

Praktikum

Forschungsprojekt

*Anthropomatik praktisch
erfahren*

3. August 2017



Michael Haas, Christoph Jost, Joanna Müller,
Kaspar Wulf
Infrarot-basierte Simultane Lokalisierung und
Kartografierung

Infrarot-basierte Simultane Lokalisierung und Kartografierung

**– Praktikum: Forschungsprojekt Anthropomatik
praktisch erfahren –**

Michael Haas, Christoph Jost, Joanna Müller, Kaspar Wulf

3. August 2017

Zusammenfassung

In dieser Arbeit wird ein Algorithmus zur Simultanen Lokalisierung und Kartographierung in unbekannter Umgebung durch einen Laufroboter untersucht und implementiert. Ebenfalls Teil der Aufgabe ist es, einen bestehenden Roboter mit der dafür benötigten Sensorik auszustatten und zugehörige Landmarken zu entwerfen und fertigen. Ein solches Kartographierungsverfahren ist besonders geeignet, um die freie Bewegung eines Roboters in neuen Orten zu ermöglichen, ohne auf genaues, aufwendig generiertes Vorwissen zurückgreifen zu müssen. Da das Testen des Verfahrens auf realer Hardware aufwendig und langwierig ist, haben wir zusätzlich eine Simulationsumgebung entwickelt, anhand derer wir den Kartographierungsalgorithmus evaluieren können. Zur Bewertung haben wir die durchschnittliche Abweichung ebenso wie deren Varianz von Landmarkenpositionen im Vergleich zu den Ground Truth-Daten gemessen, die sich am Ende eines kompletten Durchlaufs ergeben haben. Wir haben ebenfalls am Ende jedes Durchlaufs die Anzahl der identifizierten Landmarken mit der tatsächlichen Anzahl verglichen. Als Entwicklungsumgebung für den Algorithmus und die Simulationsumgebung haben wir Matlab ausgewählt, da es für Matrixberechnungen, wie die im implementierten Algorithmus, optimiert ist und eine graphische Evaluation erleichtert. Für die Robotersensorik haben wir einen preiswerter Spielcontroller demontiert und dessen Infrarotkamera entnommen. Als Resultat konnten wir bei unseren Test in der Simulationsumgebung feststellen, dass der Algorithmus in der Mehrzahl der Fälle (ca. 87%), die richtige Anzahl an Landmarken erkannt wird und die Positionen auf ungefähr 2.5 mm genau geschätzt werden können. Die Roboterhardware haben wir erfolgreich um Infrarotsensorik erweitert und haben sechs Infrarot Landmarken hergestellt. Diese Ergebnisse bilden eine gute Ausgangssituation für weitere, aufbauende Arbeiten, haben jedoch ebenso Potential für Verbesserungen.

Inhaltsverzeichnis

1 Einleitung	5
2 Der Laufroboter	6
3 Sensor	7
3.1 Wii-Fernbedienung	7
3.2 Fertigung und Anbringung	7
3.3 Kommunikations-Software	8
3.4 Testen	9
4 Landmarken	12
4.0.1 Infrarot-LEDs	12
4.0.2 Stromversorgung	12
4.0.3 Höhe der Landmarken	12
4.0.4 Gehäuse	13
5 Robotersteuerung	15
6 Software	17
6.1 Modul: SLAM	18
6.2 Modul: Crawler	18
6.3 Modul: Pathplanner	19
6.4 Modul: GUI	20
6.5 Run-Dateien	21
7 Simultaneous Localization and Mapping	22
7.1 Extended Kalman Filter	22
7.1.1 Interface	23
7.1.2 Modelle	24
7.1.3 Landmarken Zuordnung	24

8 Evaluation	26
8.1 Simulations Batch-Ausführung	26
8.1.1 Known Issues	26
8.1.2 Szenario	26
8.1.3 Ergebnisse	26
8.2 Laufzeittests zur Landmarkenzuordnung	27
9 Fazit und zukünftige Arbeiten	28
9.1 Fazit	28
9.2 Zukünftige Arbeiten	28
9.2.1 Kamerakalibrierung	28
9.2.2 Kamerakonfiguration	28
9.2.3 Kombiniertes Infrarot- und Ultraschall-SLAM-Verfahren	29
9.2.4 Kommunikation zwischen mehreren Crawlern	29

1 Einleitung

Damit Roboter als Helfer oder Assistent von Menschen beispielsweise im Haushalt eingesetzt werden können, ist es erforderlich, dass diese sich in von Menschen geschaffener, veränderlicher Umgebung zurecht finden. Hierfür ist eine genaue Lokalisierung des Agenten sowie der Objekte und Einrichtung, mit der interagiert werden soll, in einer präzisen Karte erforderlich. Allerdings ist insbesondere eine genau Karte der Umgebung, in der sich der Roboter bewegt, oft nicht vorhanden, sondern im besten Fall nur eine A-priori-Schätzung davon [3]. Dementsprechend kann man die Position des Agenten in dieser ebenfalls nur näherungsweise bestimmen. Um ein solches Szenario trotzdem umzusetzen, ist also eine Verfeinerung der Karte oder eine vollständige neue Erzeugung dieser erforderlich. Dies kann der Roboter allerdings immer nur relativ zur eigenen Position bzw. der angebrachten Sensorik (in der Karte), was zur einer wechselseitigen Abhängigkeit zwischen der Karte und der Roboterposition führt.

Verfahren, die sich mit dieser Art von Problem beschäftigt, fallen unter den Begriff der Simultanen Lokalisierung und Kartographierung (SLAM, engl. Simultaneous Localization and Mapping). Dabei handelt es sich um iterative Verfahren, für die der Roboter sich im Raum bewegt und dabei Stück für Stück eine Karte seiner Umgebung aufbaut. Da es aufgrund von Messfehlern der Sensoren zu Ungenauigkeiten kommen kann und ein perfektes Modell der Bewegung nur schwer erreichbar ist, müssen diese Unsicherheiten ebenfalls mithilfe von stochastischen Maßen wie Mittelwert und Varianz berücksichtigt werden [11]. Um solche Maßzahlen zu berechnen und um damit die Unsicherheiten darzustellen, wird bei SLAM in jedem Schritt eine Prognose der Bewegung vom Roboter und der daraus resultierenden neuen Position relativ zu seiner Umgebung mit den durch seine Sensoren aufgenommenen Daten nach dem ausgeführten Schritt verglichen [4].

Zur Durchführung haben wir im Rahmen des Praktikum Forschungsprojekts „Anthropomatik praktisch erfahren“ einen etwa zehn Zentimeter hoher Laufroboter zur Verfügung gestellt bekommen, der anhand von Infrarot (IR)-Licht ausstrahlenden Landmarken eine Karte erstellen können wird. Dafür wird der Roboter um einen IR-Sensor erweitert.

Um einen strukturierten Einblick in unser Projekt zu geben, ist die Ausarbeitung wie folgt aufgebaut: Zu Beginn wird der verwendete Laufroboter in Kapitel 2 beschrieben und abgebildet. Das Kapitel 3 beschäftigt sich mit dem neu integrierten Sensor und seinem Entwurf und wird ergänzt durch das Kapitel 4, welches Details über die verwendeten Landmarken enthält. Informationen über die Robotersteuerung und weitere Software werden in den Kapitel 5 bis 7 beschrieben. Dabei befindet sich eine Beschreibung der Bewegungssteuerung und der daran vorgenommen Optimierungen in Kapitel 5, Details über die Simulationsofware und ihren Aufbau inklusive Quelltextausschnitte in Kapitel 6 und eine ausführliche Beschreibung von SLAM zusammen mit den verwendeten Modellen in Kapitel 7. Abgerundet wird die Arbeit durch eine Übersicht der Ergebnisse und Informationen zum Versuchsaufbau in Kapitel 8 sowie abschließende Kommentare zum Projekt und darauf aufbauende Ideen für zukünftige Arbeiten in Kapitel 9.

2 Der Laufroboter

Bei dem im Praktikum verwendeten Laufroboter handelt es sich um den am Lehrstuhl entwickelten Crawler [5], der auch in Abbildung 1 zu sehen ist. Dieser kann sich durch seine sechs unabhängigen aktorischen Freiheitsgraden omnidirektional in der Ebene bewegen. Konkret bedeutet das, dass er vorwärts, rückwärts und seitwärts gehen sowie sich nach links und rechts drehen kann, wobei diese Bewegungen beliebig kombiniert werden können. Daten und Befehle lassen sich über WLAN mithilfe von HTTP-Anfragen und kabelgebunden via USB oder Ethernet übertragen [2]. Mit Spannung versorgt werden die Crawler durch Lithium-Polymer-Akkumulatoren. Den logischen Kern bildet ein Arduino Yun Board [1], auf dem sich ein ATmega32U4-Mikrocontroller befindet, der die Servomotoren steuert und die Messungen der verschiedenartigen Sensoren liest.

In seiner ursprünglichen Form sieht der Crawler wie in Abbildung 1 aus. Dieser ist in neueren Versionen mit Ultraschallsensoren vorne, hinten und seitlich ausgestattet und als Teil dieses Projekts um eine IR-Kamera erweitert.



Abbildung 1: Der ISAS-Crawler

3 Sensor

Um IR-Landmarken zu erkennen, ist die Verwendung eines passenden Sensors notwendig. Die Crawler sind jedoch bisher nicht mit solcher Sensorik ausgestattet worden, daher ist die Integration inklusive eines mit den Technikern des Lehrstuhl erarbeiteter Entwurf sowie Testen der gefertigten Sensorplatine Teil des Projekts. Grundlage bildet der IR-Sensor aus der Wii-Fernbedienung des Herstellers Nintendo [9] aufgrund seiner schnellen Verfügbarkeit und der vermuteten Zuverlässigkeit, die man von einem Teil einer Spielkonsole erwarten kann.

3.1 Wii-Fernbedienung

Die Wii-Fernbedienung [10] (Wiimote, engl. Wii Remote Control), die in Abbildung 2 zu sehen ist und für die 2006 gleichnamige Konsole hergestellt worden ist, besteht aus verschiedener Sensorik wie einer IR-Kamera, mehreren Schaltern und diversen elektrischen Bausteinen sicher eingepackt in eine Plastikhülle. Über die genauen technischen Daten der Fernbedienung gibt es nur begrenzt öffentlich zugängliche Informationen, die oft von Hobby-Projekten stammen und experimentell rekonstruiert sind. Da einige Autoren solcher Projekte ihre Arbeit sorgfältig dokumentiert haben, haben sie die Arbeit mit der Wiimote-Kamera deutlich erleichtert.



Abbildung 2: Die originale Wii-Fernbedienung, die als Ausgangspunkt für die Infrarotlichtmessung innerhalb des Praktikums dient.

3.2 Fertigung und Anbringung

Die Überlegung, die gesamte Fernbedienung mitsamt der Plastikverpackung direkt auf den Crawler zu bauen, ist bereits zu Beginn des Praktikums verworfen worden. Dies war vor allem darauf begründet, dass auch hierfür eine Anbringungsmöglichkeit geschaffen hätte werden müssen. Zusätzlich ist die Fernbedienung an sich mit 15 cm etwa doppelt so lang wie der Crawler mit 8.5 cm und stünde daher sichtbar hervor.

Stattdessen sind wir das Design einer neuen Platine frühzeitig angegangen. Für diese haben wir die Wiimote-Platine, zu sehen in Abbildung 3, aus ihrer Hülle entnommen und die Wii-Kamera von der Platine ablöten lassen. Das Design orientiert sich an einem der Projekte von

Eiji Kako [7] und anderen darauf basierten Arbeiten [12] sowie einer Anleitung des Computer Club 2 [6], die dann zu dem Schaltbild geführt haben, welches in Abbildung 4 dargestellt ist. Der rot markierte Bereich zeigt eine Schnittstelle zum Mikrocontroller, der für die Spannungsversorgung der Platine zuständig. Aus diesem Grund sind diese Pins mit dem Spannungsregler im grün markierten Bereich verbunden, der die Arbeitsspannung des ATmega32U4 von 5V auf 3.3V für den Sensor bringt, der mit blau gekennzeichnet ist. Die Daten und der Takt werden bei unserem Prototypen mithilfe des Inter-Integrated Circuit (I2C)-Busprotokolls [14] kommuniziert und, wie man in Abbildung 5 sieht, über Drähte direkt zwischen dem Arduino und dem Sensor übertragen.

Die Platine hat mit $5.3\text{ cm} \times 7.3\text{ cm}$ ungefähr das gleiche Format und die gleiche Größe wie

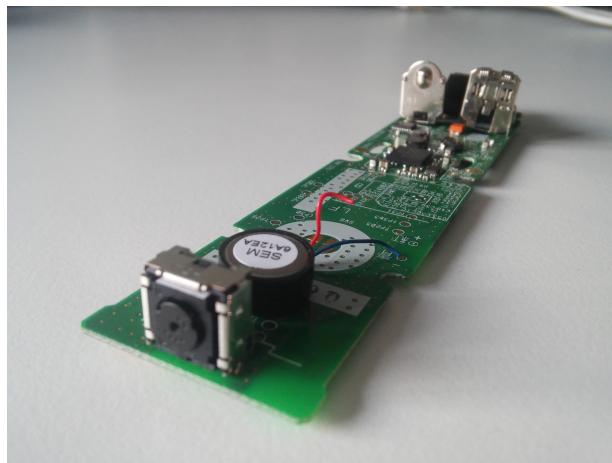


Abbildung 3: Die Wii-Fernbedienung ohne Plastikhülle mit Fokus auf der Infrarotkamera links unten im Bild

die Arduino Yun-Platine und ist auf dieser über die In-System-Programmierung (ICSP, engl. in-circuit serial programming)-Pins und zwei Plastikstützen befestigt. In Abbildung 7 kann man die fertige Konstruktion aus verschiedenen Perspektiven sehen. Dabei kann man besonders auf Abbildung 7a die einzelnen Teile sehen, die bereits im Zusammenhang mit Abbildung 4 beschrieben worden sind.

Im Entwurf der Platine haben wir ein flexibles Auf- und Absetzen der Platine angestrebt, was beim aktuellen Prototyp wegen der Drähte jedoch nicht mehr möglich ist. In zukünftigen Versionen können die Drähte allerdings durch zusätzlich angebrachte Pins auf dem Arduino und kleineren Änderungen an der Platine bezüglich der Positionierung der Bausteine und der Kanäle ersetzt werden.

3.3 Kommunikations-Software

Das Kommunikationsmodul bildet eine Schnittstelle zwischen einer Bibliothek zum Auslesen der Wiimote-Kamera namens *PVision*, die von Eiji Kako [7] bereit gestellt wird und ein bereits im Vorfeld am Lehrstuhl entwickeltes Programm zu Interaktion mit dem Crawler.

Mit diesem Programm werden Befehle, wie der zum Auslesen des IR-Sensors, die vom Arduino

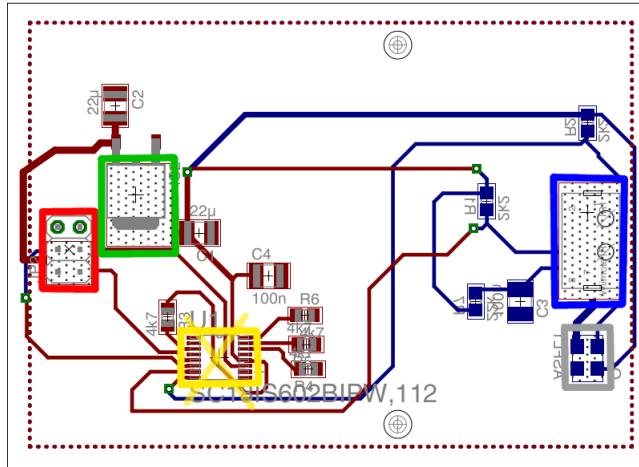


Abbildung 4: Der Schaltentwurf der Platine besteht aus vier Hauptblöcken und einigen Widerständen. Der rote Block kennzeichnet die Schnittstelle zum Mikrocontroller des Arduino, der grüne einen Spannungsregler, um die Arbeitsspannung des Arduino auf die des Sensors zu bringen, der blaue Bereich repräsentiert den Sensor und der gelbe Baustein ist eine Busschnittstelle, die die Daten des Serial Peripheral Interface (SPI) [8]-Bus des Arduino kompatibel macht mit dem I2C-Bus des Sensors. Diesen Baustein haben wir im finalen Entwurf entfernt und sind auf eine Übertragung ausschließlich per I2C umgestiegen. Der graue Block hebt den Quarzoszillator hervor, der die Taktfrequenz bei der Datenübertragung vorgibt. Diese befindet sich auf der Unterseite der Platine.

empfangen werden, an das Kommunikationsmodul zur Verarbeitung weitergereicht. Dieses gibt die gemessenen Punkte dann als Text zurück.

Das Kommunikationsmodul verfügt über eine Initialisierungsroutine für den IR-Sensor bzw. das Aufrufen und Initialisieren des benötigten PVision-Objekts. Diese Routine kann mit verschiedenen Sensibilitätsparametern durchgeführt werden, die aktuell fest eingebaut im Quelltext sind.

3.4 Testen

Die Platine haben wir auf verschiedenen Arten getestet. Vor der Fertigung haben wir zunächst überprüft, ob der Sensor aus der Wiimote funktioniert und für die Distanzen von wenigen Zentimetern bis zu einem Meter für die Erkennung von Landmarken, die selbst im Kapitel 4 näher beschrieben werden, brauchbar ist.

Nach der Fertigung haben wir die Konstruktion des im vorherigen Abschnitt 3.2 beschriebenen Entwurfs sowie den Quarzoszillator (zusammen mit den Technikern des Lehrstuhls) überprüft. Den Datenaustausch zwischen dem Sensor und dem Arduino haben wir separat untersucht, da der Sensor nur auf Anfrage Daten sendet und wir die Kommunikationssoftware erst nach der Fertigung geschrieben haben. Zuerst haben wir kontrolliert, ob überhaupt irgendwelche Daten vom Sensor beim Arduino ankommen, nachdem dieser eine Anfrage gesendet hat. Der nächste Schritt hat darin bestanden, die vom Sensor gesendeten Punkte in einer graphischen Oberfläche (GUI, engl. graphical user interface) anzuzeigen, damit wir den direkten visuellen Vergleich

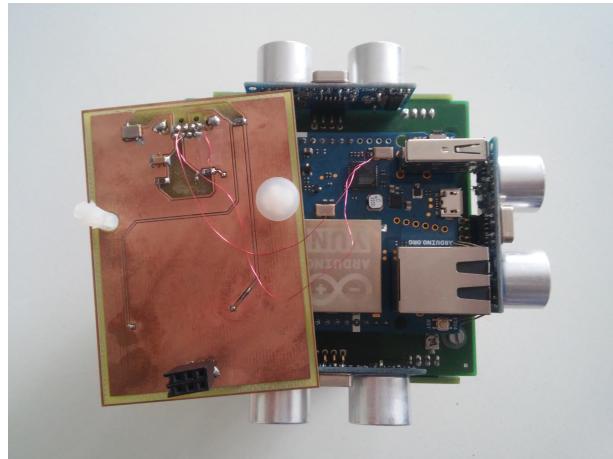


Abbildung 5: Die Unterseite der neuen Platine, die per Draht mit den Daten- und Taktpins des ATmega verbunden ist

haben zwischen dem, was die Kamera sieht und das, was wir sehen und um damit eine Plausibilitätsüberprüfung der Daten zu machen. Bei der GUI handelt es sich um ein schlicht gehaltenes Fenster, dass die Punkte gelb auf einem schwarzen Hintergrund einzeichnet wiedergegeben in Abbildung 6.

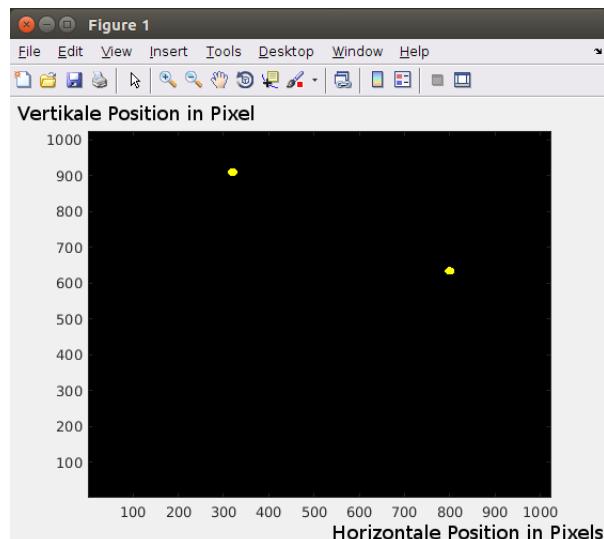
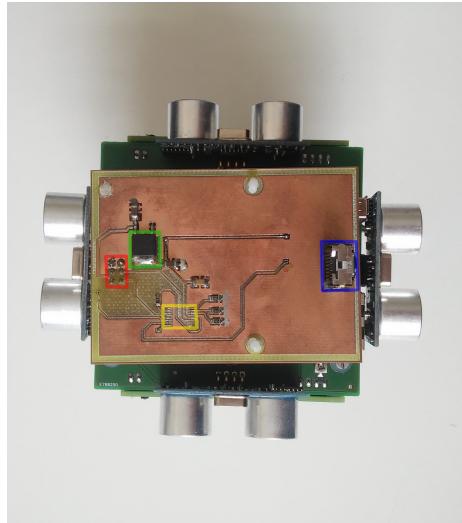
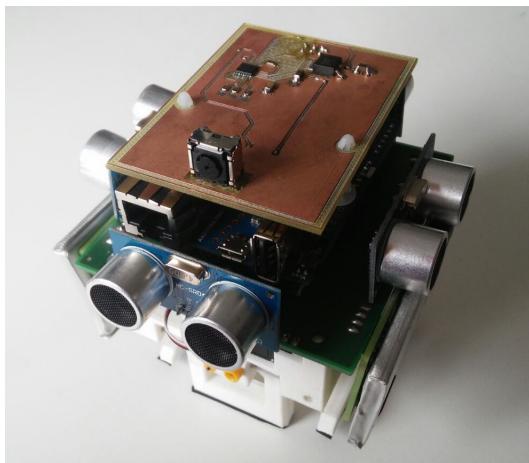


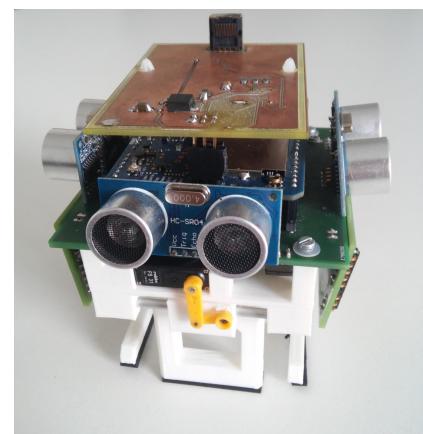
Abbildung 6: Die Graphische Benutzeroberfläche zur Fehlersuche bei der Kommunikation mit dem Sensor



(a) Die Wii-Platine von oben. Die Farben stehen analog zu Abbildung 4 für die Arduino-Schnittstelle zur Spannungsversorgung (rot), für den Spannungsregler (grün), für die (fehlende) Busschnittstelle (gelb) und für den Sensor (blau)



(b) Wii-Platine von vorne. Die Wii-Kamera ist mittig auf dem Rand der Platine oben auf dem Crawler platziert.



(c) Die Wii-Platine von hinten. Die Platine kann einfach auf die ISCP-Pins aufgesteckt werden, die dieser Halt geben.

Abbildung 7: Die Wii-Platine aufgesetzt auf dem Crawler

4 Landmarken

Im Rahmen des Praktikums haben Infrarot-Landmarken entworfen, um die Wii-Kamera und den SLAM-Algorithmus zu testen. Die Anforderungen an diese Landmarken haben sich während des Entwurfs mehrfach durch externe Faktoren geändert bzw. in Zwischenschritten herauskris tallisiert.

4.0.1 Infrarot-LEDs

Unsere Experimente haben gezeigt, das Infrarot-LEDs mit einer Wellenlänge von 940 nm von der verwendeten Wii-Infrarotkamera gut erkannt werden. Der Größte Abstrahlwinkel der verfügbaren Infrarot-LEDs beträgt jedoch nur 150°, von denen nur die zentralen 45° mehr als 50% der vollen Leuchtkraft aufweisen. Um die Landmarken von jeder Richtung aus gut erkennbar zu machen, haben wir vier dieser Infrarot-LEDs kombiniert, was zu der in Abbildung 8b sichtbaren Konstruktion geführt hat. Der Resultierende Infrarot-Lichtkegel deckt 360° ab.

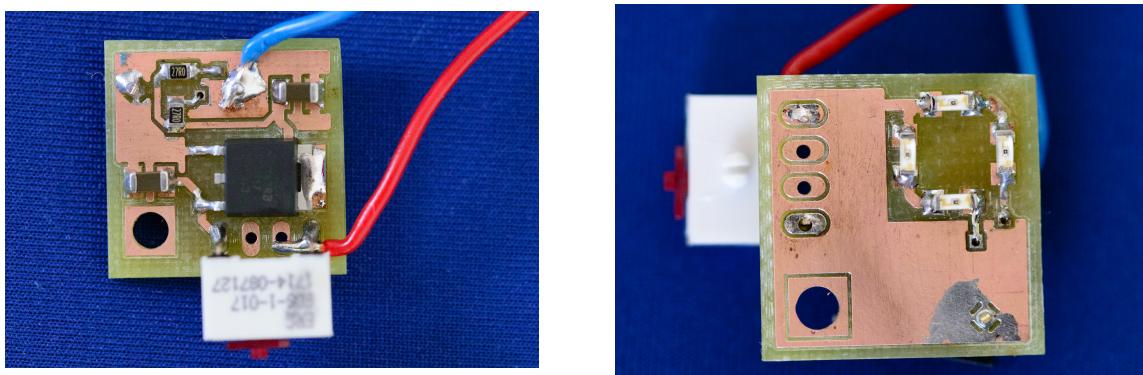


Abbildung 8: Finaler Entwurf

4.0.2 Stromversorgung

Der erste Entwurf sah eine Stromversorgung der Landmarken über Stromkabel vor. Da diese die Roboterbewegung stark eingeschränkt hätten, ist dieser Entwurf zugunsten einer Stromver sorgung mit Akkus verworfen worden. Diese Variante mit wiederaufladbaren Lithium-Polymer- Akkus bietet 20 Stunden Laufzeit und lässt sich sehr schnell auf- und abbauen.

4.0.3 Höhe der Landmarken

Um die Distanz zwischen der Infrarot-Kamera und den Landmarken festzustellen, müssen diese einen Höhenunterschied aufweisen. Zudem sollen die Landmarken zukünftig auch in Szenarien mit mehreren Robotern und anderen Hindernissen eingesetzt werden. Um von diesen nicht

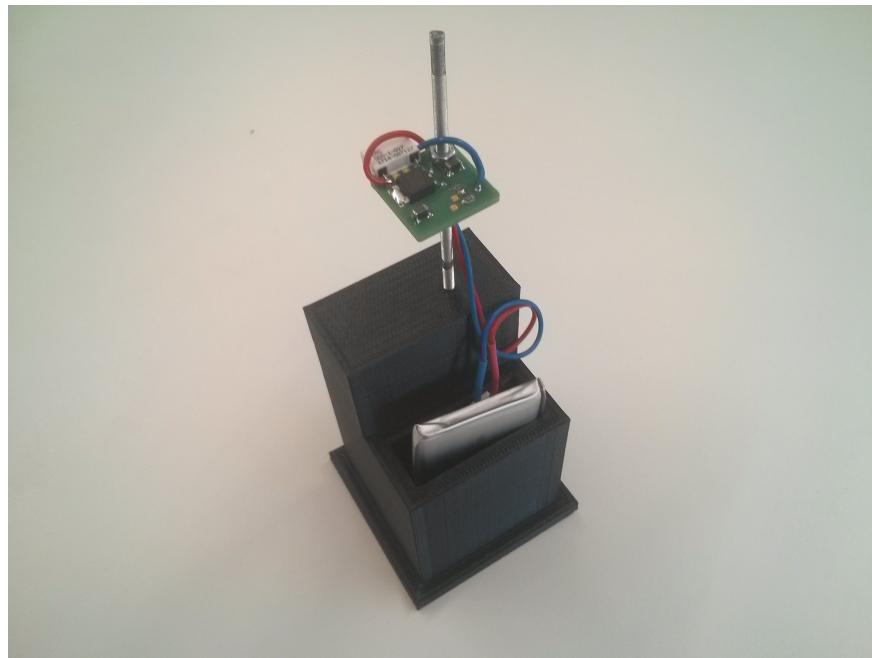


Abbildung 9: Fertige Landmarke

verdeckt zu werden, müssen sie eine bestimmte Mindesthöhe aufweisen. Da einige Faktoren, wie z.B. die Höhe der Kamera auf der von uns entwickelten Platine, zum Konstruktionszeitpunkt noch unbekannt gewesen sind, haben wir uns für eine Höhenverstellbare Lösung entschieden. Die Platine mit den Infrarot-LEDs wird dabei an einer Gewindestange angebracht, durch die die Höhe der LEDs zwischen 9 und 16 cm beliebig eingestellt werden kann. Die Gewindestange, an der die Platine befestigt ist, verursacht einen für uns akzeptablen toten Winkel von ca. 5°. Die umgesetzte Landmarke ist im Vergleich zum Crawler-Roboter in Abbildung 10 und mit stärkerem Augenmerk auf dem Aufbau in Abbildung 11 dargestellt.

4.0.4 Gehäuse

Das Gehäuse der Landmarken besteht aus einem Sockel, einem Fach für den Akku und einem Gewindeloch für die Gewindestange. Es ist mit einem 3D-Drucker hergestellt worden. Das Design ist einfach gehalten und kann mit geringen Aufwand an andere Szenarien, z.B. die gleichzeitige Verwendung als Ultraschall-Hindernis, angepasst werden.

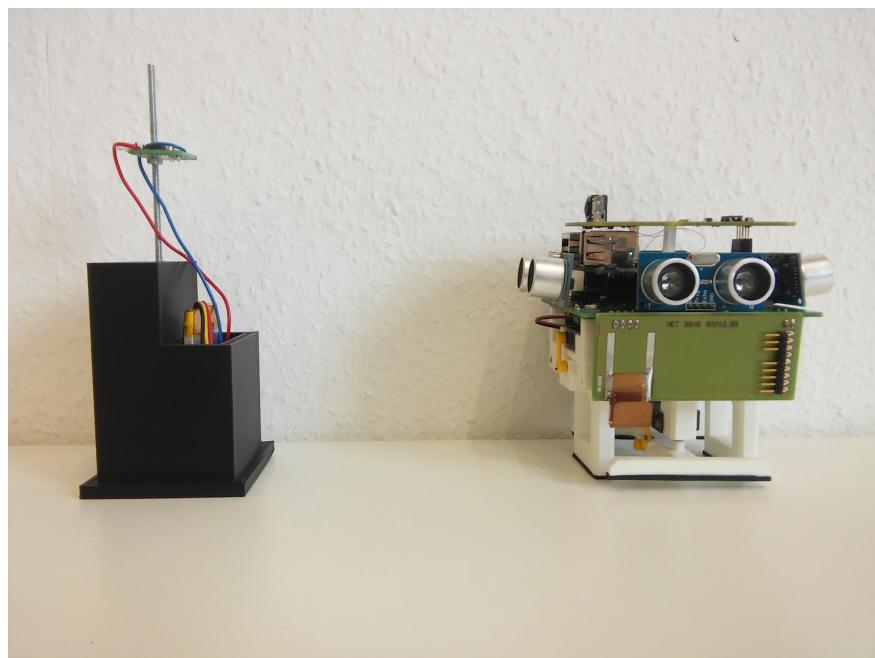


Abbildung 10: Landmarke und Crawler

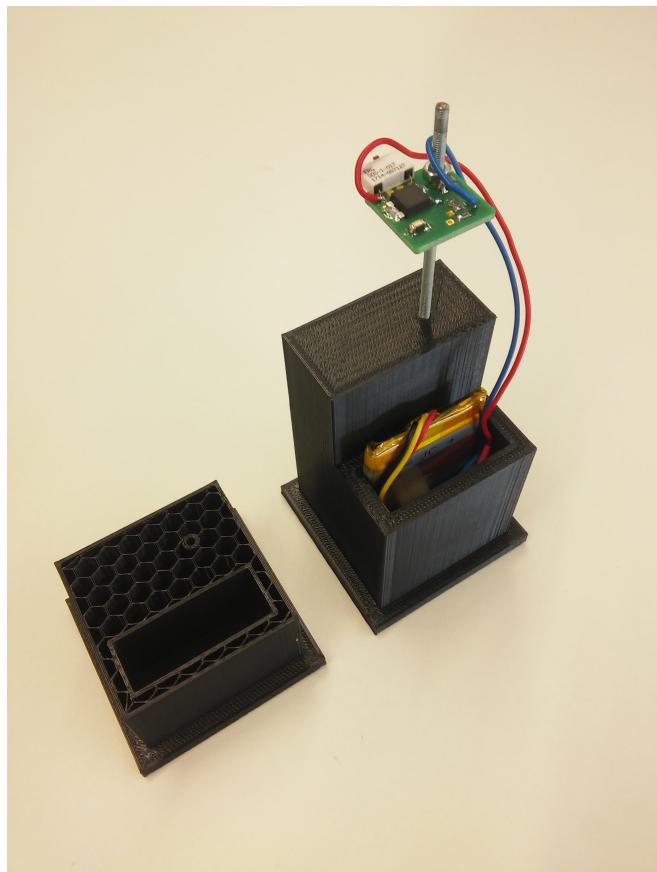


Abbildung 11: Gehäuseteil und fertige Landmarke

5 Robotersteuerung

Die Robotersteuerung umfasst sowohl die Implementierung der Steuerung des Crawlers auf dem Arduino als auch das ansteuern des Crawlers über Matlab. Der Crawler selbst steuert lediglich seine Peripherie an, in dieser Arbeit also die Servos für die Bewegung und die Infrarot-Kamera um die Landmarken zu erkennen. Sämtliche komplexere Berechnungen werden in Matlab auf dem Host-Rechner ausgeführt, der die Bewegungsbefehle an den Crawler senden und die Sensorsdaten empfangen kann. Über das WLAN-Shield des Arduino Yun wird ein Netzwerk aufgebaut, über welches die Kommunikation mittels simplen REST-Calls erfolgt. Dafür läuft auf dem Arduino ein kleiner Server, der in der Lage ist, Anfragen von Klienten, also den Matlab Host-Rechner, zu empfangen und diese abzuarbeiten. Die Arbeitsweise der Kommunikation zwischen dem Crawler und dem Matlab Host-Rechner ist in Abbildung 12 dargestellt. Wie bereits im

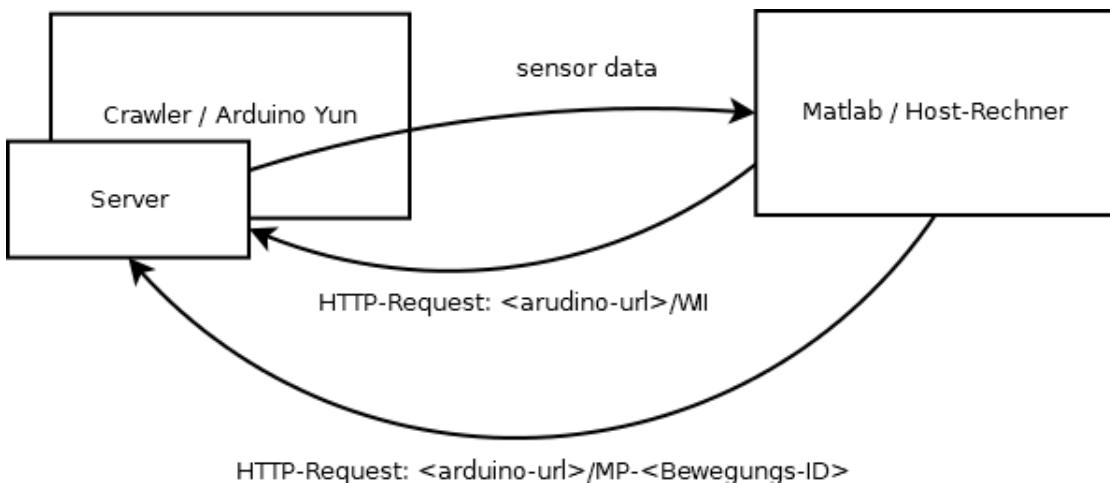


Abbildung 12: Übersicht über die Kommunikation zwischen dem Crawler und dem Matlab Host-Rechner

Kapitel 3 erwähnt, basiert das Programm für die Steuerung des Crawlers auf einem vorherigen Projekt vom Wintersemester 2016/2017. Wir haben uns dafür entschieden, den Quelltext für die Bewegung neu zu implementieren, da der vorherige Bewegungscode nicht die volle Schritt-länge des Crawlers für vorwärts und rückwärts Schritte ausnutzt. Der Grund dafür ist, dass der Crawler nach jeder Bewegung in eine neutrale Ausgangsposition fährt, aus der heraus er jeden weiteren Schritte macht. Dadurch steht der zentrale Fuß für die Bewegung immer mittig. Für die maximale Schrittweise muss dieser Servo erst maximal entlang der Bewegungsachse gestellt werden und dann, nach dem herabsetzen des großen Fußes, auf die maximal entgegengesetzte Richtung gesetzt werden. Das Fahren auf eine neutrale Position hat die Implementierung vereinfacht, da keine verschiedenen Arbeitsschritte, die die Position der Servos abhängig vom vorherigen Schritt anpassen um den folgenden Schritt korrekt auszuführen. Dabei hat man jedoch wie beschrieben vernachlässigt, für die Vorwärts- und Rückwärtsbewegung die komplette Bewegungsreichweite auszunutzen.

In der neuen Implementierung sind dabei 2 Klassen insbesondere relevant: *MotionPrimitive* und *MotionPrimitiveMapping*. Die MotionPrimitiveMapping Klasse speichert dabei die Zuordnung

eines Bewegungsindex zu einer konkreten Bewegungsprimitive. Über die zugehörige Methode kann diese Bewegungsprimitive dann ausgeführt werden. Die Bewegungsprimitive sind als abgeleitete Klassen der abstrakten Klasse *MotionPrimitive* implementiert und steuern die serielle Abfolge der einzelnen Servos.

Der Ablauf bei der Verarbeitung eines Bewegungsbefehls ist hier analog zu dem zum Auslesen der Sensordaten der IR-Kamera. Das Programm auf dem Crawler empfängt HTTP-Requests des Host-Rechners und verarbeitet diese, indem er den im GET-Requests kodierten Befehl an die jeweilige Peripherie weitergibt. Für die Bewegungsteuerung bedeutet das, dass die bereits genannte *MotionPrimitiveMapping* Klasse genutzt wird, um den korrekten Bewegungsbefehl auszuwählen und die Bewegungsprimitive anschließend von den Servomotoren ausgeführt wird. Die Struktur der Crawler-Bewegungssoftware ist zur Verdeutlichung in Listing 1 dargestellt.

```

1 class MotionPrimitive {
2 public:
3     virtual void execute(CrawlerControl* ctrl) = 0;
4 };
5 class MotionPrimitive_WalkForward : public MotionPrimitive {
6 public:
7     void execute(CrawlerControl* ctrl) {
8         // Servos seriell in richtiger Reihenfolge setzen,
9         // + notwendige Ausgangspoistion garantieren!
10    }
11 };
12
13 class MotionPrimitiveMapping {
14     MotionPrimitive* primitives [...] = { ... };
15 public:
16     bool dispatch(int type) {
17         // range check type and execute
18     }
19 };

```

Listing 1: Crawler-Bewegungscode

6 Software

Im Rahmen des Projektes haben wir eine Simulationsumgebung implementiert, die es ermöglicht den entwickelten Algorithmus rein softwareseitig zu testen und auszuwerten. Dies ermöglicht die Arbeit am SLAM-Verfahren, ohne auf die Fertigstellung der Infrarot-Landmarken und der Infrarot-Sensorplatine warten zu müssen. Außerdem sorgt eine Simulationsumgebung für ein einfacheres und schnelleres Testen als bei der Verwendung der echter Hardware.

Des Weiteren können Testläufe mit dem ISAS-Crawler aufgezeichnet und wiederholt abgespielt werden, ohne jedes Mal einen erneuten Testdurchlauf mit dem Crawler durchführen zu müssen. Bei der Umsetzung in Software haben wir besonders auf Modularität und die möglichst einfache Integration verschiedener Crawler Implementierungen Wert gelegt. Der dadurch erzielte Vorteil ist hohe Code-Wiederverwendbarkeit zwischen der Implementierung eines rein simulierten, sowie eines echten Crawlers. In den folgenden Abschnitten sind die Kernbausteine der Simulationsumgebung sowie deren Zusammenspiel dokumentiert. Insbesondere sind dabei genau die Komponenten aufgelistet, die von einem Nutzer der Simulation eventuell angepasst oder erweitert werden müssen. In Abbildung 13 ist eine generelle Übersicht der einzelnen Komponenten dargestellt, die wir während der Software-Entwicklung umgesetzt haben.

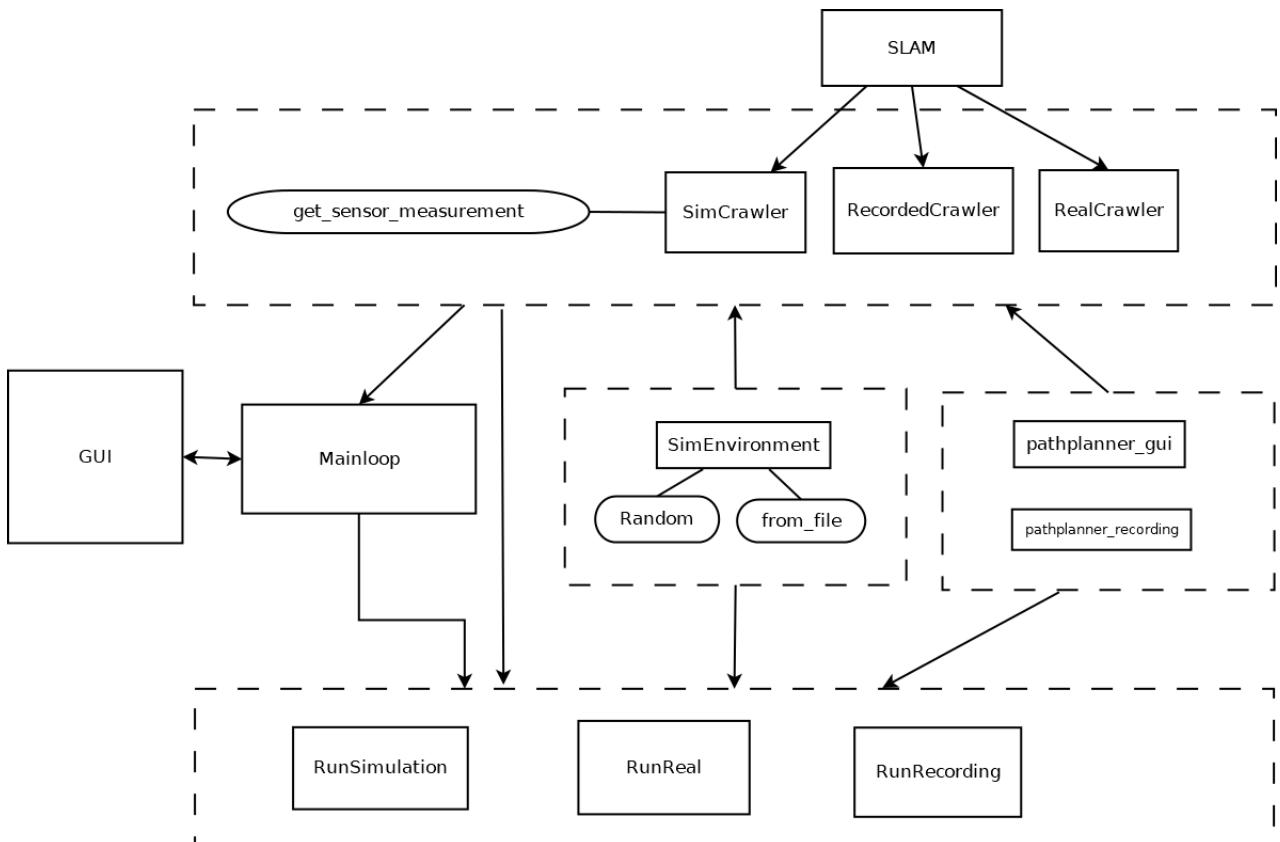


Abbildung 13: Übersicht der Softwareumgebung

6.1 Modul: SLAM

Das *SLAM*-Modul repräsentiert die Kernkomponente des Projekts: die konkrete Implementierung des EKF-SLAM Algorithmus, der dem Crawler die Möglichkeit gibt sich in einer beliebigen, unbekannten Umgebung zurecht zu finden. Lediglich eine einzelne Methode mit dem Namen *slamIt* der SLAM-Implementierung ist für das restliche System zugänglich. Die Eingabeparameter sind der Crawler-Kontrollvektor u inklusive zugehöriger Fehlermatrix N sowie die vom Crawler im aktuellen Zustand empfangenen Sensordaten y_list und der zu y_list zugehöriger Fehlermatrix R . Das notwendige Interface ist in Abbildung 2 schematisch dargestellt. Das SLAM-Objekt selbst verwaltet dabei intern die geschätzten Positionen des Crawlers und der bekannten Landmarken. Die SLAM-Klasse verfügt über weitere Methoden, die den Roboterzustand und die berechnete Kovarianz-Matrix zurück geben. Diese Methoden sind notwendig um beispielsweise die Unsicherheit der geschätzten Positionen der vom Roboter gesehenen Landmarken anzugeben.

```

1 classdef SLAM < handle
2     methods (Access = public)
3         function slamIt(obj, u, N, y_list, R)
4             % Konkrete Implementierung
5         end
6         function out = getX(obj)
7             % Konkrete Implementierung
8         end
9         function out = getP(obj)
10            % Konkrete Implementierung
11        end
12    end
13 end

```

Listing 2: Interface des SLAM Moduls

6.2 Modul: Crawler

Durch *Crawler*-Modul ist es möglich verschiedene Crawler-Implementierungen zu benutzen. Während des Praktikums haben wir einen SimCrawler, der Bewegung und Kameradaten emuliert, sowie ein Replay-Crawler, der Bewegung und Kameradaten eines Testlaufs mit dem ISAS-Crawlers wiedergibt, umgesetzt. Die Crawler-Implementierungen unterscheiden sich nur im Sensoren-Feedback und der Bewegungsansteuerung. Die restlichen Komponenten, wie beispielsweise das Bewegungsmodell und die Berechnung der Odometrie, sind identisch. Damit ist es möglich auf Basis unseres *SimCrawlers* weitere Crawler zu implementieren, die dabei lediglich diese 2 Komponenten neu definieren. Die Bewegungssteuerung ist dabei in ein eigenes Modul (siehe Abschnitt 6.3) ausgelagert, welches im Konstruktor an den *SimCrawler* übergeben werden muss. Eine schematische Darstellung einer Crawler Implementierung befindet sich

in Listing 3. Im Projektordner unter *crawler-control/src/crawler* ist die Implementierung des *ReplayCrawlers* gespeichert, die genau nach diesem Prinzip darauf aufgebaut ist und die sowohl die *readSensor*-Methode als auch den *Pathplanner* überschreibt.

```

1 classdef MyCrawler < SimCrawler
2     methods (Access = public)
3         function obj = MyCrawler ( . . . )
4             % Konstruktoraufruf SimCrawler !
5             % Konkrete Implementierung
6         end
7         function sensor_data = readSensor (obj)
8             % Konkrete Implementierung
9         end
10    end
11 end

```

Listing 3: Interface einer eigenen Crawler Implementierung

6.3 Modul: Pathplanner

Das *Pathplanner*-Modul steuert die Bewegung des Crawlers und gibt einen Kontrollvektor vor. Im Rahmen des Projektes haben wir einen GUI-Pathplanner, über den der Crawler manuell durch einen User gesteuert werden kann, sowie ein *recorded-file-pathplanner* implementiert, der einen in einer Datei festgehaltenen Pfad wie den des *ReplayCrawler* abarbeitet. Eine konkrete Implementierung muss dabei lediglich eine Methode *getNextMoveCommand* unterstützen, die den Befehl, der als nächstes vom Crawler ausgeführt werden soll, zurück gibt. Die schematische Darstellung einer solchen Klasse ist in Abbildung 4 gegeben.

```

1 classdef MyPathplanner < handle
2     methods (Access = public)
3         function obj = MyPathplanner ( . . . )
4             % Konkrete Implementierung
5         end
6         function move = getNextMoveCommand (obj)
7             % Konkrete Implementierung
8         end
9     end
10 end

```

Listing 4: Interface des Pathplanner-Moduls

6.4 Modul: GUI

Hierbei handelt es sich um ein Modul zur Darstellung der aufgezeichneten, realen oder der simulierten Daten und der durch SLAM berechneten Positionen. Das Zeichnen der Ergebnisse aus den einzelnen Schritten ebenso wie das Protokollieren dieser Daten, damit man auch nach Beenden des Programms Einsicht darauf hat, sind die grundlegenden Aufgaben der Visualisierung. Die Visualisierung in erster Linie hinsichtlich der Implementierung verändert. Zunächst haben wir mit einem reinen Matlab-Skript gearbeitet, das im Laufe des Praktikums in einer Klasse gekapselt worden ist. Ebenso haben wir die Ausgabe dieses Moduls überarbeitet. In der ersten Version der graphischen Benutzeroberfläche (GUI, engl. graphical user interface), die in Abbildung 14 zu sehen ist, werden die Positionen des Roboters und der Landmarken zunächst in eine Matrix gezeichnet. Dabei wird für die Schätzungen und die Ground Truth-Daten je eine eigene Matrix angelegt. Diese Bilder werden dann übereinander geschichtet angezeigt. Dieses Vorgehen hat sich jedoch als weniger handlich und vor allem als zu langsam erwiesen, weshalb im Laufe des Praktikums zu einer einfachen Plotdarstellung übergegangen sind. Diese enthält sowohl die Schätzung als auch die korrekten Positionen. Dies ist auch in Abbildung 15 veranschaulicht.

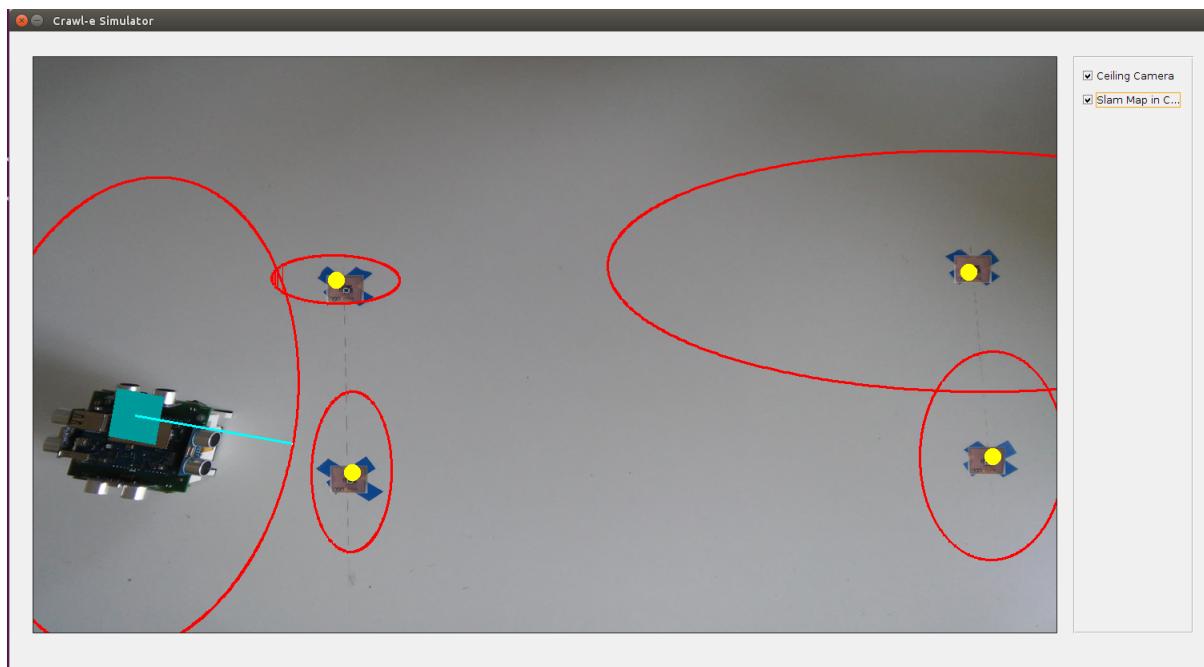


Abbildung 14: Erste Simulations-GUI, die die einzelnen Simulationsschritte anzeigen kann. Hier werden die exakten und geschätzten Positionen auf verschiedene Bilder gemalt und dann übereinander angezeigt.

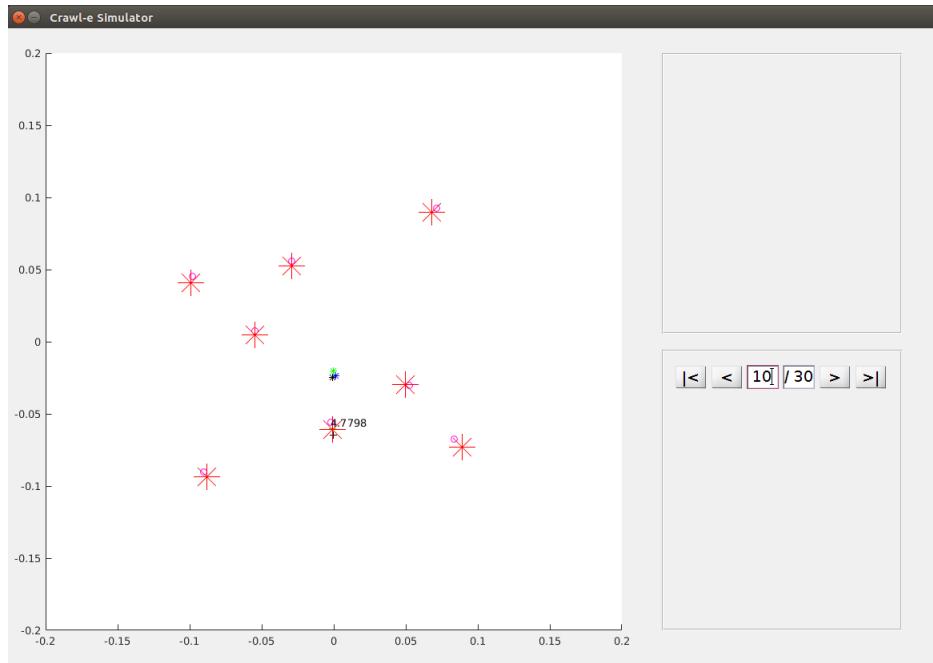


Abbildung 15: Aktuelle Simulations-GUI, mit der man ebenfalls durch die einzelnen Simulationschritte durchgehen kann. Statt in übereinander geschichteten Bildern werden hier jedoch die Positionen in einen handlichen Plot gezeichnet.

6.5 Run-Dateien

Die *Run*-Dateien im Verzeichnis *crawler-control/src/_runs* sind der Eintrittspunkt in die Simulation, in der die Umwelt und der Crawler sowie der zugehörige Pathplanner initialisiert werden. Hier kann ebenfalls die Anzahl der SLAM-Schritte festgelegt werden.

7 Simultaneous Localization and Mapping

Die zentrale Aufgabe des Projekts ist die Implementierung eines SLAM-Algorithmus auf Basis von IR-Landmarken. Wie bereits erwähnt, basiert SLAM auf der Idee, neue Positionen für den Roboter und der bereits aufgefundenen Landmarken zu schätzen und diese nach einem Bewegungsschritt mit den Sensordaten zu vergleichen wie in Abbildung 16 dargestellt. Aus dem Vergleich wird der Fehler rekonstruiert, der bei der Schätzung gemacht worden ist und die Positionen rückwirkend korrigiert.

Da die Sensordaten Rauschen enthalten und die Bewegung des Roboters unpräzise ist, wird ein probabilistisches Filter benötigt. Zu diesem Zweck haben wir uns für die Umsetzung eines Kalman-Filters entschieden. Allerdings verwenden wir eine leicht abgewandelte Version mit dem Namen Extended Kalman-Filter (EKF), da dieses nicht eine Gaußverteilung des Rauschens voraussetzt, sondern in der Lage ist, eine Linearisierung der Sensordaten vorzunehmen. Ein Überblick über SLAM-Verfahren haben wir uns mithilfe eines Einführungsdokuments der Autoren Riisgard und Blas [13] erarbeitet. Die Implementierung richtet sich nach dem theoretischen Teil eines Handbuchs zu einer Matlab-Implementierung eines EKF von Joan Solà [15].

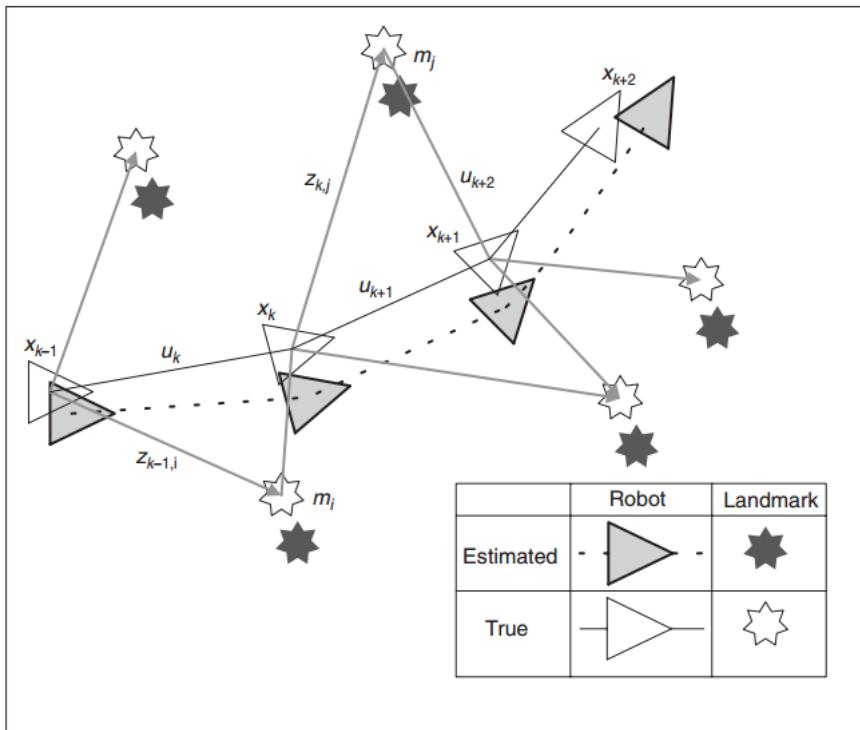


Abbildung 16: Grundsätzliche Idee von SLAM [4]

7.1 Extended Kalman Filter

Das Extended Kalman Filter (EKF) arbeitet mit Wahrscheinlichkeitsdichten und Korrelation der Roboter- und Landmarkenzustände. Dabei wird ein Mittelwertvektor X und eine Kovarianzmatrix P verwendet. X enthält die Erwartungswerte der Roboter- und Landmarkenkoordinaten.

Der Roboterzustand wird mit drei Freiheitsgraden dargestellt: x-Koordinate, y-Koordinate und Orientierung. Ihre Erwartungswerte belegen die ersten drei Elemente von X . Eine Landmarkenzustand besteht aus x- und y-Koordinate. Bekannte Landmarken bilden mit je zwei Einträgen den restlichen X -Vektor.

P ist eine $n \times n$ -Matrix mit $n = \dim X$. P enthält Informationen sowohl über die Varianzen der Wahrscheinlichkeitsdichten der Zustände des Roboters und einzelner Landmarken als auch über die Korrelationen, die zwischen Roboter- und Landmarkenzuständen bestehen.

Nach jeder abgeschlossenen Roboterbewegung, deren Odometrie bekannt ist, kann ein EKF-Update durchgeführt werden. Ein solches Update besteht aus den Folgenden Schritten:

Odometrisches Update Die Wahrscheinlichkeitsdichte des Roboters wird über das Bewegungsmodell mit den bekannten Odometriedaten und Rauschen aktualisiert.

Landmarkenzuordnung Die Messdaten des Infrarotsensors werden bereits bekannten Landmarken zugeordnet (siehe Abschnitt 7.1.3).

Reobservations-Update Für jede gefundene Zuordnung von Landmarke zu einem Satz Messdaten wird nacheinander ein Reobservations-Update durchgeführt. Dabei werden Messdaten mit erwarteten Messdaten verglichen. Unter Berücksichtigung der Unsicherheit des Roboterzustandes und der Unsicherheit der Messdaten wird die Wahrscheinlichkeitsdichte des Roboterzustandes angepasst. Darüber hinaus werden anhand der Kovarianz-Matrix P auch die Wahrscheinlichkeitsdichten aller Landmarken angepasst.

Hinzufügen neuer Landmarken Für jeden Satz an Messdaten die nicht einer bekannten Landmarke zugeordnet werden konnten, wird nun eine neue Landmarke in X und P eingetragen. Die Varianz und Korrelation wird dabei aus der aktuellen Varianz der Wahrscheinlichkeitsdichte des Roboters und der Rauschmatrix des Sensors berechnet.

7.1.1 Interface

Das EKF ist in eine Klasse *Slam* gekapselt, die X und P als inneren Zustand enthält. Die Implementierungen der Crawler-Klasse besitzen ein Slam-Objekt. Die Methode *slamIt* kann ausgeführt werden um einen Slam-Update durchzuführen. Als Parameter werden der Odometrie-Kontrollvektor u , die odometrische Rausch-Matrix N , die Messdaten des Sensors y_list und die Sensor-Rausch-Matrix R erwartet. Der Zustand kann nach einem Slam-Update mit den Methoden *getX* und *getP* abgerufen werden. Darüber hinaus existieren noch öffentliche Felder, die diverse Debug-Informationen enthalten.

7.1.2 Modelle

Bewegungsmodell Das Bewegungsmodell beschreibt mit einer Funktion, wie der Roboterzustand durch eine Roboterbewegung verändert wird. Der Roboterzustand wird mit drei Freiheitsgraden dargestellt: x-Koordinate, y-Koordinate und Orientierung. Die Roboterbewegungen werden in Form eines Kontrollvektors $u = (u_1, u_2, u_3)$ kodiert. u_1 beschreibt die vorwärts/rückwärts Translation des Roboters, u_2 die seitwärts Translation und u_3 die Änderung der Orientierung des Roboters. In dem Modell werden zunächst die Translationen und erst anschließend die Orientierungsänderung durchgeführt (für die definierten Roboterbewegungen ist das jedoch nicht von Relevanz, da Translation und Rotation nur exklusiv zueinander auftreten). Damit können alle 6 Roboterbewegungen dargestellt werden. Die Formel 1 zeigt, wie sich aus dem bisherigen Roboterzustand $R = (r_1, r_2, r_3)$ und dem Kontrollvektor u der neue Roboterzustand R' ergibt. Die Schreibweise $R_{i,j}$ beschreibt den Vektor (r_i, r_j) .

$$M_{rot} = \begin{bmatrix} \cos R_3 & -\sin R_3 \\ \sin R_3 & \cos R_3 \end{bmatrix} R' = \begin{bmatrix} R_{1,2} + M_{rot} * u_{1,2} \\ R_3 + u_3 \end{bmatrix} \quad (1)$$

Sensormodell Das Sensormodell beschreibt wie bei gegebenen Roboterzustand R aus den Sensordaten y einer Landmarke die Landmarkenposition L berechnet werden kann. Die Sensordaten sind $y = (y_1, y_2)$. Wobei y_1 den (aus Sensorsicht) horizontalen Winkel und y_2 den vertikalen Winkel zwischen Landmarke und Infrarotsensor angibt. Da der Höhenunterschied $zDiff$ zwischen Sensor und Landmarke bekannt ist, kann aus y_2 die Distanz zwischen Sensor und Landmarke bestimmt werden. Die Formel 2 zeigt, wie L aus y und R berechnet wird.

$$diff = (zDiff / \tan y_2) \Theta = y_1 + R_3 L = \begin{bmatrix} R_1 + diff * \cos(\Theta) \\ R_2 + diff * \sin(\Theta) \end{bmatrix} \quad (2)$$

Inverses Sensormodell Das inverse Sensormodell beschreibt wie bei gegebenen Roboterzustand R aus der Position L einer Landmarke die erwarteten Sensordaten y berechnet werden können. Die Formel 3 zeigt die Berechnung.

$$L' = \begin{bmatrix} \cos X_3 & \sin X_3 \\ -\sin X_3 & \cos X_3 \end{bmatrix} * \begin{bmatrix} L_1 - R_1 \\ L_2 - R_2 \end{bmatrix} \quad (3)$$

$$y = \begin{bmatrix} \text{atan2}(L'_2, L'_1) \\ \text{atan2}(diffZ, \|L'\|) \end{bmatrix} \quad (4)$$

$zDiff$ ist wie im Sensormodell die gegebene Höhendifferenz zwischen Landmarke und Sensor.

7.1.3 Landmarken Zuordnung

Szenario Der Sensor des Roboters liefert eine Liste an Messdaten. Es wird versucht, möglichst viele Messdaten bereits bekannten Landmarken zuzuordnen. Für zugeordnete Landmarken wird

ein Reobservationsupdate durchgeführt. Übrig gebliebene Messdaten werden anschließend als neue Landmarken eingetragen. Für die Zuordnung einer Landmarke L zu einem Messwert y kann ein Zuordnungswert $value(L, y)$ berechnet werden. Mit Hilfe der Mahalanobis-Distanz $mDist$ von y zur Wahrscheinlichkeitsdichte der erwarteten Messung von L (unter Berücksichtigung des Messfehlers) kann der Zuordnungswert berechnet werden: $value = threshold - mDist$. $threshold$ beschreibt einen festgelegten Schwellwert. Wenn er von der Distanz überschritten wird (wenn also $value$ negativ ist), dann ist die Zuordnung ungültig und wird nicht weiter in Erwägung gezogen. Es werden gültige Zuordnungen (L_i, y_i) gesucht, sodass $\sum value(L_i, y_i)$ maximal ist. Das Zuordnungsszenario entspricht einem Zuordnungsproblem in einem gewichteten bipartiten Graphen.

Implementierung Für die Zuordnung haben wir eine einfach zu implementierende erschöpfende Suche gewählt. Diese ist als rekursive Funktion implementiert. Im allgemeinen Fall führt das zu einer faktoriellen Laufzeit. Für diesen Anwendungsfall gibt es aber zwei Faktoren, die die Laufzeit limitieren: Zum Einen werden vom Roboter Sensor maximal 4 Landmarken gleichzeitig erkannt, wodurch die Rekursionstiefe auf 4 beschränkt ist. Dadurch entsteht für den Algorithmus die Worst-Case Laufzeit von $\mathcal{O}(n^4)$, was bereits einer langsam Implementierung des Ungarischen Algorithmus entspricht. Zum Anderen werden nur Landmarken für die Zuordnung in Erwägung gezogen, deren Mahalanobis-Distanz zum Messwert unter einem Schwellwert liegt. Bei praktischen Szenarien trifft das erwartungsgemäß nur auf ein bis zwei Landmarken zu. Die Erwartungslaufzeit liegt damit bei $\mathcal{O}(n)$. Diese Schätzungen werden in der Evaluation bestätigt.

Optimierungspotential In der momentanen Implementierung werden die Zuordnungswerte wiederholt berechnet. Für größere Szenarien kann die Laufzeit der vorhandenen Zuordnungsimplementierung optimiert werden, indem die Zuordnungswerte gecached werden. Zusätzlich kann die Vorberechnung der Wahrscheinlichkeitsdichten der bekannten Landmarken im Messraum die Laufzeit auf ein Viertel senken (da sie sonst für jeden Messwert wiederholt neu berechnet werden). Wird der Roboter in Zukunft mit einem Sensor ausgestattet, der mehr als vier Landmarken erkennen kann und/oder größere Reichweite hat, so wird es lohnenswert den Zuordnungsalgorithmus durch die ungarische Methode zu ersetzen.

8 Evaluation

8.1 Simulations Batch-Ausführung

8.1.1 Known Issues

Die Evaluation unserer SLAM-Implementierung haben wir hauptsächlich software-seitig mit Hilfe unserer Simulationsumgebung durchgeführt, die bereits in Kapitel 6 im Detail besprochen worden ist. Um ein aussagekräftiges Ergebnis zu erhalten, haben wir die 1000 Wiederholungen in der Simulationen unter möglichst den selben Bedingungen durchgeführt. Dabei haben wir bei jeder Durchführung die gleiche Roboterbewegungen und die gleiche Anzahl an Landmarken verwenden. Die Positionen der Landmarken variieren von Durchgang zu Durchgang.

8.1.2 Szenario

Die Landmarken werden zufällig in einer quadratischen Fläche von 16 cm^2 platziert. Die Zahl der Landmarken beträgt acht. Der verwendete Satz an Roboterbewegungen ist zuvor per Hand mit dem GUI-Pathplaner aufgezeichnet worden und beschreibt einen Pfad, der den Landmarken-Cluster mehrfach durchfährt und ihn vom Rand her aus unterschiedlichen Winkeln betrachtet. Es werden insgesamt 140 Schritte durchgeführt.

Für die Testdaten werden die Crawler-Bewegungen und Kameramessungen mit einem normalverteilten Fehler von bis zu 4 mm in der Bewegung und bis zu 2° in der Rotation bei jedem Bewegungsschritt simuliert.

Dieses Szenario wird 1000 mal ausgeführt.

8.1.3 Ergebnisse

Fehler In 12,6% der Fälle liefert der Slam Algorithmus ein fehlerhaftes Ergebnis. Als fehlerhaft wird das Resultat bewertet, wenn die Anzahl der erkannten Landmarken die tatsächliche Anzahl von acht übersteigt. Fehlerhafte Ergebnisse sind von der weiteren Evaluation ausgeschlossen, da Phantomlandmarken leicht zu großen Positionsfehlern führen, die wiederum neue Phantomlandmarken erzeugen. Diese Kaskade erzeugt stark abweichende Daten, die den Durchschnitt verfälschen können.

Positionsabweichung Die durchschnittliche Positionsabweichung liegt bei etwa 2.5 mm, wobei die Positionsabweichung zwischen 2.7 mm und 2.3 mm variiert am Ende einer Durchführung. Die durchschnittliche Varianz beträgt also 0.000131.

Auswertung Die Positionsabweichung und Varianz der Landmarken bei den erfolgreichen Durchläufen ist zufriedenstellend. Ein bekannter, noch nicht behobener, Fehler, der Messdaten um 180° versetzt zuordnet, ist vermutlich für den Großteil der Fehler verantwortlich. Der Fehler ist in Abbildung 17 illustriert.

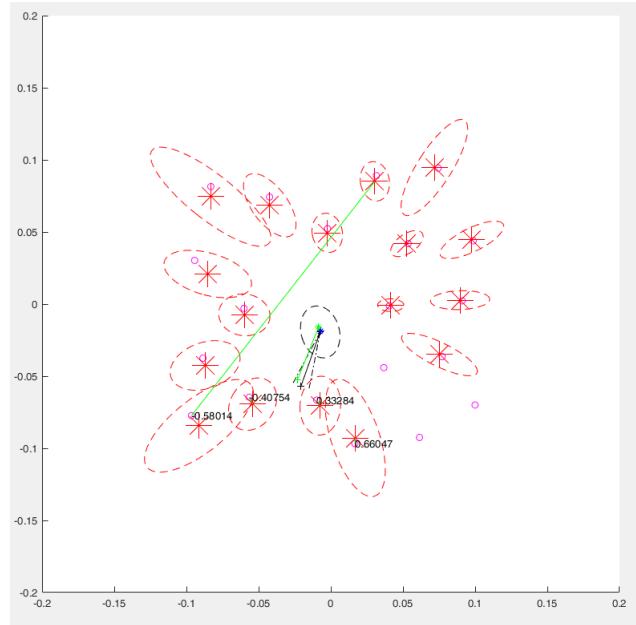


Abbildung 17: Auftreten eines Fehlers im SLAM-Algorithmus. Die lange grüne Linie kennzeichnet die fehlerhafte Zuordnung.

8.2 Laufzeittests zur Landmarkenzuordnung

Um den Zuordnungsalgorithmus zu testen haben wir zwei Szenarien durchgeführt und die durchschnittliche Anzahl der Aufrufe der rekursiven Zuordnungsfunktion, bei variierender Anzahl von Landmarken, aufgezeichnet. Die übrigen Parameter gleichen dem vorherigen Szenario der Batch-Ausführung. Bei einer Anzahl von 8 Landmarken beträgt die durchschnittliche Anzahl 5,78 Aufrufe. Bei einer Anzahl von 20 Landmarken liegt die Zahl bei 11,29. Diese Daten stützen die Vermutung des linearen Ausführungsaufwands.

9 Fazit und zukünftige Arbeiten

9.1 Fazit

Das Ziel dieses Praktikum war die Entwicklung von Hard- und Software, die einem ISAS-Crawler die Navigation und Kartografierung einer Umgebung mit Infrarotlandmarken erlaubt. Dafür sollten eine Infrarotkamera für den Crawler, Infrarotlandmarken für die Testumgebung und ein SLAM-Algorithmus für die Lokalisierung und Kartografierung entworfen werden.

Aus diesem Projekt ging eine funktionsfähige Implementierung eines Erweiterten Kalman Filters hervor. Für die Evaluation dieses Filters mit echten oder Testdaten wurde eine Simulationsumgebung implementiert. Der ISAS-Crawler wurde um eine Platine mit einer Infrarotkamera und um eine Softwareschnittstelle zum Abfragen der Kameradaten erweitert. Für die Evaluation und den Aufbau von zukünftigen Experimenten wurden Infrarotlandmarken entworfen und gefertigt.

Die Evaluation mit den simulierten Testdaten zeigt, dass das implementierte Kalman-Filter funktioniert. Mit den Daten der Testläufe mit dem ISAS-Crawler weißt unser Algorithmus eine sehr hohe Fehlerrate auf. Die Gründe dafür sind ein hoher Messfehler der Infrarotkamera und die falsche Erkennung von reflektiertem Infrarotlicht als Infrarotlandmarke. Dieser Messfehler kann in zukünftigen Arbeiten durch eine Kamerakalibrierung minimiert werden und die Auswirkungen der Falscherkennung von Landmarken kann durch Konfiguration der Sensibilität der Kamera reduziert werden. Beide Maßnahmen werden im nächsten Abschnitt 9.2 erläutert.

9.2 Zukünftige Arbeiten

9.2.1 Kamerakalibrierung

Um präzise Messungen zu erhalten, müssen die intrinsischen und extrinsischen Parameter der Wii-Kamera berechnet werden. Die verwendete Matlab-Umgebung bietet bereits Hilfsfunktionen für eine solche Kamerakalibrierung¹. Anstatt des üblichen Schachbrett-Musters muss für die Wii-Kamera eine Infrarot-LED-Matrix entworfen werden, um Referenzdaten für die Kalibrierung zu erzeugen. Dabei ist die Einschränkung der Wii-Kamera auf maximal vier erkannte Punkte pro Bild zu beachten.

9.2.2 Kamerakonfiguration

Die verwendete Wii-Infrarotkamera erkennt in jedem Bild die vier hellsten Infrarotpunkte. Sie verfügt über eine Bildvorverarbeitung, die mehrere eng zusammen liegende Punkte zu einem

¹<https://de.mathworks.com/help/vision/camera-calibration.html>

zusammenfassen kann. Die Konfiguration der Sensibilität und der Bildvorverarbeitung wird nur durch eine inoffizielle Dokumentation beschrieben [7]. Eine Untersuchung der Kameraparameter könnte zu einer verringerten Fehlerkennungsrate führen.

9.2.3 Kombiniertes Infrarot- und Ultraschall-SLAM-Verfahren

Sensoren und Software für ein ultraschallbasiertes SLAM-Verfahren mit dem verwendeten Crawler sind bereits vorhanden. Die Kombination der SLAM-Karte aus dem Ultraschallverfahren mit der aus unserem Infrarotverfahren könnte zu einer Karte führen, die zuverlässigeren und präziser ist als die Karten der einzelnen Verfahren.

9.2.4 Kommunikation zwischen mehreren Crawlern

Unsere Hardware ist für Szenarien ausgelegt, in denen mehrere Crawler in einem Versuchsaufbau agieren. Da die Crawler über ein WLAN-Netzwerk miteinander kommunizieren können, ist ein Informationsaustausch zwischen mehreren Crawlern denkbar. Sobald ein Crawler die Position des anderen mit einer Mindestpräzision bestimmt hat, können beide Crawler Informationen über die ihnen bekannte Karte austauschen. Das erhöht die Anzahl an Sensormessungen, die für die Navigation verfügbar sind und liefert besonders für SLAM-Verfahren interessante Sensordaten aus unterschiedlichen Positionen und Sichtwinkeln.

Literatur

- [1] Beschreibung der technischen Details des Arduino Yun, Aufgerufen am 02.August 2017. <https://www.arduino.cc/en/Main/ArduinoBoardYun>.
- [2] Getting Started with the Arduino Yun, Aufgerufen am 02.August 2017. <https://www.arduino.cc/en/Guide/ArduinoYun>.
- [3] Jose A Castellanos, JMM Montiel, José Neira, and Juan D Tardós. The spmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15(5):948–952, 1999.
- [4] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.
- [5] Achim Hekler and Daniel Lyons. Beschreibung eines Miniaturisierten Laufroboters, Aufgerufen am 02.August 2017. <http://isas.uka.de/Research>.
- [6] Hyla. ATM18::Projekt::Wii Remote IR Sensor, 2008. <http://cc-zwei.de/wiki>.
- [7] Eiji Kako. Connect the infrared sensor of the Wii remote control to the microcomputer and try using (+ mouse), 2008. <http://kako.com/neta>.
- [8] Frédéric Leens. An introduction to i 2 c and spi protocols. *IEEE Instrumentation & Measurement Magazine*, 12(1):8–13, 2009.
- [9] Nintendo - official, Aufgerufen am 02.August 2017. <https://www.nintendo.com/>.
- [10] Nintendo - official: Wii, Aufgerufen am 02.August 2017. <http://www.nintendo.de/Wii/Wii-94559.html>.
- [11] Benjamin Noack, Daniel Lyons, Matthias Nagel, and Uwe D. Hanebeck. Nonlinear Information Filtering for Distributed Multisensor Data Fusion. In *Proceedings of the 2011 American Control Conference (ACC 2011)*, San Francisco, California, USA, June 2011.
- [12] Peter Recktenwald. Wii IR camera as standalone sensor, 2009. <http://robotshop.com/letsmakerobots>.
- [13] S Riisgard and M Blas. Slam for dummies (2004): A tutorial approach to simultaneous localizing and mapping.
- [14] Philips Semiconductors. The i2c-bus specification. *Philips Semiconductors*, 9397(750):00954, 2000.
- [15] Joan Sola. Simulataneous localization and mapping with the extended kalman filter. *unpublished. Available: http://www.joansola.eu/JoanSola/eng/JoanSola.html*, 2013.