Size
GB	Grid Based
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IMU	Inertial Measurement Unit
ISAS	Intelligente Sensor-Aktor-Systeme
LED	Light Emitting Diode
MCL	Monte Carlo Localization
MIPS	Microprocessor without Interlocked Pipeline Stages
MSE	Mean Squared Errors
MUX	Multiplexer
NLE-Toolbox	Nonlinear-Estimation-Toolbox
PF	Particle Filter
REST	Representational State Transfer
SD-Card	Smart Digital Card
SIRPF	Sampling Importance Resampling Particle Filter
SLAM	Simultane Lokalisierung und Kartografierung
UML	Unified Modeling Language
USB	Universal Serial Bus
WLAN	Wireless Local Area Network

Abbildungsverzeichnis

1	Aufbau des aktuellen Crawlers	9
2	Graphische Darstellung der Bewegungsmöglichkeiten der Beine des Crawlers . .	11
3	Schematischer Ablauf eines Befehlsaufrufes	12
4	Beispiel Grid: belegte Zellen sind durch schwarz markiert, freie Zellen durch weiß. [7]	16
5	Sensormodell eines Ultraschallsensors. Schwarz markierte Zellen gelten als belegt. [7]	17
6	Inkrementelle Updates der Karte und finale Maximum Likelihood Map	18
7	Darstellung dreier Partikel mit der jeweiligen GridMap [6]	20
8	UML Diagramm der Softwarearchitektur	21
9	GUI der Simulation	22
10	Systemmodell: Bewegungsrichtung in Weltkoordinaten und Plot mit 200 Samplebewegungen für die Bewegungsrichtungen vorwärts, rückwärts und seitwärts.	26
11	Vergleich des Mittelwerts über alle Messungen der Rohdaten, gefilterter Messungen und der realen Distanz	28
12	Vergleich der Varianz der Roh- sowie gefilterten Daten	29
13	Absolute Differenz der Roh- und gefilterten Daten im Vergleich zur „realen“ Entfernung	29
14	Absolute Differenz der Roh- sowie gefilterten Daten zur „realen“ Entfernung . .	30
15	Kamera und Kalibrierungsschablone mit Schachbrettmuster	34
16	Kamerabild und Zusammenhang von Roboterkoordinaten und Weltkoordinaten .	35
17	Ergebnisse der Evaluation der Simulation, durchgeführt mit 1000 Partikeln. Reale Route (rot) und geschätzte Route (blau) des Crawlers, sowie die Distanz der Positionen (grün)	36
18	Ergebnisse der Evaluation der Simulation, durchgeführt mit einem Partikel. Reale Route (rot) und geschätzte Route (blau) des Crawlers, sowie die Distanz der Positionen (grün)	37
19	(a) Gridmap aus Simulation mit einem Partikel, (b) reale Karte, (c) Gridmap aus Simulation mit 1000 Partikeln	38
20	Reale Route (rot) und geschätzte Route (blau) des Crawlers sowie die Distanz der Positionen (grün)	38
21	Reale Karte und erstelle Karte des Roboters	39

1 Einleitung

Der ISAS-Crawler (kurz: Crawler) ist ein mobiler Agent, der am Institut für Intelligente Sensor-Aktor-Systeme (ISAS) entwickelt wird. Dieser besitzt drei Beine, mithilfe derer er sich fortbewegen kann und vier Ultraschallsensoren, die für die Erkundung der Umwelt verwendet werden können. Weitere Details bzgl. des Aufbaus des Crawlers können Abschnitt 3 entnommen werden. Jedoch existiert noch keine Software, die es ermöglicht, den Crawler zu steuern und die von ihm gelieferten Messdaten zu verwerten. Ein Ansatz, mithilfe dessen diese Problematik gelöst werden kann, wird als SLAM bezeichnet. SLAM steht für Simultane Lokalisierung und Kartographierung (engl. simultaneous localization and mapping) und ermöglicht es, eine Karte der Umwelt zu erstellen, sowie sich selbst in dieser zu lokalisieren. Verwendung findet es vor allem in der Robotik (s. [4], [5]), da es die Grundlage für ein autonomes Verhalten bietet. Ziel dieser Arbeit ist der Entwurf sowie die Umsetzung eines SLAM Verfahrens für den o.g. Crawler und die Erstellung einer Simulationsumgebung für dieses System.

Tabelle 1: Projektplan

Arbeitsaufgabe	M1↓					M2↓					M3↓					M4↓		
	KW 43	KW 44	KW 45	KW 46	KW 47	KW 48	KW 49	KW 50	KW 51	KW 2	KW 3	KW 4	KW 5	KW 6				
Präsentation	Green				Green									Green	Yellow			
Crawler		Green	Yellow															
Simulationsumgebung			Green			Zwischenbericht										Endbericht + Abgabe der Ausarbeitung		
Ultraschall			Green															
Evaluation Ultraschall				Green														
Auswahl Map & SLAM Verfahren	Green																	
SLAM Implementierung			Red	Green			Green	Green	Yellow									
Evaluation									Red									
Schriftliche Ausarbeitung	Green												Yellow					

Farblegende: rechtzeitig, verzögert, entfallen

Abkürzungen: m Meilenstein, KW Kalenderwoche

2 Projektplan

Zu Beginn des Projektes wurde ein Zeitplan (s. Tabelle 1) zwecks besserer Koordinierung und Zeitmanagement erstellt. Hierbei wurden die einzelnen Aufgaben derart gewählt, dass sie zu zweit bzw. zu dritt bearbeitet werden können. Dabei wurde darauf geachtet, dass möglichst kein Stillstand entsteht, d.h. die Teams aus zwei bis drei Personen möglichst parallel arbeiten können.

Bei der zeitlichen und organisatorischen Einteilung sind somit drei logische Blöcke entstanden:

1. Inbetriebnahme und Ansteuerung der Peripherie des Crawlers
2. Simulationsumgebung
3. Implementierung des SLAM Verfahrens

Hierbei können 1 und 2 parallel bearbeitet werden, wodurch genügend Zeit für die Implementierung des SLAM Verfahrens (3) bleibt.

Da es bei der Bearbeitung von 1 zu unvorhergesehenen Problemen (s. Kapitel 3.2) kam, verzögerte sich die Fertigstellung von 1 um ca. zwei Wochen. Zusätzlich kam es bei der Umsetzung des SLAM Verfahrens ebenfalls zu einer Verzögerung, weshalb die Evaluation gekürzt werden musste.

3 ISAS-Crawler

Der ISAS-Crawler (kurz: Crawler), wird am Lehrstuhl für intelligente Sensor-Aktor-Systeme (ISAS) entwickelt. Hierbei handelt es sich um einen mobilen Agenten, welcher Sensoren besitzt, die verrauschte Daten zurückliefern. Sein Aufbau sowie seine Funktionsweise sind Inhalt dieses Kapitels.

3.1 Aufbau

Da der Crawler ständig weiterentwickelt wird, gibt es verschiedene Versionen von ihm. Die von uns verwendete ist in Bild 1 zu sehen.

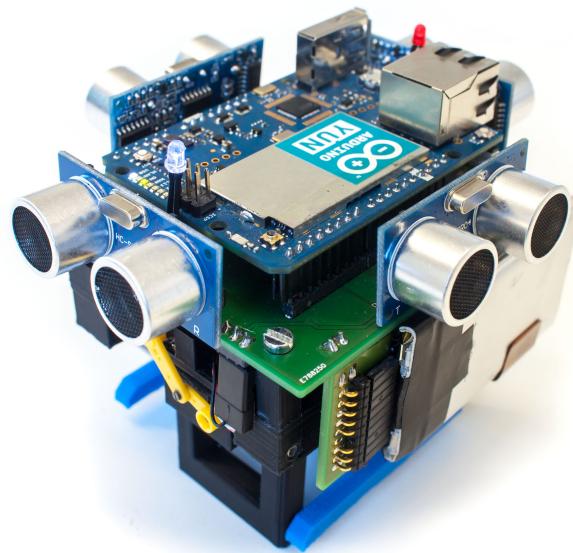


Abbildung 1: Aufbau des aktuellen Crawlers

3.1.1 Hardware

Der Crawler besitzt folgende Hardwarekomponenten:

Arduino Yún Dieser besitzt zwei Prozessoren: eine MIPS sowie einen ATmega. Der ATmega dient als Mikrocontroller und kann somit die auf der Platine angebrachte Hardware ansteuern. Auf der MIPS ist ein OpenWRT vorinstalliert. Da dieser leistungsfähiger ist, sollten ressourcenintensive Berechnungen möglichst hier durchgeführt werden. Zusätzlich besitzt der Arduino Yun ein WLAN Modul, mit dem ein WLAN erzeugt bzw. sich in ein vorhandenes eingewählt werden kann. Des Weiteren besitzt er einen USB- und Ethernet-Eingang sowie ein SD-Card Slot. Hierbei wird die Seite, an der sich der SD-Card Slot befindet, als Vorderseite des Crawlers bezeichnet.

Ultraschallsensoren Es sind insgesamt vier Ultraschallsensoren verbaut, die jeweils aus einem Empfänger sowie einem Sender bestehen. Angesteuert werden die Sender sowie Empfänger über jeweils einen Multiplexer (MUX). Diese Designentscheidung wurde getroffen, da bei einer vorherigen Version nicht genügend Pins auf dem Arduino Yun zur Verfügung gestanden hatten. Die Sensoren sind in jeweils 90°-Winkeln zueinander angeordnet, wodurch ein überschneidungsfreies Erkunden der Bewegungssachsen möglich ist.

Servomotoren Damit der Crawler sich mithilfe der drei Beine fortbewegen kann, sind aktuell vier Servomotoren verbaut. Dadurch kann dieser sich vorwärts, rückwärts und seitwärts bewegen, sowie sich um sich selbst drehen.

LEDs Zusätzlich befinden sich auf der Oberseite zwei LEDs (blau und rot), mithilfe derer der Roboter leichter bspw. von einer Deckenkamera, verfolgt werden kann.

IMU Des Weiteren ist eine IMU verbaut, die spezifischere Bewegungsdaten misst, z.B. die Beschleunigung. Diese wird jedoch aktuell nicht verwendet.

3.1.2 Fortbewegung

Eine Fortbewegung ist mittels der drei Beine möglich. Eine graphische Darstellung der Bewegungsrichtungen der Beine ist in Bild 2 zu sehen.

Front- & Backfoot: Hierbei handelt es sich um die beiden schwarzen Beine, die durch ihre relative Position zueinander als forderer und hinterer Fuß bezeichnet werden. Sie werden jeweils durch einen Servomotor angetrieben, weshalb sie nur einen Freiheitsgrad besitzen, d.h. sie können sich lediglich nach vorne bzw. hinten bewegen (Abb. 2(a)).

Bigfoot: Als Bigfoot wird der blaue Fuß bezeichnet. Dieser wird durch zwei Motoren angetrieben und kann sich dadurch nach oben und nach unten (Abb. 2(c)), sowie vorwärts und rückwärts (Abb. 2(b)) bewegen.

Durch diese Anordnung der Beine kann der Crawler sich nach vorne, hinten sowie seitwärts bewegen und sich um sich selbst drehen.

3.1.3 Energieversorgung

Die Energieversorgung erfolgt durch zwei Akkus. Diese werden über einen Steckverschluss angeschlossen, wodurch ein Wechsel ermöglicht und vereinfacht wird. Jedoch ist während des Gebrauchs auf den Ladestatus der Akkus zu achten, da es keine Schutzvorrichtung bzgl. der Tiefenentladungen gibt. Diese zerstören die Akkus und können während der nächsten Ladung zu einem Brand führen.

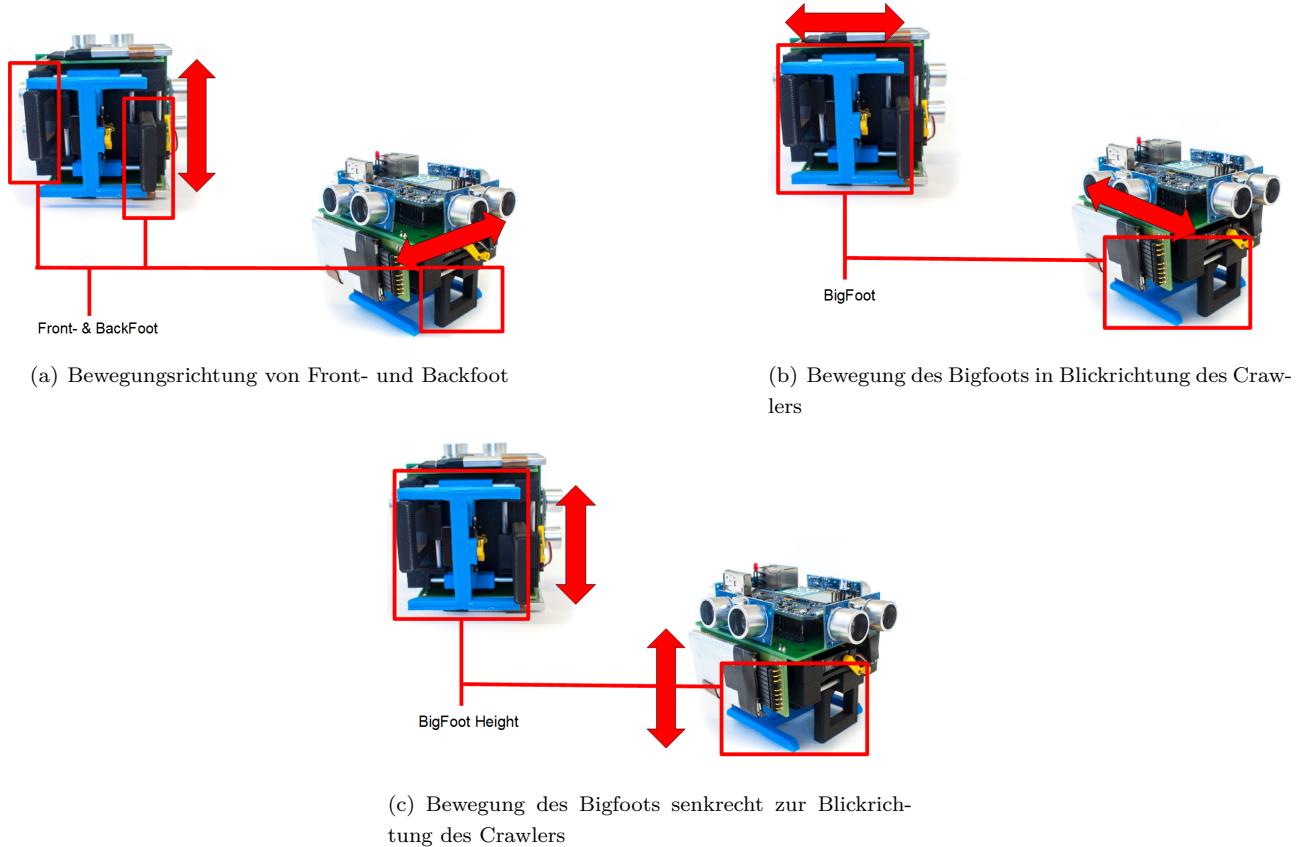


Abbildung 2: Graphische Darstellung der Bewegungsmöglichkeiten der Beine des Crawlers

3.2 Hardwarefehler

Während des Entwurfs sowie dem Aufbau des Crawlers kam es zu Fehlern.

Im Entwurf wurde der sechste Anschluss, auch inhibitor genannt, nicht auf ground gesetzt, wodurch die MUX unabhängig von der Eingabe, kein „HIGH“ ausgeben konnte. Dieser Fehler hat dazu geführt, dass die Ultraschallsensoren nicht richtig angesteuert werden konnten, wodurch diese keine Daten zurückgeliefert haben. Um dies in Zukunft zu verhindern, wurden die entsprechenden Pläne geändert.

Während des Zusammenbaus des Crawlers wurden die Stecker für die Stromversorgung der Servomotoren falsch angeschlossen. Dies führte dazu, dass die Servomotoren die Crawlerbeine nicht bewegen konnten. Um diesen Fehler in Zukunft zu verhindern wurde, bzw. wird in neueren Crawlerversionen die Polarität auf der Platine markiert.

3.3 Software

Für die Software, die auf dem Crawler ausgeführt werden soll, wurden folgende Vorgaben formuliert:

- Vereinfachung des Top-Level Python Codes, wodurch die Wartbarkeit verbessert und Speicher gespart werden soll und dadurch die Benutzung einer SD-Karte überflüssig wird
- Anpassung der Bewegungsperipherie, d.h. die Bewegungen sollen mit vier statt fünf (Vorgängerversion, s. [1]) Servomotoren durchgeführt werden
- Optimierung und Verbesserung des Low-Level Codes
- Implementierung einer Matlab-Bibliothek für die Kommunikation mit dem Crawler

Der Crawler kann auf dem Top-Level sowie dem Low-Level bzw. dem Mikrocontroller Software ausführen.

3.3.1 Top-Level

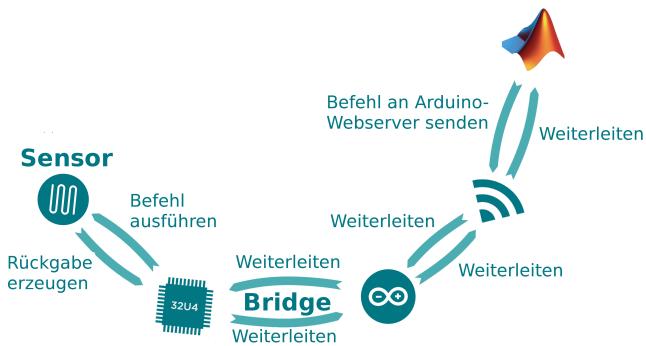


Abbildung 3: Schematischer Ablauf eines Befehlsaufrufes

Mit Top- oder High-Level Software ist diejenige gemeint, die auf der MIPS ausgeführt wird. Hierzu können Python-Skripte oder Funktionen des bereits vorhandenen bzw. vorinstallierten OpenWRT verwendet werden. Hierbei handelt es sich vor allem um die Kommunikation mit der Umwelt und die Ansteuerung des Mikrocontrollers.

Mit dem Wegfallen der SD-Karte, ist der zur Verfügung stehende Speicher deutlich reduziert worden. Daher wurde nach einer Alternativlösung zur Aufsetzung eines eigenen HTTP-Servers zur Kommunikation mit der Umwelt gesucht. Als sehr speichereffizient stellte sich die Verwendung der Built-In Funktionen des OpenWRTs heraus, wozu die zur Verfügungstellung einer REST-Schnittstelle gehört. Befindet sich ein Client in dem vom Arduino aufgespannten Netzwerk, können REST-Aufrufe via <http://arduino.local/arduino/> getätigt werden. Diese werden dann über die Bridge an den ATMega weitergeleitet. Ursprünglich wurde hier die serielle Schnittstelle über ein selbst implementiertes Interface genutzt. Abbildung 3 zeigt schematisch den neuen Programmablauf vom Matlab Client bis hin zum Sensor.

Zusätzlich zu den o.g. Anforderungen sollten keine komplexen Bewegungsbefehle gesendet werden, sondern möglichst atomare Aufrufe durchgeführt werden. Daher werden die Parameter des Aufrufes lediglich an den Mikrocontroller (s. 3.3.2) weitergegeben. Hierbei wurden für die Ansteuerung der einzelnen Komponenten Keywords eingeführt, die als Parameter mit übergeben werden müssen (s. Tabelle 2).

Keyword	Komponente	Effekt
US	Ultraschellsensoren	Durchführung einer Ultraschallmessung
MOVE	Servomotoren	Bewegt die Motoren in die entsprechende Konfiguration
IMU	IMU	Liest die von der IMU gemessenen Werte aus

Tabelle 2: Übersicht über die zur Verfügung stehenden Keywords zur Ansteuerung der einzelnen Komponenten des Crawlers

Da *imu* und *us* an sich atomare Befehle sind, ergeben sich hieraus keine Probleme. Der *move* Befehl bewegt die Servomotoren entsprechend der Konfiguration. Die Reihenfolge der Argumente ist dabei bei jedem Aufruf festgelegt:

1. *BigFoot Movement* $\in [-100, 100]$ die relative Bewegung des BigFoots in %
2. *FrontFoot Movement* $\in [-100, 100]$ die relative Bewegung des FrontFoots in %
3. *BackFoot Movement* $\in [-100, 100]$ die relative Bewegung des BackFoots in %
4. *MinHeight* $\in [0, 100]$ die relative Bewegung der Höhe des BigFoots in % vor der eigentlichen Bewegung des BigFoots
5. *MaxHeight* $\in [0, 100]$ die relative Bewegung der Höhe des BigFoots in % nach der eigentlichen Bewegung des BigFoots
6. *AmountMovements* $\in \mathbb{N}$ gibt die Anzahl der Bewegungen an, die der Roboter mit den oben genannten Parametern durchführen soll

Der Crawler senkt sich dabei zuerst auf *minHeight* und führt dann die Bewegung des BigFoots aus. Anschließend hebt er sich auf *maxHeight* und führt anschließend die Bewegungen des Front- und BackFoots aus. Um ideale Ausgangshaltungen des Crawler vor jeder Bewegung zu garantieren, wird vor jeder Befehlsausführung die Pose des Roboters normiert. Dafür werden alle Bewegungsmotoren auf 50% gestellt und die Höhe des Crawler auf 0% gesetzt. Diese Normierung wird natürlich so durchgeführt, dass alle Verstellungen der Motoren die Position und Orientierung des Roboters nicht beeinflussen.

Schließlich wurden vorgefertigte Bewegungsbefehle der Matlab-Bibliothek hinzugefügt. Unter anderem beinhaltet diese ein Beispielskript, das die Bewegung entlang jeder Bewegungsrichtung exemplarisch durchführt. Die Bewegungsbefehle sind dabei vorgefertigte Profile, die dann während des Programms als Funktionen aufgerufen werden können. Damit der Roboter die Befehle erhält, muss sich der Matlab-Client aber in dem vom Arduino aufgespannten Netzwerk befinden.

3.3.2 Mikrocontroller

Der bereits vorhandene Code des Low-Level ATMega Prozessors wurde komplett überarbeitet und an die neue Problemstellung angepasst. Da als neue Kommunikationsschnittstelle die

Bridge zwischen dem MIPS und dem ATMega Prozessor gewählt wurde, um bereits vorhandene Funktionen des OpenWRT zu nutzen, mussten die benötigten Bibliotheken des Arduino eingebunden werden. Dieser Mehraufwand sorgte jedoch dafür, dass der bereits gegebene Code der Vorgängerversion [1] aufgrund von Speicherbegrenzungen nicht mehr benutzbar war. Deshalb haben wir uns entschlossen, den gegebenen Code vollständig zu überarbeiten. Unsere Änderungen umfassen dabei:

- Objektorientierte Module für Ultraschall-Sensoren, IMU und Servomotoren
- Angepasste Bewegungsperipherie für die Bewegung mit vier anstatt fünf Servomotoren
- Eine deutlich einfachere Struktur, die man nur durch die Änderung von Konfigurationskonstanten anpassen kann. Diese Konstanten werden zu Beginn des Programms initialisiert.

4 Simultane Lokalisierung und Kartografierung

Das Ziel des Praktikums ist die Implementierung eines *simultaneous Localization and Mapping* (SLAM) - Verfahrens. Hierbei ist zu beachten, dass SLAM auf zweierlei Arten interpretiert werden kann:

1. SLAM als Problemdimension
2. SLAM als eigenständiges Verfahren

Wie bereits erwähnt handelt es sich hierbei um die Implementierung eines Verfahrens (s. Punkt 2). Jedoch wurden keine Vorgaben bezüglich des zu verwendenden Verfahrens getätigt, weshalb auch SLAM als Problem (s. Punkt 1) bzw. die Problemdimension von SLAM betrachtet wird.

Das SLAM-Problem wird auch als Henne-Ei-Problem bezeichnet, da hierbei zwei voneinander abhängige Probleme gelöst werden müssen:

- Erstellen einer Umgebungskarte mithilfe einer gegebenen Position
- Lokalisierung bzw. Bestimmung der eigenen Position anhand einer gegebenen Umgebungs-karte

Um dieses Problem lösen zu können werden in der Regel ein bzw. mehrere Sensoren zur Erkundung der Umwelt verwendet. Der in diesem Projekt verwendete Crawler besitzt hierfür vier Ultraschallsensoren (s. Kapitel 3.1.1). Da Ultraschallsensoren rauschanfällig sind und lediglich die Entfernung des am nächsten liegenden Objektes in einem spezifischen Kegel misst, wird ein Verfahren benötigt, das mit der daraus resultierenden Unsicherheit umgehen kann. Weit verbreitete Verfahren sind:

1. landmarkenbasiert oder
2. verwenden occupancy-gridmaps

Für die Verwendung von 1 werden Landmarken benötigt. Landmarken sind charakteristische Merkmale der Umgebung, wie z.B. Objekte oder Wände. Aufgrund der Rauschanfälligkeit und der schlechten Auflösung des Ultraschallsensors wäre ein solcher Ansatz bezüglich der Erkennung sowie der Wiedererkennung (z.B. nach Bewegung) von Landmarken problematisch. Daher werden *occupancy-gridmaps* (s. Kapitel 4.1) verwendet.

Für die Lokalisierung innerhalb einer Karte gibt es bereits verschiedenste Verfahren (siehe [4], [5]). Jedoch verwenden viele Verfahren Landmarken, z.B. Extended Kalman Filter (EKF), die jedoch nicht zur Verfügung stehen. Daher wird ein Verfahren benötigt, das keine Landmarken verwendet und optional die durch die verwendeten Sensoren entstehende Unsicherheit ebenfalls modelliert. Dadurch eignet sich besonders die Verwendung eines Partikelfilters (s. Abschnitt 4.2), da dieser eben nicht auf Landmarken angewiesen ist. Hierbei gibt es viele Partikel, die

jeweils eine mögliche Pose darstellen. Ziel ist es, Partikel derart zu verschieben, dass sie möglichst wahrscheinlich sind. Anhand dieser „wahrscheinlichen“ Partikel kann nun die Position geschätzt werden.

Um die beiden genannten Verfahren zu kombinieren verwendet man *grid-based FastSLAM* (s. Kapitel 4.3). Hierbei ist jedoch anzumerken, das *FastSLAM* kein allgemeines Verfahren, sondern eine spezifische Umsetzung bzw. Implementierung des Partikelfilters ist.

4.1 Occupancy Grid-Maps

Zur Representation der Umgebung des hier verwendeten Roboters (ISAS-Crawler) kommt eine so genannte Occupancy Grid Map zum Einsatz. Occupancy Grid Maps sind ein populärer, probabilistischer Ansatz, um vor allem strukturierte Umgebung innerhalb eines Gebäudes darzustellen [5].

Grid Maps diskretisieren die Umgebung in Zellen fester Größe und ordnen jeder dieser Zellen einen Wahrscheinlichkeitswert zu, je nachdem, ob diese frei oder belegt ist. Abbildung 4 zeigt beispielhaft ein solches Grid. Der Wahrscheinlichkeitswert für eine belegte Zelle strebt hier gegen 0: $p(m_i) \rightarrow 0$, die Wahrscheinlichkeit für eine freie Zelle gegen 1: $p(m_j) \rightarrow 1$.

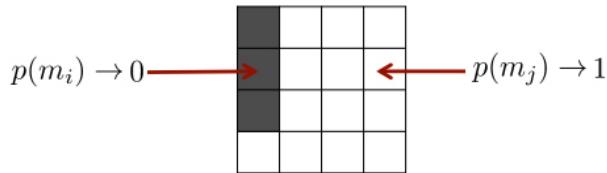


Abbildung 4: Beispiel Grid: belegte Zellen sind durch schwarz markiert, freie Zellen durch weiß. [7]

Occupancy Grid Maps liegen drei Annahmen zugrunde [7]:

1. Die zu einer Zelle korrespondierende Umgebung ist entweder komplett frei, oder komplett belegt.
2. Die Welt ist statisch.
3. Die Zellen sind unabhängig voneinander.

Die Wahrscheinlichkeitsverteilung der gesamten Karte ergibt sich aus dem Produkt über alle Zellen:

$$p(m) = \prod_i p(m_i)$$

Um eine Messung in die Karte einzeichnen zu können, wird ein Sensormodell (cf. Abb. 5) des verwendeten Sensors verwendet. Im Falle des ISAS-Crawlers handelt es sich hierbei um

Ultraschallsensoren. Durch die Messung eines Ultraschallsensors kann eine Entfernung zwischen der Position des Crawlers und des Hindernisses berechnet werden. Auf Basis des bekannten Öffnungswinkels des Ultraschallsensors, der gemessenen Entfernung und der Orientierung des Roboters können die Zellen der Karte ermittelt werden, deren Wahrscheinlichkeitswerte aktualisiert werden müssen.

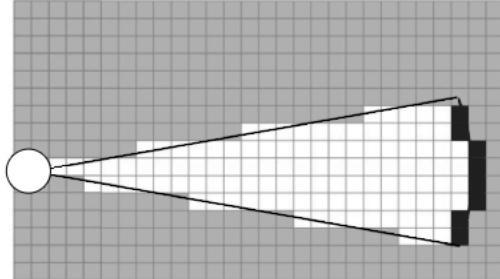


Abbildung 5: Sensormodell eines Ultraschallsensors. Schwarz markierte Zellen gelten als belegt. [7]

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t}) \quad (1)$$

Sind gegeben die Karte m , die Posen des Roboters $x_{1:t}$ und die Sensormessungen $z_{1:t}$ kann auf Basis der Formel 1 die Karte aktualisiert werden [7].

Auf Basis der oben erwähnten Annahmen und unter Anwendung der Bayes'schen Regel ergibt sich Verhältnis zwischen der Wahrscheinlichkeit für Zelle belegt und Zelle frei zu [7]:

$$\frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})} = \frac{p(m|z_t, x_t)}{1 - p(m|z_t, x_t)} \frac{p(m|z_{1:t-1}, x_{1:t-1})}{1 - p(m|z_{1:t-1}, x_{1:t-1})} \frac{1 - p(m_i)}{p(m_i)}$$

Aus Performancegründen werden diese Berechnungen mit Hilfe der Log Odds Notation (cf. Formel 2) berechnet [7].

$$l(x) = \log \frac{p(x)}{1 - p(x)} \quad (2)$$

Die finale Formel, welche zur Aktualisierung der Karte verwendet wird ergibt sich somit zu [7]:

$$l(m_i|z_{1:t}, x_{1:t}) = \underbrace{l(m_i|z_t, x_t)}_{\text{Sensormodell}} + \underbrace{l(m_i|z_{1:t-1}, x_{1:t-1})}_{\text{rekursiver Term}} - \underbrace{l(m_i)}_{\text{Priore}}$$

Durch inkrementelle Updates (cf. Abb. 6(a)) kann nach und nach eine Karte der Umgebung aufgebaut werden. Abschließend lässt sich durch Runden des Wertes einer jeden Zelle eine Maximum Likelihood Map berechnen (cf. Abb. 6(b)) [7].



Abbildung 6: Inkrementelle Updates der Karte und finale Maximum Likelihood Map

4.2 Partikelfilter

Unter Partikelfiltern versteht man Filter, die zur Positionsschätzung viele, möglichst wahrscheinliche Positionen (genannt Partikel) benutzen. Die Motivation für Partikelfilter für SLAM entstammt der *Monte-Carlo Lokalisierung* (MCL) [4]. Hier wird von einem Bewegungsmo dell und einem Sensormodell mit bekannter Karte ausgehend, die wahrscheinlichste Position $\arg\max_{Bel(x_t)}$ geschätzt. Hierbei wird nach Markov angenommen, dass jede Position nur von dem unmittelbaren Vorgänger abhängig ist. $Bel(x_t)$ bezeichnet dabei die Wahrscheinlichkeit, dass sich der Roboter an Position x_t befindet. Die Wahrscheinlichkeit einer Position wird dabei durch die Odometrie und das Sensorergebnis bedingt. Deshalb gilt:

$$Bel(x_t) = p(x_t | z_t, z_{t-1}, \dots, z_0, u_t, u_{t-1}, \dots, u_0)$$

wobei z_t die Sensorergebnisse und u_t die Odometriemessungen zum Zeitpunkt t darstellen. Der Einfachheit gilt nachfolgend $a^t = a_t, \dots, a_0$. Daraus folgt:

$$Bel(x_t) = p(x_t | z^t, u^t) \quad (3)$$

Leider kann diese Formel nicht effizient berechnet werden. Durch probabilistische Umformungen und die Markov-Annahme, kann Formel 3 folgendermaßen umgewandelt werden:

$$Bel(x_t) = \eta p(z_t | x_t) \int_{x_{t-1}} p(x_t | u_t, x_{t-1}) p(x_{t-1} | u^{t-1}, z^{t-1}) dx_{t-1}$$

$p(x_{t-1} | u^{t-1}, z^{t-1})$ ist dabei offensichtlich $Bel(x_{t-1})$. η ist eine Normalisierungskonstante und $p(z_t | x_t)$ ist das Sensor- und $p(x_t | u_t, x_{t-1})$ das Bewegungsmodell, welche experimentell bestimmt werden müssen. Da hierbei jedoch das Integral über den kompletten Zustandsraum gebildet werden muss, ist auch diese Form nicht effizient berechenbar. Aus diesem Grund werden Positionen im kontinuierlichen Raum durch gewichtete Samples approximiert:

$$Bel(x_t) = \{x_t^i, w_t^i\}_{i=1, \dots, N}$$

Damit kann das oben stehende Integral folgendermaßen umgeformt werden.

$$Bel(x_t) = \eta p(z_t | x_t) \sum_N p(x_t | u_t, x_{t-1}) p(x_{t-1} | u^{t-1}, z^{t-1})$$

Logischerweise hängt hier die Genauigkeit von der Anzahl der Partikel ab. Wenn kein Partikel die echte Position des Roboters schätzt, ist die echte Position des Roboters für diesen Zeitpunkt verloren, auch wenn durch spätere Fehlschätzungen diese wiedergefunden werden kann. Jedoch ist es normalerweise eher der Fall, dass bei zunehmenden t , die Positionsschätzungen immer schlechter werden. Dies liegt daran, dass, durch die Streuung des Bewegungsmodells, die Partikel immer weiter gestreut werden, und unwahrscheinliche Partikel behalten werden und immer kleinere Gewichte bekommen. Aus diesem Grund wurde das Resampling eingeführt. Dabei wird zu geeigneten Zeitpunkten aus der Menge der gewichteten Partikel, neue Partikel mit der Wahrscheinlichkeiten ihrer Gewichte gezogen. Ein hochgewichtetes Partikel kann so mehrmals gezogen werden, während ein geringgewichtetes Partikel evtl. gar nicht gezogen wird. Da nach dem Resampling manche Positionen evtl. mehrfach belegt sind, müssen anschließend die Gewichte aller Partikel auf $\frac{1}{N}$ normiert werden.

4.3 Grid-based FastSLAM

Grid-Based FastSLAM ist, ebenso wie reines FastSLAM, eine Implementierung eines SLAM Verfahrens, welche auf dem zuvor beschriebenen Partikelfilter basiert. Beim FastSLAM Algorithmus setzt sich jedes Partikel aus einer Zustandsschätzung, der Pose des Roboters, beliebig vielen Landmarkenpositionen und dem bereits beschriebenen Gewicht zusammen.

Der Zustand eines jeden Partikels wird hierbei durch den Partikelfilter geschätzt. Auf Basis dieser Schätzung können im Anschluss die Positionen der Landmarken bestimmt werden. Durch die Trennung von Positionsschätzung und der Bestimmung der Landmarkenpositionen, also der Aktualisierung der Karte, kann dieses Verfahren effizient durchgeführt werden.

Der Unterschied zu Grid-Based FastSlam besteht nun darin, dass jedes Partikel nicht eine Anzahl von Landmarkenpositionen beinhaltet, sondern jeweils eine eigene GridMap (cf. Sec. 4.1). Im Prädiktionsschritt wird nun jeweils die Zustandsschätzung eines jeden Partikels auf Basis des Systemmodells aktualisiert, im Korrekturschritt werden anhand der von den Sensoren durchgeführten Messungen und der jeweiligen GridMap die Gewichte der Partikel bestimmt.

Abbildung 7 veranschaulicht drei mögliche Partikel mit der jeweils dazugehörigen GridMap. Bei genauem Hinsehen erkennt man, dass sich jedes Partikel an einer eigenen Position befindet. Zusätzlich unterscheiden sich die drei erstellten GridMaps voneinander.

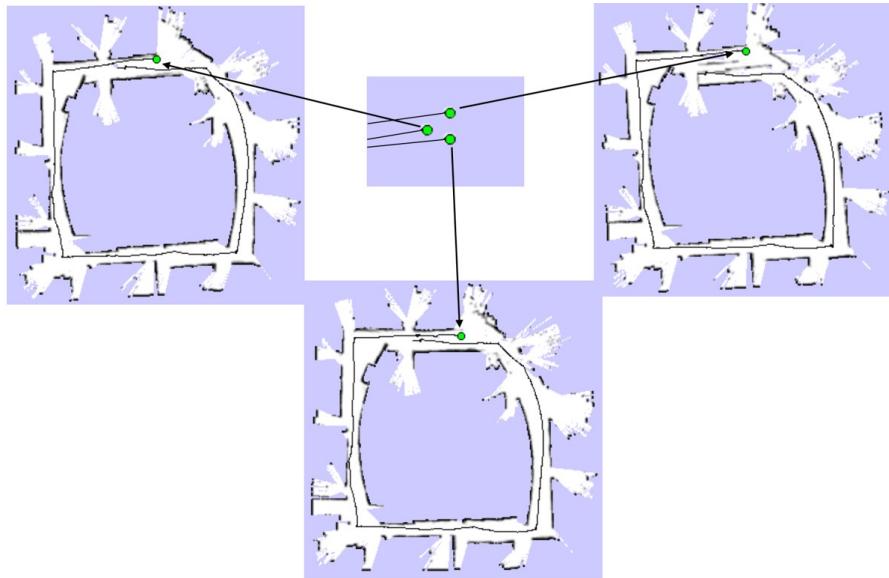


Abbildung 7: Darstellung dreier Partikel mit der jeweiligen GridMap [6]

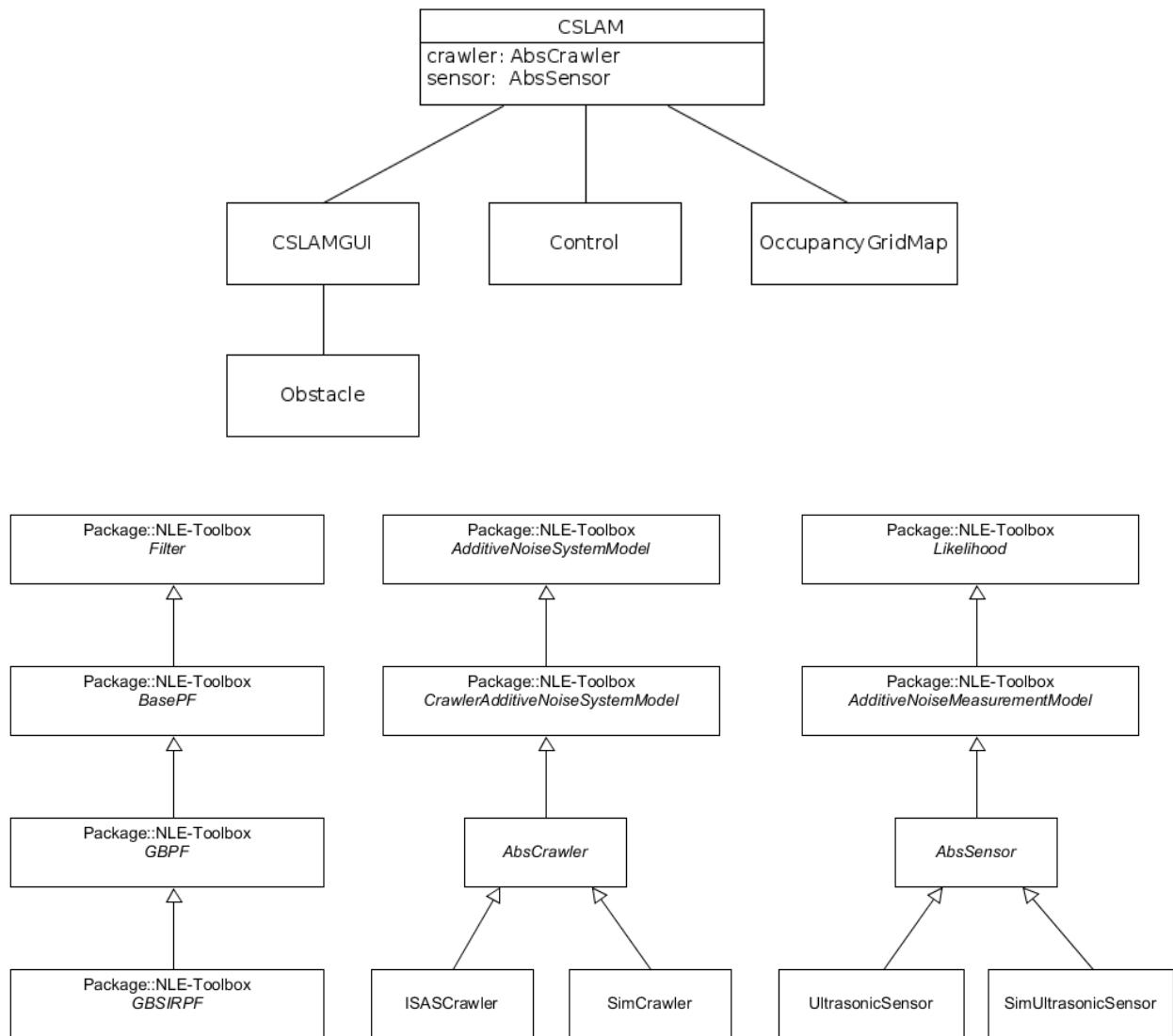
5 Simulationsumgebung

Die Simulationsumgebung kann verwendet werden, um eine virtuelle Umgebung mit Hindernissen zu erstellen und darin mit einem simulierten Crawler und simulierten Sensoren das SLAM Verfahrens nachzubilden. Das Ergebnis des SLAM Verfahrens kann dann in der Simulationsumgebung graphisch dargestellt werden. Während des Entwurfs wurde darauf geachtet, dass die einzelnen Komponenten, z.B. der Crawler oder ein Sensor, austauschbar sind. So lässt sich die grafische Oberfläche der Simulationsumgebung auch mit dem ISAS-Crawler verwenden und mit dessen Sensoren ein SLAM Verfahren mit realen Werten durchführen. Dies wird in der Übersicht über den Simulationsaufbau (s. Abschnitt 5.1) deutlich. Im darauf folgenden Kapitel (s. Abschnitt 5.2) wird auf die Bedienung eingegangen. Dazu wird die Oberfläche und die darauf vorhandenen Button näher beschrieben.

5.1 Simulationsaufbau

Der Aufbau der Simulation ist in Abbildung 8 dargestellt.

Bei *CSLAM* handelt es sich um die Hauptklasse, die beim Programmstart aufgerufen wird und die einen *Crawler* sowie dessen *Sensoren* als Eingabeparameter erhält. Bei der Eingabe kann es sich sowohl um einen echten *ISASCrawler* als auch um einen simulierten *SimCrawler* handeln, die Klasse muss nur vom abstrakten *AbsCrawler* erben. Das gleiche gilt auch für das Array der Sensoren. *CSLAM* erstellt daraufhin die *GUI*. Falls simulierte Sensoren verwendet werden können nun die Hindernisse (*Obstacle*) in der *GUI* eingezeichnet werden. Außerdem erhält *CSLAM* die Tastatureingaben, die durch die *Control* Klasse eingelesen werden. Das SLAM-Verfahren wird dann mit Hilfe der *OccupancyGridMap* und einem *PartikelFilter* durchgeführt.

**Abbildung 8:** UML Diagramm der Softwarearchitektur

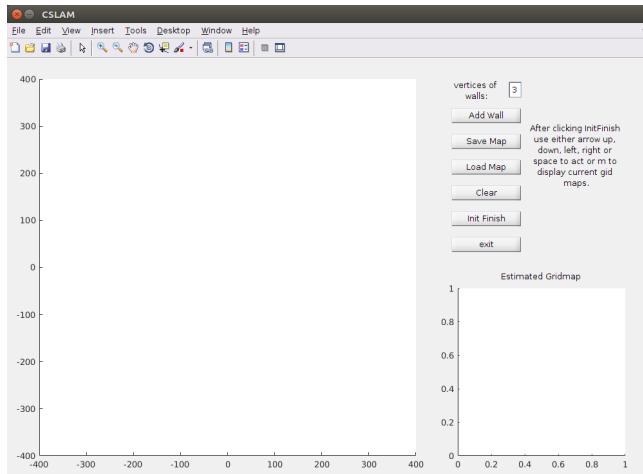


Abbildung 9: GUI der Simulation

Ein Teil der Klassen, darunter der Filter, das Systemmodell und das Messmodell, werden durch die Nonlinear-Estimation-Toolbox (NLE-Toolbox)¹ [8] bereitgestellt. Die NLE-Toolbox enthält verschiedene Filter, Modelle und weitere Hilfsmittel, die wir anpassen und in unserem Programm verwenden können. Weitere Details werden im Abschnitt 6.1 beschrieben.

5.2 Benutzung der grafischen Benutzeroberfläche

Um das Programm bedienen zu können, wird eine Graphical User Interface (GUI) zur Verfügung gestellt. Mit ihr kann ein realer oder simulierter Crawler gesteuert werden, außerdem werden die Ergebnisse des SLAM Verfahrens anschaulich dargestellt. Des weiteren werden auch die reale Route, die geschätzte Route und die Occupancy Grid Map, die am wahrscheinlichsten ist, in der GUI abgebildet. Beim Programmstart sollte die Oberfläche wie in Abbildung 9 aussehen.

Die Funktionen der einzelnen Steuerelemente werden im Folgenden ausführlich erklärt.

- *Vertices of walls:* In diesem Textfeld kann die Anzahl an Eckpunkten, die das als nächste erstellte Hindernis besitzt, konfiguriert werden. Das Hindernis wird anschließen vom Benutzer mit der *Add Wall* Schaltfläche eingezeichnet. Der vorgegebene Zahlenwert ist drei.
- *Add Wall:* Nach dem Betätigen dieser Schaltfläche kann der Benutzer neue Hindernisse auf der Karte einzeichnen. Hierfür zeichnet der Benutzer Eckpunkte in die Karte ein, die gewählten Punkte werden durch eine Linie verbunden. Diese Linie stellt die Hindernisse in Form von Wänden dar, die das Signal des simulierten Ultraschallsensors reflektieren. Die Anzahl der Eckpunkte, die der Benutzer setzen kann, ist dabei von dem Wert von *Vertices of walls* abhängig. Der Benutzer kann diese Schaltfläche mehrmals verwenden, um weitere Hindernisse hinzuzufügen.

¹ Jannik Steinbring, <http://isas.uka.de/User:Steinbring>

- *Save Map*: Mit dieser Schaltfläche kann der Benutzer die angezeigte Karte in einem Ordner seiner Wahl als *.m Datei speichern.
- *Load Map*: Mit dieser Schaltfläche kann der Benutzer eine zuvor gespeicherte Karte laden. Diese
- *Clear*: Löscht alle bisher eingezeichneten Hindernisse.
- *Init Finish*: Sobald alle Hindernisse auf der Karte eingezeichnet wurden, kann mit Hilfe dieses Buttons das Erstellen der Hindernisse beendet werden. Der Text des Buttons ändert sich nach dem ersten Klick darauf zu *Init Done*. Des Weiteren können nun die bisher genannten Schaltflächen nicht mehr verwendet werden.
- *Exit*: Beendet das Programm.

Neben diesen Steuerelementen ist eine kurze Anleitung zur Steuerung des Roboters zu finden. Nachdem die Erstellung der Karte abgeschlossen ist, bzw. der Init Finish Button betätigt wurde, beginnt die Simulation des SLAM Verfahrens. Der Benutzer bewegt den Crawler mit den Pfeiltasten. Die Bewegungsbefehle werden gesammelt, bis der Benutzer die Leertaste betätigt. Nach dem Betätigen der Leertaste werden die Befehle je nach Umgebung entweder vom simulierten Crawler verarbeitet oder an den ISAS-Crawler geschickt, der diese dann ausführt.

Ebenfalls auf der GUI, in der Box unterhalb der Schaltflächen, wird die geschätzte Karte angezeigt. Hier ist die Karte des Partikels zu sehen, das am wahrscheinlichsten die aktuelle Pose approximiert.

6 Implementierung

Anschließend an die Entwicklung der Simulationsumgebung, wurde FastSLAM implementiert.

Zu Beginn wird der Aufbau und die Funktionsweise der NLE-Toolbox (s. Abschnitt 6.1), sowie getätigte Anpassungen beschrieben, mithilfe welcher FastSLAM implementiert wurde.

Daran schließt die Definition des System- sowie Messmodells (s. Abschnitt 6.2 bzw. 6.3) an, welche Bezug zur Toolbox nehmen. Ebenfalls wird die Ansteuerung der Sensoren erklärt, so wie die Probleme die durch Sensorrauschen aufgetreten sind und deren Behebung.

Abschließend wird eine Ausführliche Beschreibung der CSLAM-Klasse (s. Abschnitt 6.4) durchgeführt, welche auch die Bedienung des Programms beinhaltet.

6.1 NLE-Toolbox

Ein Teil der Funktionen, die die Software nutzt werden durch die *Nonlinear-Estimation-Toolbox* [8] bereitgestellt. Innerhalb dieser sind bereits eine Vielzahl an Rauschverteilungen und nicht-lineare Filter- bzw. Schätzerklassen definiert. Die Software nutzt aber mit *Sampling Importance Resampling Particle Filter* (SIRPF) nur einen kleinen Teil der Toolbox aus. Zu den bereitgestellten Funktionen gehören unter anderem:

- Eine Definition von abstrakten Klassen für das Systemmodell und das Sensormodell mit additivem Rauschen. Die genauen Implementierungen dieser abstrakten Klassen und die dafür benötigten Definitionen des Rauschens folgen in den Kapiteln 6.2 und 6.3
- Eine fertige Klassenstruktur des Partikelfilters
- Vordefinierte Klassen für mehrdimensionales Gaußsches Rauschen

6.1.1 Änderungen an der Toolbox

Insebsondere die Klassenstruktur für den Partikelfilter musste an unsere Problemstellung angepasst werden. Ursprünglich nutzte der Partikelfilter kontinuierliche Karten, daher mussten die Klassen *BasePF*, *PF* und *SIRPF* um die Verwaltung von Occupancy-Grid-Maps erweitert werden. Dazu wurde ein zusätzlicher Verktor angelegt, der Instanzen der von uns angelegten Klasse Occupany-Grid-Map (s. Kapitel 4.1) enthält. Dabei verweist der Index auf den dazugehörigen Partikel. Um keine der bereits existierenden und funktionierenden Klassen zu verändern, haben wir neue Klassen erstellt, die jeweils den Präfix *GB* (Grid-Based) nutzen.

Außerdem haben wir uns entschieden, den Roboter und alle dazugehörigen Partikel immer von Position (0, 0, 0) aus starten zu lassen. Die Karte wird anschließend um diese Position herum aufgebaut und der Roboter sucht seine gelaufene Route in diesem Koordinatensystem. Dazu wurde eine neue Klasse für Diracsches Rauschen für den initialen Zustand erstellt. Jedes

Partikel und der Roboter ziehen sich eine "zufällige" Position dieses Rauschens, um so ihre Startpose zu definieren.

Schließlich mussten die für das Gewichtsupdate und Partikelvorhersage relevanten Funktionen angepasst werden. Da wir uns für die Verwendung einer *Maximum-Likelihood*-Methode für das Gewichtsupdate entschieden haben, finden sich genauere Informationen hierzu in Kapitel 6.3. Weitere Informationen zur Partikelvorhersage finden sich in Kapitel 6.2.

6.1.2 Funktionen der Toolbox

Alle filterinternen Funktionen, die unser Programm nutzt, sind in der Toolbox bereits definiert. Dazu gehört insbesondere die Verwaltung von N Partikeln, die Verwaltung der dazugehörigen Gewichte und das *Resampling*. Da wir eine modifizierte Version des *Sampling Importance Resampling* nutzen, bestimmt ein heuristischer Wert, wann ein Resampling durchgeführt werden muss. Im Fall der Non-Linear-Estimation-Toolbox ist dies die *Estimated Sample Size* (ESS). Diese ist definiert durch:

$$\text{NormalizedESS} = \frac{1}{\text{numParticles} * \sum \text{ParticleWeights}^2}$$

Ist dieser Wert < 0.5 , dann wird ein Resampling durchgeführt. Durch die Verwendung der quadrierten Gewichte wird der ESS klein, sobald viele Partikel geringe Gewichtswerte aufweisen. Haben alle Partikel aber das gleiche Gewicht, so wird der ESS maximiert. Es stellt also eine Heuristik für die Gewichtsverteilung der Partikel dar.

Im Resampling werden dann anhand der akkumulierten Gewichte, multipliziert mit der Anzahl an Partikeln, neue Partikel aus der Menge der bereits existierenden N Partikel gezogen. Dazu wird ein Zähler immer weiter erhöht, bis eine aus dem Intervall $[0, 1]$ gezogene Zufallszahl größer als das akkumulierte Gewicht des Partikels an stelle des Zählers ist. Dies entspricht einem zufälligen Ziehen von Partikeln mit ihren Gewichten als Wahrscheinlichkeiten. So werden nach und nach N neue Partikel bestimmt, mit denen anschließend das SLAM Verfahren fortgesetzt wird. Da unsere Klassen um Gridmaps erweitert wurden, werden hier ebenfalls die Karten der Partikel gezogen.

6.2 Systemmodell

Im Prädiktionsschritt des Partikelfilters wird anhand des kinematischen Modells eine neue Positionsschätzung für jedes Partikel berechnet. Dieses Modell wird auch Systemmodell genannt und ist neben dem Messmodell wesentlicher Bestandteil des Partikelfilters und damit des SLAM Verfahrens.

Das Systemmodell stellt die Modellierung des ISAS-Crawlers dar. Ziel ist es, das Bewegungsverhalten des Crawlers, also die Bewegung an für sich und das Systemrauschen möglichst genau zu modellieren.

Die verwendete Nonlinear-Estimation-Toolbox stellt bereits das Grundgerüst für die Implementierung eines Systemmodell dar. Für das Modell wird daher auf die Klasse *AdditiveNoiseSystemModel* der Toolbox zurückgegriffen.

Um das Bewegungsverhalten des ISAS-Crawler zu modellieren wurde folgendes nichtlineares, zeitdiskretes Modell [9] implementiert:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} \cos(\theta_k) & \sin(\theta_k) & 0 \\ -\sin(\theta_k) & \cos(\theta_k) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta x_k \\ \Delta y_k \\ \Delta \theta_k \end{bmatrix} + \begin{bmatrix} w_k^x \\ w_k^y \\ w_k^\theta \end{bmatrix}$$

Dieses Modell gilt für den Fall, dass die Translationen vor den Rotationen ausgeführt werden. Anhand von Bewegungsbefehlen und einer gegebenen Startpose $\underline{x}_t = [x, y, \theta]^T$ wird die Pose \underline{x}_{t+1} berechnet. Das Systemrauschen wurde als additives Rauschen durch die Rauschterme w_k^x , w_k^y und w_k^θ beschrieben.

Um das Systemrauschen möglichst real beschreiben zu können, wurden zunächst die unterschiedlichen Bewegungen, welche der Crawler ausführen kann (cf. Abb. 10(a)) analysiert und vermessen. Anhand dieser Ergebnisse konnten die mittleren Schrittweiten, sowie deren Varianz für jede Bewegung aufgestellt werden.

Die Ergebnisse von jeweils 200 simulierten Ergebnissen der Vorwärts- und Seitwärtsbewegungen sind in Abbildung 10(b) dargestellt.

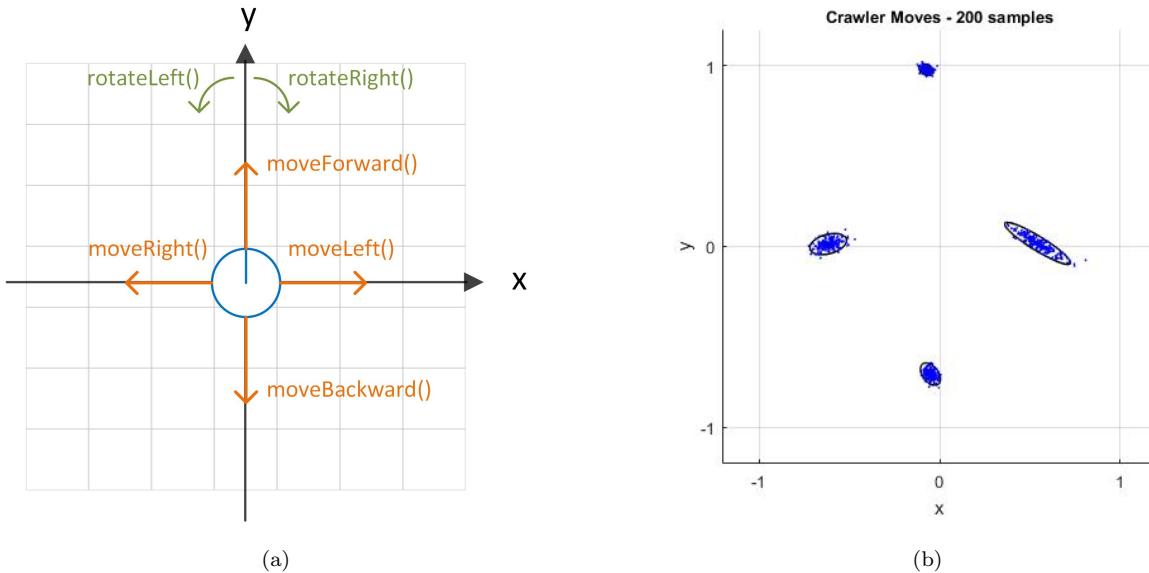


Abbildung 10: Systemmodell: Bewegungsrichtung in Weltkoordinaten und Plot mit 200 Samplebewegungen für die Bewegungsrichtungen vorwärts, rückwärts und seitwärts.

Aus Abbildung 10(b) wird ersichtlich, dass der Crawler stark unterschiedlich verrauschte Bewegungen durchführt. Bewegungen nach vorne oder zurück erfolgen relativ zuverlässig, Bewegungen nach links, oder vor allem nach rechts hingegen sehr verrauscht.

Das Systemmodell definiert daher für jede Bewegung eine eigene Schrittweite, sowie eine eigene Kovarianzmatrix. Die entsprechende Klasse in der verwendeten Toolbox wurde durch die Klasse *CrawlerAdditiveNoiseSystemModel* angepasst und erweitert. Je nach Bewegung wird die Positionsschätzung durch die entsprechenden Rauschterme verrauscht.

6.3 Messmodell

Das Messmodell übernimmt in der Implementierung mehrere Aufgaben:

- Beschreibung des Sensorrauschen
- Ansteuerung der Sensoren
- Verarbeitung der Messdaten (Toolbox)

6.3.1 Sensorrauschen

Als Sensorrauschen wird im Folgenden der Fehler bezeichnet, den der Sensor bei einer Messung macht. Zur Bestimmung des Fehlers wurde der Fehler eines Sensors über mehrere Aufnahmen hinweg betrachtet (Details siehe Kapitel 6.3.2). Die Einschränkung auf einen Sensor wurde durchgeführt, da alle vier verbauten Ultraschallsensoren baugleich sind. Um die gemessenen Ergebnisse robust gegenüber Fehlmessungen zu machen, wird zusätzlich immer der Mittelwert über zehn Aufnahmen gebildet.

Insgesamt wurden pro Entfernung 20 Messungen durchgeführt, wobei folgende Entfernungen (in cm) betrachtet wurden: 2, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200. Das heißt, es wurden $13 * 20 = 260$ Messungen bzw. $260 * 10 = 2600$ Aufnahmen durchgeführt.

Bei einer ersten Sichtung der Messergebnisse fiel auf, dass trotz der Mittelwertbildung über zehn Aufnahmen, bzw. einer Messung, das Verfahren sehr anfällig für Ausreißer ist. Daher wurden zusätzlich die Daten mithilfe einer Outlier-detection gefiltert und mit den Rohdaten, sowie der realen Entfernung verglichen. Dies wurde mittels der 1.5-fachen Standardabweichung durchgeführt.

6.3.2 Bestimmung des Fehlers

Für die Bestimmung der in 6.3.1 beschriebenen Werte wurde der Crawler in entsprechendem Abstand vor eine Wand gestellt und anschließend 20 mal 10 Aufnahmen durchgeführt. Berechnet wurde für jede Messung der Mittelwert, die Varianz sowie die Standardabweichung auf den Roh- und den vorher gefilterten Daten. Das heißt, es wurde für jede Messung der Mittelwert, die Varianz sowie die Standardabweichung und anschließend den Mittelwert der drei genannten, über die Messungen einer Distanz, berechnet. Anzumerken ist, dass aufgrund des Öffnungswinkels (15°) sowie der Höhe des Sensors (ca. 8,3 cm), die Schallwelle nach ca. 63

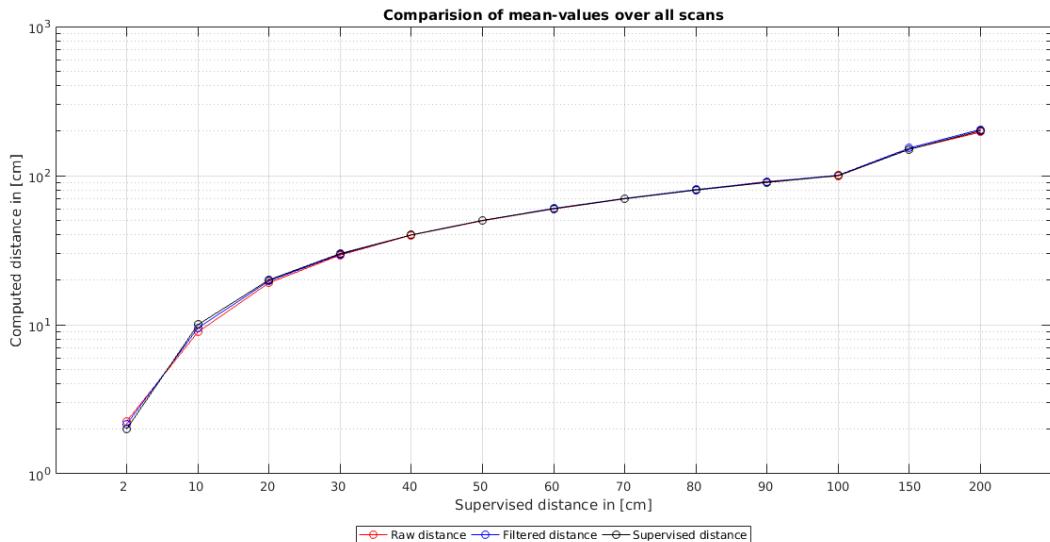


Abbildung 11: Vergleich des Mittelwerts über alle Messungen der Rohdaten, gefilterter Messungen und der realen Distanz

cm auf den Untergrund trifft. Der Sensor liefert jedoch bei Distanzen über dieser Grenze nicht den Abstand von 63cm zurück, was vermuten lässt, das aufgrund des Auftreffwinkels nicht ausreichend Schall reflektiert wird, um die Messung zu verfälschen. Eine genaue Analyse dessen wurde jedoch, aufgrund des Umfangs nicht durchgeführt.

In Abbildung 11 ist ein Vergleich des Mittelwerts der realen und der gemessenen Distanz, roh, sowie gefiltert dargestellt. Um die Lesbarkeit zu erhöhen wurde die y-Achse logarithmisch skaliert. Zu entnehmen ist, dass sich die Werte unterhalb einer Distanz von 20 cm minimal unterscheiden, oberhalb ist (fast) keiner mehr zu erkennen.

Anschließend wurde die Varianz der Roh- sowie gefilterten Daten verglichen (s. Abb. 12). Hierbei wurde der Mittelwert über die Varianz aller Messungen einer Distanz berechnet. Die y-Achse ist ebenfalls logarithmisch skaliert. Hier ist zu entnehmen, dass die gefilterten Daten eine deutlich geringere als die Rohdaten, besitzen.

Abschließend wurde der gemachte Fehler, d.h. die Differenz zw. den Roh- bzw. gefilterten Daten und der realen Entfernung betrachtet (s. Abb. 13 und 14). Auch hier wurden die y-Achsen logarithmisch skaliert. In der ersten Abbildung ist die absolute Differenz zwischen Roh- und gefilterten Daten dargestellt. Hier ist zu sehen, dass die Werte ab einer Distanz von 40 cm zunehmen. Die zweite Abbildung zeigt die Differenz zw. den Roh- bzw. gefilterten Daten und der „realen“ Distanz. Hier ist auffällig, dass diese erst ab einer Distanz von 90 cm (mit der Annahme, dass der Wert für 10 cm verrauscht ist bzw. einer gewissen Messungenauigkeit unterliegt) über 1 cm liegen.

Aus den obigen Beobachtungen lässt sich somit schlussfolgern:

1. Der endgültig berechnete Wert liegt nahe bei dem realen, unabhängig ob roh oder gefiltert.

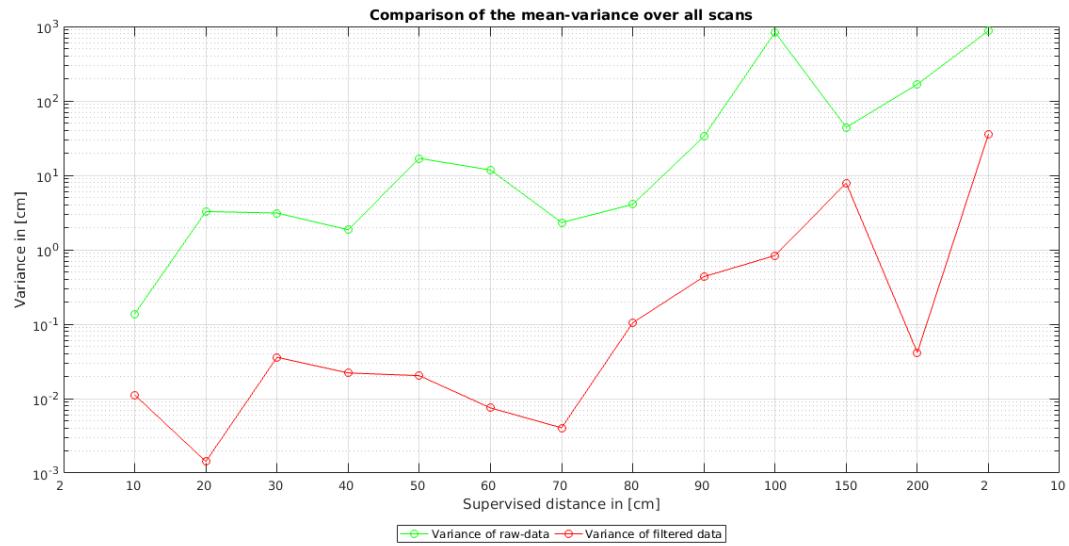


Abbildung 12: Vergleich der Varianz der Roh- sowie gefilterten Daten

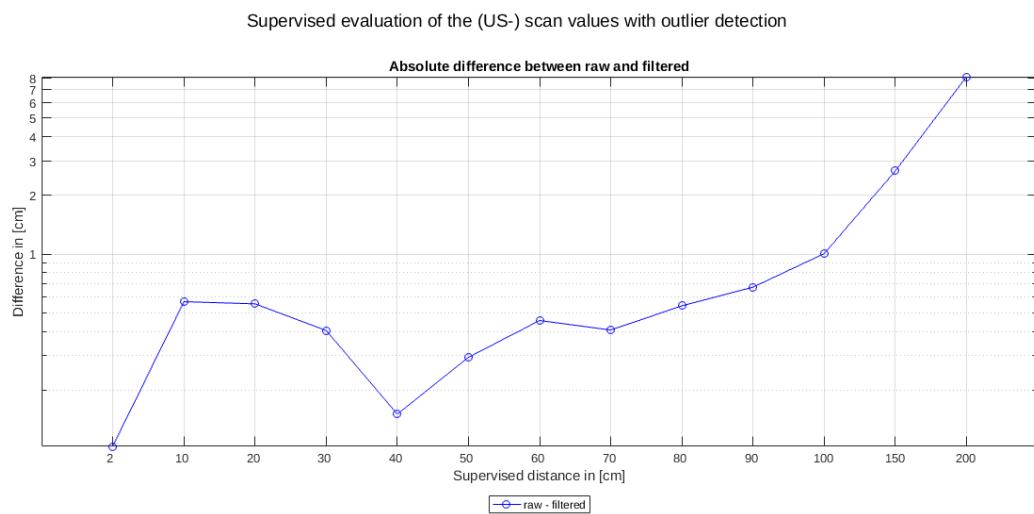


Abbildung 13: Absolute Differenz der Roh- und gefilterten Daten im Vergleich zur „realen“ Entfernung

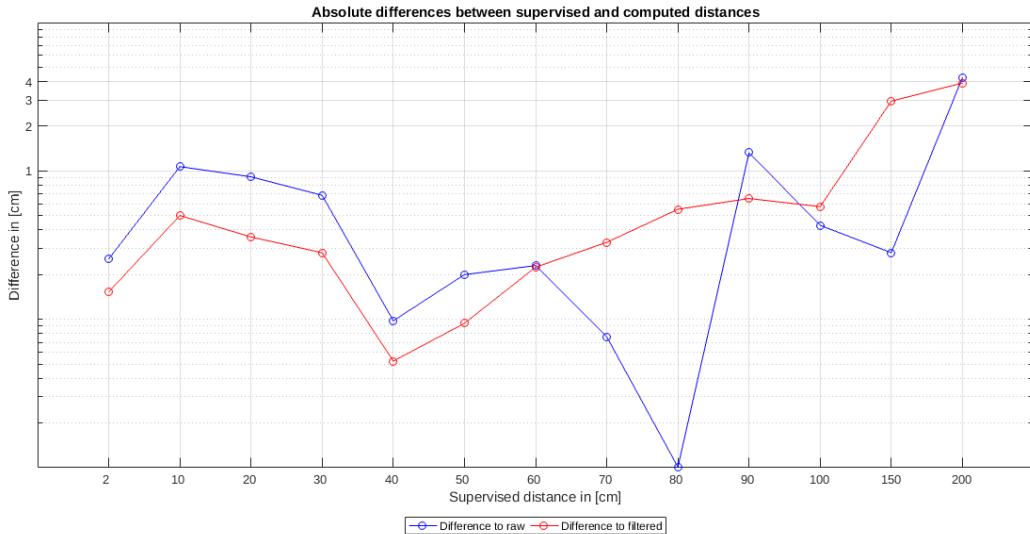


Abbildung 14: Absolute Differenz der Roh- sowie gefilterten Daten zur „realen“ Entfernung

2. Die Varianz der gefilterten Daten ist deutlich geringer als die der ungefilterten. Dies ist aber dahingehend nicht verwunderlich, da die zur Filterung verwendete Standardabweichung von der Varianz abhängig ist.
3. Die Varianz nimmt mit steigendem Abstand zu. Daher wäre bezüglich des simulierten Sensors ein multiplikatives, statt additives Rauschen, zu bevorzugen. Jedoch wurde aufgrund des Umfanges des Praktikums auf die Bestimmung eines multiplikativen Rauschens verzichtet und stattdessen ein additives verwendet.
4. Ab einer gewissen Distanz nimmt der Unterschied zwischen den Roh- und den gefilterten Daten zu.
5. Bis zu einer Distanz von 60 cm ist das Ergebnis mit Filterung besser, anschließend ohne.

6.3.3 Ansteuerung der Sensoren

Die Ansteuerung der Sensoren erfolgt unabhängig davon, ob es sich um einen real vorhandenen Sensor oder einen simulierten handelt (Details s. Kapitel 3.3).

Der reale sowie der simulierte Sensor unterscheiden sich intern jedoch im Falle der Durchführung einer Messung.

Der reale Sensor führt folgende Schritte durch um eine Messung zu tätigen:

1. Senden des HTTP-Befehls für das Durchführen von zehn Messungen.
2. Outlierdetektierung und entsprechende Filterung.
3. Mittelwertbildung.

Zu Beginn wird, mithilfe einer Matlab-Bibliothek, ein entsprechender HTTP-Befehl generiert und an den Crawler gesendet (1). Hierbei handelt es sich um einen GET-Befehl, der an die Bridge auf dem Crawler, der eine REST-Schnittstelle zur Verfügung stellt, gesendet wird. Ein Aufruf der zehn Messungen durchführt und den Sensor mit der Sensor-ID 0 verwendet, könnte somit wie folgt aussehen (Vgl s. 3.3.2):

`http://arduino.local/arduino/US,0,10`

Nachdem die Ergebnisse der Messungen vorliegen, wird eine Outlierdetektierung durchgeführt (2). Hierbei werden alle Messungen entfernt, die vom Mittelwert weiter als die drei-fache Standardabweichung entfernt sind. Anschließend wird über die verbleibenden Messungen der Mittelwert gebildet (3), welcher als Messergebnis zurückgegeben wird.

Vorgehen beim simulierten Sensor:

1. Gerade bzw. Kegel einzeichnen.
2. Schnittpunkt im Objekt berechnen.
3. Entfernung zehn Mal aus Schnittpunkt und Standort bestimmen.
4. Die Entfernungen jeweils verrauschen.
5. Berechnung des Durchschnitts.
6. Keine Outlierdetektierung.

6.3.4 Bezug zur Toolbox

Im Gegensatz zum Systemmodell (s. Kapitel 6.2), werden die entsprechenden Teile der Toolbox überschrieben. Hierbei handelt es sich um:

1. simulate
2. update

Die simulate-Funktion ist für das Berechnen der Messwerte zuständig, wobei es hierfür auf ein zu definierendes Modell zurückgreift. Aufgrund des „starren“ Gerüsts, das aufgrund der Toolbox eingehalten werden muss, erschien es im Rahmen dieses Praktikums sinnvoll, in diesem Fall nicht auf die Toolbox zurück zu greifen, sondern die Berechnung der Werte zu implementieren (Details zur Berechnung: s. Kapitel 6.3.3).

Die Update-Funktion berechnet, wie wahrscheinlich es ist, dass, gegeben ein Partikel und ein Messung, das Partikel diese Messung „durchgeführt“ hat. Diese Berechnung wird ebenfalls nicht von der Toolbox durchgeführt. Als Basis für diese Berechnung wurde die Annahme getroffen, dass eine Messung die Karte (Gridmap) des Partikels so wenig ändern soll wie möglich. Dies

wurde durch die Überlegung motiviert, dass je größer die Änderung der Karte, umso größer die Unsicherheit des aktuellen Partikels. Das heißt, wenn eine Messung die Karte nur gering ändert lässt sich annehmen, dass es sehr wahrscheinlich ist, dass das Partikel an der von ihm angenommenen Pose diese Messung durchgeführt hat. Bei starker Änderung der Karte ist anzunehmen, dass die Sicherheit der Positionsschätzung des Partikels bezüglich der Karte einer größeren Unsicherheit unterliegt, da ansonsten keine Änderung nötig wäre. Auf Basis dieser Annahme wurde folgendes Vorgehen festgelegt:

1. Berechnung des mean-squared-errors (MSE) auf Basis der Änderung der Karte durch die aktuelle Messung.
2. Berechnen der Gewichte anhand des MSE.
3. Normalisieren der Gewichte.

Die Änderung der Karte wird mittels MSE berechnet. Das heißt, es wird berechnet, wie sich die Karte aufgrund der gegebenen Messung ändert und mit der alten sowie der „neuen“ der MSE berechnet. Anschließend wird das Gewicht wiefolgt berechnet:

$$\text{Gewicht} = \frac{1}{1 + \text{MSE}}$$

Durch diese Berechnung wird erreicht, dass ein kleiner MSE ein großes Gewicht erhält und umgekehrt. Abschließend werden die Gewichte normalisiert, d.h. sie ergeben in Summe eins.

6.4 CSLAM

Um die einzelnen Teile zu steuern haben wir eine Klasse definiert, die gemäß des *Model-View-Controller* Patterns die Rolle des Controllers übernimmt. Sie steuert den View, welcher in unserem Fall die GUI (s. Kapitel 5.2) darstellt und die Modelle, die in unserem Fall das Systemmodell (s. Kapitel 6.2) und das Messmodell (s. Kapitel 6.3) darstellen. Diese Struktur garantiert, dass einzelne Teile des Systems nicht zu mächtig werden und dass wir keine Co-Abhängigkeiten im Programm haben.

Der Controller nimmt dabei bei der Initialisierung ein Objekt der Klasse *AbsCrawler* und ein Objekt der Klasse *AbsSensor* entgegen. Durch die Verwendung von abstrakten Klassen, kann CSLAM mit den simulierten, oder den realen Objekten gestartet werden. Während der Initialisierung wird ebenfalls der Filter, sowie die GUI erstellt. Im Falle des simulierten Crawlers wird der Benutzer dazu aufgefordert, seine Karte zu erstellen und Hindernisse zu definieren. Diese Hindernisse werden dann an den simulierten Sensor weitergegeben, damit dieser Abstandsmessungen in der simulierten Umgebung erzeugen kann. Nach der Initialisierung fragt der Controller in einer *While*-Schleife die Tastatureingaben des Benutzers ab. Dabei sind folgende Tastaturbelegungen vorgegeben, die auf verschiedene Funktionen des Systemmodells 6.2 zugreifen:

- **”Arrow up”-Taste:** Mit dieser Taste wird ein *Move-Forward*-Befehl an das Systemmodell gesendet
- **”Arrow down”-Taste:** Mit dieser Taste wird ein *Move-Backward*-Befehl an das Systemmodell gesendet
- **”Arrow left”-Taste:** Mit dieser Taste wird ein *Rotate-Left*-Befehl an das Systemmodell gesendet
- **”Arrow right”-Taste:** Mit dieser Taste wird ein *Rotate-Right*-Befehl an das Systemmodell gesendet
- **”,,”-Taste:** Mit dieser Taste wird ein *Move-Left*-Befehl an das Systemmodell gesendet
- **”.,”-Taste:** Mit dieser Taste wird ein *Move-Right*-Befehl an das Systemmodell gesendet
- **”Space”-Taste** Mit dieser Taste wird ein *Execute-Commands*-Befehl an das Systemmodell gesendet
- **”esc”-Taste** Diese Taste beendet die Schleife und damit das Programm

Mit den Pfeil- und ”.”- und ”,”-Tasten werden Befehle in einer *Queue* im Systemmodell gespeichert. Mit der Leertaste werden die Befehle in der Queue ausgeführt. CSLAM ruft dann außerdem Funktionen des Sensors und des Filters auf, um die Partikel und deren Gewichte aufgrund der Messung und der Position zu aktualisieren und diese dann in der GUI anzuzeigen. Als neue echte Position wird im simulierten Fall einfach eine zusätzliche Position geschätzt. Im realen Fall wird hier die Kamera angesteuert und eine reale Position errechnet. Die echte und die geschätzte Position des Crawlers werden anschließend in zwei Matrizen gespeichert. Diese werden dynamisch vor jeder Befehlsausführung des Roboters um eine Spalte erweitert. Des Weiteren wird das Partikel mit dem höchsten Gewicht an die GUI weitergegeben, sodass die dazugehörige Gridmap angezeigt werden kann. Nach Beendigung der While-Schleife werden die Routen in der GUI angezeigt und anschließend das Programm beendet.

7 Evaluation

7.1 Aufnahmesysteme

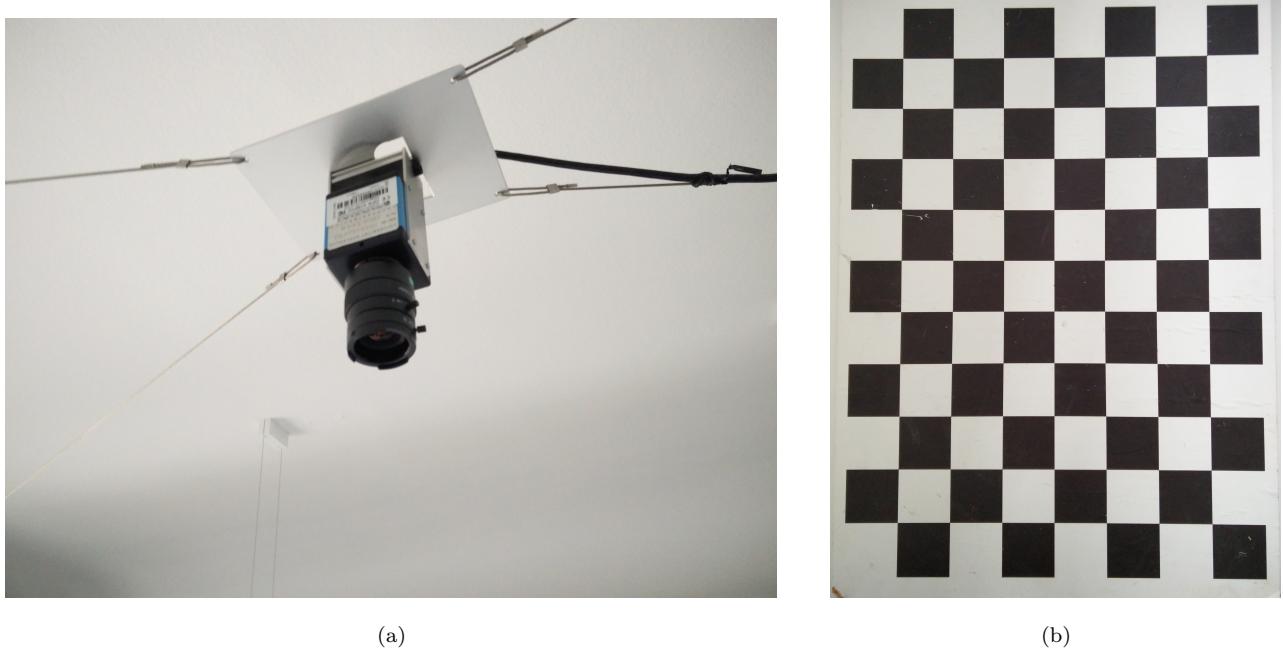
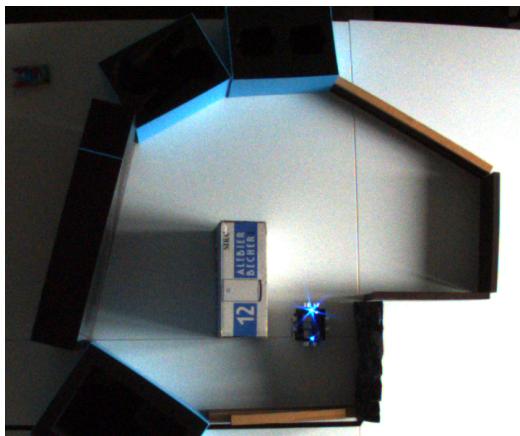


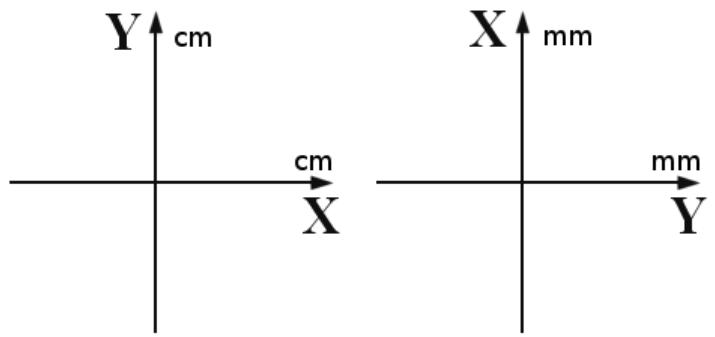
Abbildung 15: Kamera und Kalibrierungsschablone mit Schachbrettmuster

Um im realen Anwendungsfall die echte Position des Roboters zu bekommen, benutzen wir eine *Imaging Source DFx 41BF02* Deckenkamera. Diese Deckenkamera berechnet anhand der LEDs des Crawler die Pose des Roboters. Damit diese Berechnung funktioniert, musste die Kamera intrinsisch und extrinsisch kalibriert werden. Hierfür wurde die Camera Calibration Toolbox für Matlab benutzt [2]. Diese Toolbox stellt dabei Funktionen bereit, um Bilder mit der Kamera aufzunehmen und für die Kalibrierung weiter zu verarbeiten. Eine Kamerakalibrierung ist dafür zuständig, Verzerrungen eines Bildes zu entzerren, welche durch eine Objektiv-Kamera verursacht wurden. Außerdem ist eine Kamerakalibrierung notwendig, um eine Rekonstruktion eines Weltmodells durchzuführen. Intrinsische Parameter sind dabei für die Umrechnung von Pixelkoordinaten in Kamerakoordinaten ausschlaggebend. Extrinsische Parameter dagegen beschreiben die Ausrichtung der Kamera im Weltmodell, und sind daher für die Umrechnung von Kamerakoordinaten in Weltkoordinaten notwendig. Um eine gute intrinsische Kalibrierung zu garantieren, wurden mehrere Bilder mit einer Kalibrierungsschablone (s. Abb. 15(b)) aus unterschiedlichen Blickwinkeln aufgenommen. Abbildung 15(b) zeigt eine solche Schablone mit Schachbrettmuster. Anschließend mussten die Position und Größe der Schablone und die Größe und die Anzahl der Quadrate manuell eingegeben werden, damit die Toolbox die intrinsischen Parameter berechnet. Zu den intrinsischen Parametern gehören *Focal length*, *Principal point*, *Skew* und *Distortion*. Bei der extrinsischen Kalibrierung galt es zu beachten, dass sich die LEDs, die zur Positions berechnung benutzt werden, 9,5cm über der Tischplatte befinden. Darum wurde wieder eine Kalibrierungsschablone genutzt, die jedoch um den gegebenen Abstand erhöht

auf den Tisch gelegt wurde. Da auch hier wieder Position und Größe der Schablone und Größe und Anzahl der Quadrate angegeben werden mussten, ist die Kamera nun in der Lage, von Kamerakoordinaten auf Weltkoordinaten umzurechnen. Somit gehören die für diese Umrechnung notwendigen Parameter *Translation vector*, *Rotation vector* und *Rotation matrix* zu den extrinsischen Parametern. Die Weltkoordinaten werden in mm zurückgegeben und haben ihren Ursprung in der, bei der Kalibrierung definierten Position. Eine Position des Roboters wird



(a)



(b)

Abbildung 16: Kamerabild und Zusammenhang von Roboterkoordinaten und Weltkoordinaten

dabei anhand der blauen LEDs ermittelt. Hierzu wird in der aufgenommenen Bildmatrix nach maximalen Blauwerten gesucht und die erhaltenen Positionen anschließend gemittelt. Dabei wird darauf geachtet, dass weiße Regionen (*Rot = Grün = Blau*) nicht bei der Berechnung berücksichtigt werden. Da der Tische eine weiße Oberfläche hat, würde dies zu Fehlern führen. Abbildung 16(a) zeigt, welche Aufnahmen die Kamera dabei nutzt, um blaue Regionen im Bild zu finden. Zu Beginn unseres Programmes wird die Position des Roboters normiert, das heißt von allen Positionen, die der reale Crawler von der Kamera erhält, wird die Position abgezogen, zu der sich der Roboter zu Beginn des Programms befand. Dies ist notwendig, da alle Partikel als initiale Position $(0, 0, 0)$ annehmen. Leider ist aber auch die berechnete Position der Kamera fehlerbehaftet, da die Kamera manchmal bei gleichen Eingabebildern verschiedene Weltkoordinaten zurückgibt, was bei der Umrechnung dann natürlich zu falschen Roboterkoordinaten führt. Bei der Umrechnung von Weltkoordinaten in Roboterkoordinaten muss außerdem darauf geachtet werden, dass es durch die Anordnung der Kamera zu falschen Achsenbezeichnungen kommt. Abbildung 16(b) zeigt, wie sich die Weltkoordinaten (rechts) zu den Roboterkoordinaten (links) verhalten. Aus diesem Grund muss die erhaltene Position der Kamera noch einmal auf den Roboter angepasst werden.

7.2 Evaluierung

7.2.1 Vorgehen

Um die Ergebnisse des implementierten SLAM Verfahrens evaluieren zu können, wurden sowohl im realen Anwendungsfall, als auch im Simulierten für jeden Schritt die Koordinaten der Roboterposition festgehalten. Im Falle des simulierten Crawlers wurden hierzu einfach die tatsächlichen Koordinaten der Crawler Route und zusätzlich die geschätzten Positionen des zu jedem Zeitschritt als am Besten bewerteten Partikels aufgezeichnet. Bei der Durchführung mit dem realen Crawler wurde für die vom Crawler gelaufene Route die jeweilige Position mit Hilfe der Kamera gemessen (cf. Sec. 7.1).

7.2.2 Ergebnis simulierter Crawler

Um die grundsätzliche Funktionsweise des SLAM Verfahrens zu verifizieren, wurden die Ergebnisse zunächst mit Hilfe der hierfür entwickelten Simulation evaluiert. Bei dieser Evaluierung gilt es zu beachten, dass die Modellierung des Systems, in diesem Fall des Crawlers, optimal ist. Dies liegt daran, dass für die Prädiktion der Bewegung des simulierten Crawlers und zur Prädiktion der Partikel jeweils auf das gleiche Modell zurückgegriffen wurde. Somit lässt sich davon ausgehen, dass bei ausreichend hoher Partikelanzahl, eines der Partikel mit großer Wahrscheinlichkeit die Route des simulierten Crawlers ablaufen wird.

Abbildung 17 visualisiert die Ergebnisse eines simulierten Testdurchlaufs mit 1000 Partikeln.

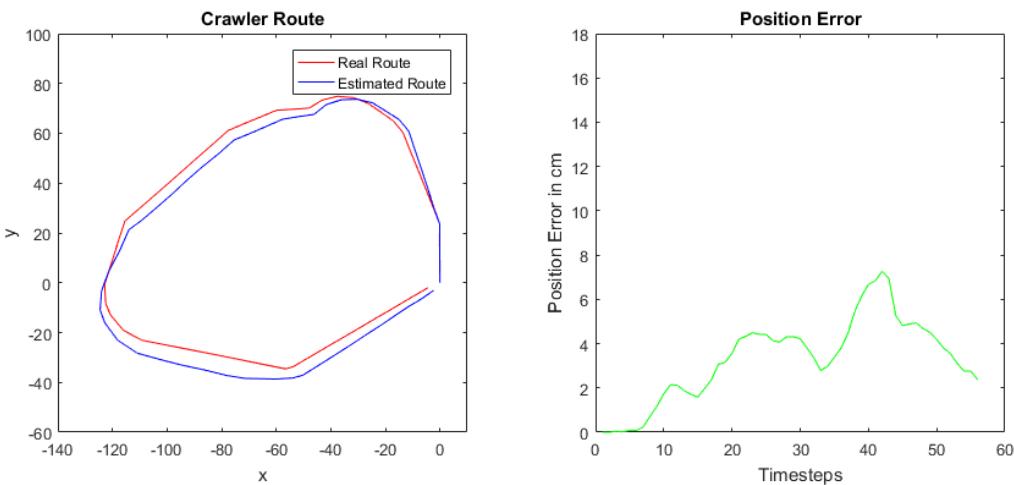


Abbildung 17: Ergebnisse der Evaluation der Simulation, durchgeführt mit 1000 Partikeln. Reale Route (rot) und geschätzte Route (blau) des Crawlers, sowie die Distanz der Positionen (grün)

Anhand des dargestellten Fehlers, der euklidischen Distanz zwischen geschätzter und tatsächlicher Position des Crawlers zu jedem Zeitschritt, lässt sich erkennen, dass sich im Verlaufe des

Testdurchlaufs, bis etwa zu Zeitschritt 50, die Schätzung der Position kontinuierlich verschlechtert. Der maximal gemessene Fehler beträgt in etwa 7cm, steigt aber im Verlaufe der Messung nicht ausschließlich an, sondern wird ab und an wieder geringer.

Zum Abschluss der Messung verbessert sich die gemessene Position von etwa 7cm Abweichung, auf lediglich 2cm. In Anbetracht der Tatsache, dass die Gridmap mit einer Zellengröße von 2cm x 2cm initialisiert wurde, ist dies als sehr gutes Ergebnis zu interpretieren.

Die geschätzte Position verbesserte sich eklatant, als der Crawler Messungen im Bereich der Karte durchführte, der bereits zuvor abgetastet wurden. Dies ist eventuell als eine Art "Loop Closing" Effekt zu interpretieren.

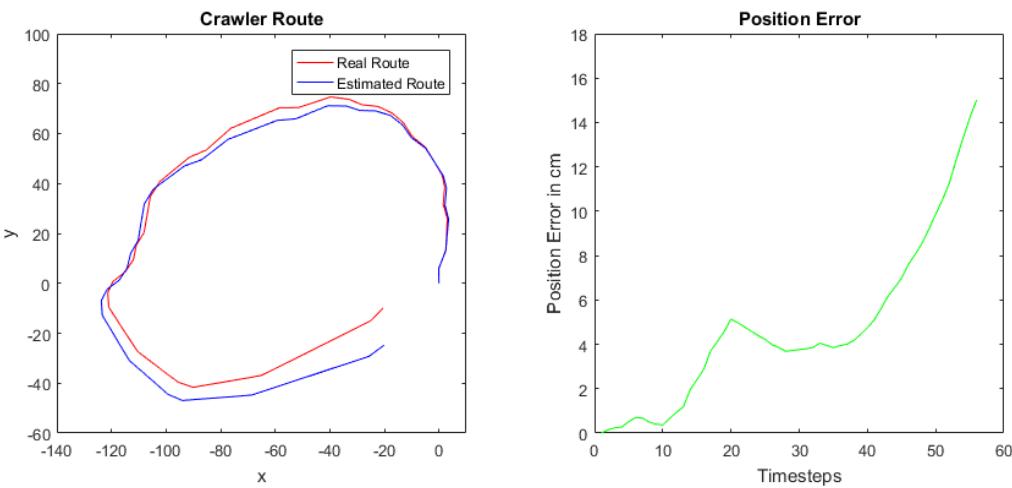


Abbildung 18: Ergebnisse der Evaluation der Simulation, durchgeführt mit einem Partikel. Reale Route (rot) und geschätzte Route (blau) des Crawlers, sowie die Distanz der Positionen (grün)

Zur Verifikation der Ergebnisse wurde im Anschluss an die Messung mit 1000 Partikeln, eine Messung mit nur einem Partikel durchgeführt. Da nur ein Partikel verwendet wird, wird das Partikel ausschließlich auf Basis des Systemmodells prädiziert, es findet kein Resampling statt.

Betrachtet man die Fehlerkurve aus Abbildung 18, so erkennt man das erwartungsgemäße Fehlerverhalten eines Lokalisierungsverfahrens, welches ausschließlich auf Basis der Odometriiedaten die Position des Roboters schätzt. Der Fehler akkumuliert sich über den Verlauf der Messung und beträgt zum Ende der Messung in etwa 15cm.

Die Ergebnisse lassen sich ebenfalls anhand der erstellten Gridmaps veranschaulichen. Abbildung 19(a) zeigt die resultierende Karte aus der Messung mit nur einem Partikel, Abbildung 19(c) die Karte der durchgeführten Messung mit 1000 Partikeln. Vergleicht man die beiden erstellten Karten mit der in der Simulation erstellten, "realen" Karte, so erkennt man deutlich, dass die aus der mit 10000 Partikeln durchgeführte Messung resultierende Karte mehr Ähnlichkeit mit der realen Karte aufweist, als die Karte aus der zweiten Messung mit nur einem Partikel.

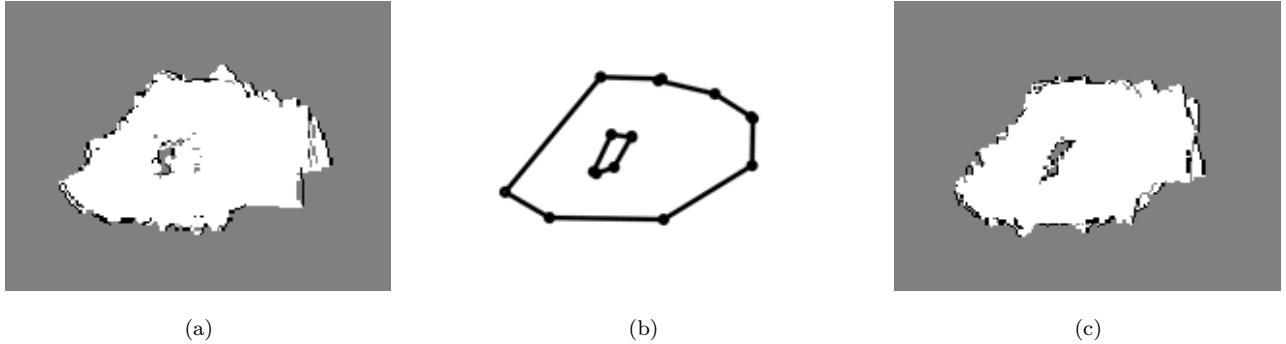


Abbildung 19: (a) Gridmap aus Simulation mit einem Partikel, (b) reale Karte, (c) Gridmap aus Simulation mit 1000 Partikeln

7.2.3 Ergebnis realer Crawler

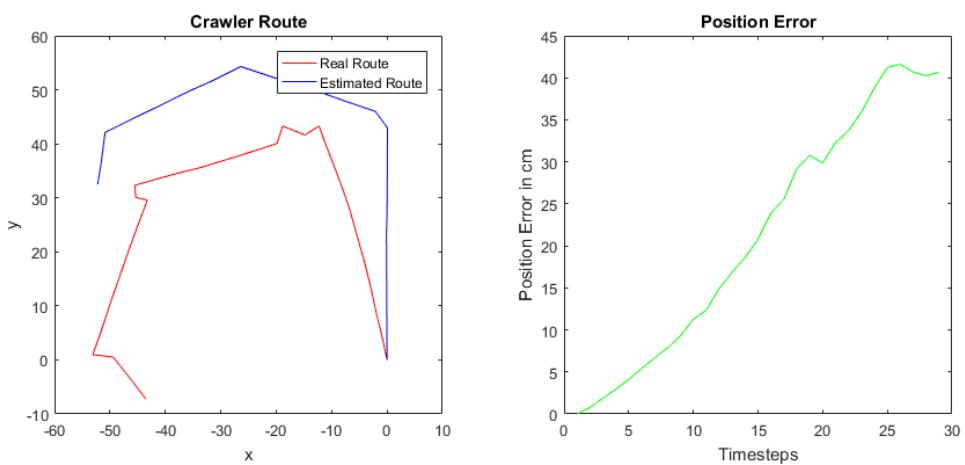
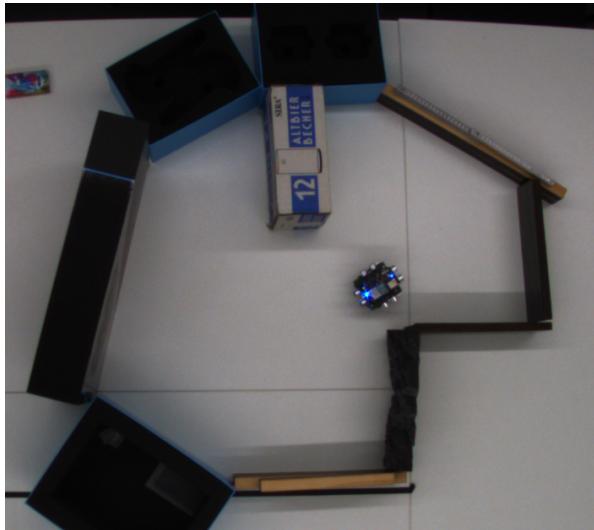


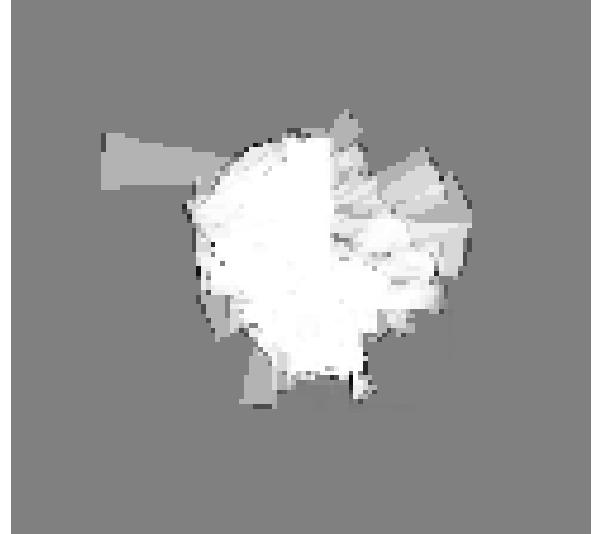
Abbildung 20: Reale Route (rot) und geschätzte Route (blau) des Crawlers sowie die Distanz der Positionen (grün)

Um den echten Crawler zu evaluieren wird das in Kapitel 7.1 beschriebene Aufnahmesystem verwendet. Partikel und Filter arbeiten hier genauso wie im simulierten Modell, lediglich die Bewegung des Crawlers, die Abstandsmessung und die Berechnung der echten Position des Crawlers werden hier mit realen Komponenten durchgeführt. Obwohl in Kapitel 7.2.2 gezeigt wurde, dass unser Programm gute Ergebnisse mit simulierten Daten erzielt, lässt sich dies im realen Fall nicht gänzlich nachvollziehen. Abbildung 20 zeigt, dass auch unter Verwendung mehrerer Partikel (hier 500), die Position irgendwann verloren wird. Dies kann mehrere Gründe haben. Eine mögliche Ursache könnte eine schlechten Approximation der Kovarianzmatrizen sein. Dies würde auch erklären, weshalb der simulierte Crawler bessere Ergebnisse erzielt, als der reale, da die reale Position und die berechneten Messungen des simulierten Crawler ebenfalls aus diesen Verteilungen gezogen werden. Eine weitere mögliche Ursache können nicht modellierte Unsicherheiten durch den Benutzer darstellen. Da in der realen Anwendung der Crawler annimmt, mit Orientierung 0° zu starten, könnten kleine Fehler bei der initialen Aufstellung des Crawler bereits zu großen Fehlern bei der Lokalisierung führen. Dies sieht man deutlich in

Abbildung 20, da der reale Crawler schon zu Beginn des Programms eine andere Orientierung hat, als die Partikel. Dieses Phänomen wurde bei fast allen unserer Aufnahmen festgestellt. Außerdem kann auch die in Kapitel 7.1 angesprochene Unsicherheit der Kamera zu fehlerhaften Roboterpositionen führen. Bei Tests haben wir Fehlmessungen von bis zu $29mm$ festgestellt. Eine solche Fehlschätzung ist auch für den Ausschlag an Position $(-18, 42)$ verantwortlich. Schließlich kann, da es sich bei SLAM um ein probabilistisches Problem handelt, auch einfach der Zufall eine Rolle spielen. Trotzdem sieht man an Abbildung 20, dass die grobe Richtung des



(a)



(b)

Abbildung 21: Reale Karte und erstelle Karte des Roboters

Roboters durch die Partikel approximiert wird. Auch die gefundene Karte entspricht zumindest annähernd der Realen. Abbildung 21(a) zeigt dies. Markante Regionen wie Kanten werden trotz falscher Positionierung der Partikel gefunden, und in den Raum ragende Hindernisse sind trotz vieler Fehlmessungen erkennbar. Somit ist aufgrund mehrerer Möglichkeiten die reale Positionierung und Kartographierung des Crawlers nicht besonders gut, aber trotzdem wird eine ähnliche Route des Roboters und eine ungefähre Karte der Umgebung gefunden.

8 Zusammenfassung & Ausblick

Abschließend folgt nun eine Zusammenfassung sowie ein Ausblick über mögliche Verbesserungen bzw. Erweiterungen.

Zu Begin wurde die Crawler-Software überarbeitet. Hierdurch wurden die vorher benötigten Pythonskripte überflüssig, was zur Folge hatte, dass keine SD-Karte mehr benötigt wird. Zusätzlich wurde die Anzahl an Servomotoren von fünf auf vier verringert und der Low-Level Code optimiert. Abschließend wurde eine Matlab-Bibliothek erstellt, die für die Kommunikation mit dem Crawler via REST-Schnittstelle verwendet werden kann. Während der anschließenden Inbetriebnahme des Crawlers, wurden zwei Hardwarefehler entdeckt und behoben.

Parallel dazu wurde mit der Erstellung einer Simulationsumgebung und Benutzeroberfläche begonnen. Hierbei wurde auf eine intuitive Bedienung geachtet sowie die Zurverfügungstellung einer grafischen Oberfläche, mit der die reale sowie simulierte Umgebung angesteuert werden kann. Vorteil dieser engen Verzahnung ist, dass der Crawler sowie die verwendeten Sensoren austausch- bzw. erweiterbar sind. Zusätzlich können mithilfe von „Obstacles“ ganze Szenen konstruiert werden, die durch eine integrierte Speicher- sowie Ladefunktion beliebig oft wiederverwendet und bspw. zum Vergleich verschiedener Systeme verwendet werden können.

Anschließend wurde mit der Implementierung des SLAM Verfahrens (grid-based FastSLAM) begonnen. Hierzu wurde die NLE-Toolbox um die Verwendung von Occupancy GridMaps erweitert und ein System- sowie Messmodell definiert. Hierbei wurde darauf geachtet, die Struktur der Toolbox nicht grundlegend zu ändern, wodurch ein späteres hinzufügen von weiteren Modellen vereinfacht wird. Zusätzlich wird die Möglichkeit des hinzufügens bzw. verwendens anderer Fiterverfahren hierdurch nicht beeinträchtigt.

Abschließend wurde eine Evaluierung durchgeführt, wobei eine Trennung des realen sowie simulierten Crawlers erfolgte. Bei Verwendung des zweiten kam es zu guten Ergebnissen. Während der Verwendung des realen Crawlers kam es jedoch wahrscheinlich zu einer Summation des gemachten Fehlers über die Laufzeit, wodurch sich die geschätzte Pose immer weiter von der realen entfernte. Mögliche Gründe hierfür könnten sein: eine schlechte Approximation der Kovarianzmatrix, nicht modellierte Unsicherheiten sowie Fehler im Messsystem.

Mögliche zukünftige Arbeiten an dem System lassen sich somit in zwei Kategorien einteilen:

1. Verbesserungen bzw. Optimierungen
2. Erweiterungen

Unter (1) fallen Performanceoptimierungen, wie bspw. eine Parallelisierung. Je performanter, desto mehr Partikel können verwendet werden, wodurch sich die Approximation verbessern kann. Aber auch Verbesserungen bez. der verwendeten Modelle sollte zu einer besseren Approximation führen. Hierbei sollte vor allem das Messmodell überarbeitet werden, welches aktuell eine eher rudimentäre Beschreibung vornimmt.

Alternativ sind auch Erweiterungen (2) sowohl seitens Hardware als auch Software möglich. Bez. ersterem könnten verschiedene Sensoren zu Erkundung der Umwelt evaluiert oder die in einer vorherigen Crawlerversion (s. [1]) entwickelten Abgrundsensoren verwendet werden. Softwareseitig wäre eine Erweiterung auf dynamische Szene denkbar, wodurch eine Verwendung in nicht-gestellten Szenarien oder gar eine Interaktion mit der Umwelt möglich wäre. Eine weitere, mögliche Erweiterung wäre die Verwendung eines Pathfinder-Algorithmus zur Steuerung des Crawlers, sodass dieser seine Umwelt autonom erkundet.

Literatur

- [1] Nils Berg, Matthias Holoch, and Michael Vollmer. Aufbau und Regelung von Miniatur-Laufrobotern. Technical report, Lehrstuhl für Intelligente Sensor-Aktor-Systeme (ISAS), Institut für Anthropomatik, Fakultät für Informatik, Karlsruher Institut für Technologie (KIT), 2016. Praktikum Forschungsprojekt: Anthropomatik praktisch erfahren (SS2015).
- [2] Jean-Yves Bouguet. Camera calibration toolbox for matlab.
https://www.vision.caltech.edu/bouguetj/calib_doc/, 2015. Accessed: 2017-02-09.
- [3] Dietrich Brunn, Felix Sawo, and Uwe D. Hanebeck. Modellbasierte Vermessung verteilter Phänomene und Generierung optimaler Messsequenzen. *tm - Technisches Messen, Oldenbourg Verlag*, 3:75–90, March 2007.
- [4] Adam Milstein. *Occupancy Grid Maps for Localization and Mapping*. INTECH Open Access Publisher, 2008.
- [5] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
- [6] Cyrill Stachniss. Robot mapping - grid based fast slam. <http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam13-gridfastslam.pdf>, WS13/14. Accessed: 2016-12-02.
- [7] Cyrill Stachniss. Robot mapping - grid maps. <http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam10-gridmaps.pdf>, WS13/14. Accessed: 2016-11-15.
- [8] Jannik Steinbring. Nonlinear estimation toolbox.
<https://bitbucket.org/nonlinearestimation/toolbox>. Accessed: 2017-02-05.
- [9] Florian Weissel, Marco F. Huber, and Uwe D. Hanebeck. Efficient Control of Nonlinear Noise-Corrupted Systems Using a Novel Model Predictive Control Framework. In *Proceedings of the 2007 American Control Conference (ACC 2007)*, pages 3751–3756, New York, New York, July 2007.