

# Projektbericht

Innenraumerfassung mit dem Turtlebot

Autor

Fabian Saccilotto

Advisor

Interstaatliche Hochschule für Technik NTB Buchs

Datum

Norbert Frei

30. Januar 2013

# Inhaltsverzeichnis

<b>1. Glossar</b>	<b>4</b>
<b>2. Ausgangslage</b>	<b>5</b>
2.1. Roboter Framework . . . . .	5
2.2. Turtlebot . . . . .	7
<b>3. ROS - Robot Operating System</b>	<b>9</b>
3.1. Allgemein . . . . .	9
3.2. Dateisystem . . . . .	10
3.3. Der Computation Graph . . . . .	11
3.4. Overlays . . . . .	12
<b>4. Simultaneous Localization and Mapping</b>	<b>14</b>
4.1. Particle Filter . . . . .	14
<b>5. Inbetriebnahme</b>	<b>17</b>
5.1. Lieferprobleme . . . . .	17
5.2. Arbotix Pincher Arm . . . . .	17
5.3. Kalibrierung . . . . .	19
5.4. Netzwerk . . . . .	20
<b>6. Erfassung</b>	<b>21</b>
6.1. Aufnahmestrategien . . . . .	21
6.2. Aufnahmeobjekte . . . . .	22
6.3. Ergebnisse . . . . .	23
6.4. Erweiterung 3D . . . . .	26
6.4.1. 3D Photogrammetry . . . . .	26
6.4.2. SLAM and Octomap . . . . .	26
6.4.3. RGBDSLAM . . . . .	27
6.5. Fazit . . . . .	28
<b>7. Schlussfolgerung</b>	<b>30</b>
7.1. Interessante Quellen . . . . .	31

<b>A. Inbetriebnahme des Roboterarms</b>	<b>35</b>
A.1. Arduino IDE . . . . .	35
<b>B. Installation Entwicklungsrechner</b>	<b>36</b>
B.1. Wireless unter Linux . . . . .	36
B.2. Turtlebot ROS . . . . .	36
B.2.1. SSH . . . . .	36
B.2.2. Chrony . . . . .	36
B.2.3. Overlay . . . . .	37
B.2.4. Udev Regeln . . . . .	37
B.2.5. Batteriekennung . . . . .	38
B.3. Anpassung Robotermodell . . . . .	38
B.4. Kalibrierung . . . . .	38
B.5. OpenNI . . . . .	39
B.6. Extraktion von Bilddaten . . . . .	39
B.7. Android Remote . . . . .	39

# 1. Glossar

**ROS - Robot Operating System** Verbreitetes Framework für die Ansteuerung von Robotern

**SLAM - Simultaneous Localization and Mapping** Aufnahme einer Karte mit einem mobilen Gerät und dessen gleichzeitige Positionierung

**RGBDSLAM** Algorithmus welcher in der Lage ist synchronisierte Farb- und Tiefebilder zu einem 3D-Modell zusammenzustellen

**Trajektorie** Bewegungspfad eines Roboters

**Odometrie** Positionsbestimmung eines mobilen Systems anhand der Daten seines Antriebssystems

**Roomba** Produktnname für Staubsaugerroboter der Firma iRobot

**iCreate** Produktnname für den Roomba-Entwicklungsroboter

**Voxel** Dreidimensionaler Bildpunkt

## 2. Ausgangslage

Im MSE Vertiefungsprojekt (Ersteinführung der Kinect am Institut INF [8]) wurden Methoden zur Rückgewinnung von Innenraummodellen mit Hilfe des Microsoft Kinect Sensors evaluiert. Da die Bewegung der Kinect aus Bildmerkmalen extrahiert wird, benötigen die Algorithmen viel Rechenleistung und Arbeitsspeicher.

SLAM (Simultaneous Localization and Mapping) Algorithmen werden eingesetzt um während der Bewegung eines mobilen Roboters eine Karte der Umgebung zu erstellen und diesen innerhalb dieser Karte zu positionieren. Aufgrund der Mobilität haben diese Algorithmen meist geringe Hardwareanforderungen [10].

Eine mittels SLAM erzeugte Karte, erweitert mit den Tiefeninformationen der Kinect, könnte für die kostengünstige Aufnahme von Innenräumen verwendet werden. Ziel dieser Arbeit ist es, ein derartiges System aufzubauen und zu evaluieren. Um die Entwicklung zu beschleunigen, soll eine Roboterplattform mit SLAM-Fähigkeit eingesetzt werden.

### 2.1. Roboter Framework

Für die Auswahl einer geeigneten Plattform wurde nach verschiedenen Roboter-Frameworks recherchiert:

- ROS - Robot Operating System

<http://ros.org/wiki/ROS/Introduction>

Schwerpunkt bei der Wiederverwendung von bestehender Software. Teile anderer Systeme sind darin integriert (Player's Simulation) oder verfügen über Schnittstellen (MRPT, Orocosp).

- YARP - Yet Another Robot Platform

<http://eris.liralab.it/yarpdoc/index.html>

Bibliotheken für die Datenkommunikation, Signalverarbeitung sowie Geräteschnittstellen.

- Carmen  
<http://carmen.sourceforge.net>  
 Schwerpunkt Navigation (Pfadplanung, Lokalisierung, Kartografie).
- Orocosp  
<http://www.orocos.org/>  
 Schwerpunkt Echtzeitsysteme, Roboterkinematik und Filtertechniken (Kalman-, Partikelfilter etc.)
- MRPT - Mobile Robot Programming Toolkit  
<http://www.mrpt.org/>  
 Schwerpunkte SLAM, Computer Vision und Pfadplanung
- Player  
<http://playerstage.sourceforge.net/>  
 Bibliotheken für Datenkommunikation sowie 2D und 3D Simulation. Der 3D Simulator Gazebo wurde zu ROS migriert.
- MOOS - Mission Orientated Operating Suite  
<http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php>  
 Schwerpunkt Maritime Roboter. Bibliotheken für die Matlab Anbindung.
- MRDS - Microsoft Robotics Developer Studio  
<https://www.microsoft.com/robotics/>  
 Kommunikationsbibliotheken, Visuelle Programmierung, Simulation

Alle Systeme bieten Bibliotheken für die einfache Kommunikation von Komponenten auf dem selben Rechner sowie in einem Netzwerk. Frameworks mit Unterstützung für erwerbbare Roboter oder SLAM sind in Tabelle 2.1 aufgelistet.

Framework	SLAM	Anzahl Roboter
ROS <sup>a</sup>	●	30
Carmen <sup>b</sup>	●	5
Orocosp	(nur Filter)	
MRPT	●	
MOOS	●	
MRDS <sup>c</sup>		6

<sup>a</sup> <http://www.ros.org/wiki/Robots>

<sup>b</sup> <http://carmen.sourceforge.net/hardware.html>

<sup>c</sup> <https://www.microsoft.com/robotics/#Robots>

Tabelle 2.1.: SLAM-Fähigkeit und Robotersysteme

## 2.2. Turtlebot

Unter dem Namen *Turtlebot* bietet die Firma [Willow Garage](#) - die Entwickler von ROS - einen kostengünstigen Entwicklungsroboter an. [Clearpath Robotics](#) bietet den Turtlebot komplett montiert für 1400 USD an. Im Gegensatz dazu kostet der für Carmen verwendete Pioneer-3DX in der Grundversion 4500 USD<sup>1</sup>.

Folgende Punkte waren ausschlaggebend für die Wahl dieser Plattform:

- Tiefer Preis
- Videos<sup>2</sup> von erfolgreicher Innenraumerfassung (siehe Abbildung 2.1)
- Turtlebot SLAM Tutorials [5]
- Verbreitung und Funktionalität von ROS

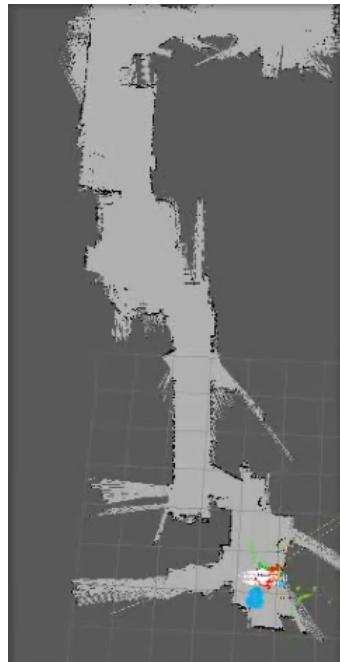


Abbildung 2.1.: Turtlebot SLAM

Der Turtlebot baut auf einem Roomba iCreate auf, verfügt über Encoder sowie ein Gyroskop und wird von einem Lenovo X130e Netbook angesteuert (siehe Abbildung 2.2).

---

<sup>1</sup>Der Hersteller des Pioneer - [Adept Mobile Robots](#) - hat für diese Aufgabenstellung gar ein System für 44'000 USD offeriert.

<sup>2</sup>Youtube [Navigation with the Turtlebot](#)

Spezifikationen Lenovo X130e:

- AMD 1.3 GHz E-300 APU (Accelerated Processing Unit)
- AMD Radeon HD 6310 GPU (integriert in APU)
- 2 GB RAM



Abbildung 2.2.: Turtlebot Entwicklungsroboter

Mittels einer Dreh- und Kippvorrichtung soll zudem der Aktionsradius der Kinect erweitert werden. Für spätere Experimente und Vorführungen wird weiter ein 5-Achsen Roboterarm montiert (Details siehe Abschnitt 5.2).

# 3. ROS - Robot Operating System

ROS ist ein Open Source, Meta-Betriebssystem für Roboter. Es stellt Dienste zur Verfügung, welche von einem Betriebssystem erwartet werden: Hardwareabstraktion, Gerätetreiber, Utilityfunktionen, Interprozess Kommunikation und Paketmanagement. Des Weiteren sind Werkzeuge und Bibliotheken für das Beziehen<sup>1</sup>, Builden, Schreiben und Ausführen von Code über mehrere Computer vorhanden. ROS kann in einigen Aspekten mit anderen Roboterframeworks verglichen werden. Dazu gehören: Player, YARP, Orocosp, CARMEN, Orca, MOOS sowie Microsoft Robotics Studio. ROS ist kein Echtzeit Framework obwohl es möglich ist ROS mit Echtzeitkomponenten zu kombinieren.[4]

Um den Einstieg zu erleichtern ist in diesem Kapitel eine Einführung zu den Konzepten von ROS zu finden. Viele Pakete von ROS sind gut dokumentiert, es lohnt sich jedoch die Konzepte detailliert zu studieren und die Einführungstutorials durchzuführen<sup>2</sup>. Das *Cheat Sheet* ist ebenfalls sehr nützlich und auf der Hauptseite des ROS-Wikis zu finden.

## 3.1. Allgemein

Der ROS Laufzeitgraph (Computation Graph) ist ein Peer-to-Peer Netzwerk von Prozessen, welche via die ROS Kommunikationsinfrastruktur lose gekoppelt sind. ROS stellt über das Paket *ros\_comm* verschiedene Kommunikationsarten zur Verfügung.

- Synchron über Services
- Asynchron über Topics
- Datenspeicher auf dem Parameter Server

---

<sup>1</sup>ROS stellt Werkzeuge zur Verfügung, welche Code aus Quellcodeverwaltungen wie SVN, Git und Mercurial beziehen und in den Workspace integrieren kann.

<sup>2</sup><http://www.ros.org/wiki/de/ROS/StartGuide>

Namen haben eine wichtige Funktion im Computation Graph. Nodes, Topics, Services und Parameter verfügen alle über einen Namen. Damit die Komponenten auch in anderen Laufzeitumgebungen funktionieren, können Namen über die *ROS client library* während der Laufzeit neu zugeordnet werden.

Der Hauptzweck von ROS ist die Wiederverwendung von Code in der Roboterforschung und -entwicklung. ROS ist ein verteiltes System von Prozessen (Nodes), welches die lose Kopplung von individuellen Komponenten ermöglicht. Für die einfache Handhabung und Verteilung werden diese in Paketen und Stacks organisiert. ROS unterstützt zudem den Zusammenschluss von Code Repositories, wodurch auch die Zusammenarbeit über verteilte Infrastrukturen ermöglicht wird. Dieses Design, vom Dateisystem bis zur Community, ermöglicht unabhängige Entscheidungen bezüglich Entwicklung und Implementierung, welche mit Hilfe der ROS Infrastrukturwerkzeuge vereint werden können.

Weitere Zielsetzungen:

**Thin** Für ROS geschriebener Code soll auch mit anderen Roboter Frameworks verwendet werden können. ROS wurde bereits erfolgreich mit anderen Systemen kombiniert (darunter OpenRAVE, Orocosp, Player).

**ROS-agnostic libraries** Das bevorzugte Entwicklungsvorgehen ist Bibliotheken mit klaren Schnittstellen zu schreiben, welche nicht an ROS gebunden sind.

**Language independence** Das ROS Framework kann einfach in jeder modernen Programmiersprache implementiert werden. Es bestehen Implementierungen in Python, C++ und Lisp, für Java und Lua existieren experimentelle Bibliotheken.

**Easy testing** ROS verfügt über ein eingebautes Unit- und Integrationstest Framework *rostest*.

**Scaling** ROS funktioniert in grossen Laufzeitumgebungen sowie auch für grosse Entwicklungsprozesse.

Zurzeit läuft ROS nur auf Unix-basierten Plattformen, hauptsächlich Linux Ubuntu und Mac OS X. Eine Portierung für Microsoft Windows ist im Gange, jedoch noch im experimentellen Stadium.

Das ROS Kernsystem wird zusammen mit nützlichen Werkzeugen und Bibliotheken regelmässig als ROS Distribution ausgeliefert. Namen von Distributionen sind Diamondback, Electric, Fuerte und aktuell Groovy.

## 3.2. Dateisystem

Folgende Ressourcen findet man im Dateisystem:

**Packages** Pakete sind die hauptsächliche Art der Softwareorganisation in ROS.

Das Paketmanifest *manifest.xml* enthält Metadaten über das Paket. Dazu gehören Lizenzinformationen und Abhängigkeiten sowie sprachspezifische Einstellungen wie Compilerflags. Ein Paket kann folgende Elemente enthalten:

- Laufzeitprozesse (Nodes)
- Von ROS abhängige Bibliotheken
- Datensätze
- Konfigurationsdateien
- Alles was sonst noch zu diesem Kontext gehört

**Stacks** Stacks sind eine Sammlung von Paketen, welche zusammen eine Funktionalität bereitstellen (z.B. der *navigation stack*). Zudem wird ROS Software in Form von Stacks ausgeliefert und versioniert. Das Stack Manifest *stack.xml* enthält Metadaten zum Stack. Dazu gehören Lizenzinformationen sowie Abhängigkeiten zu anderen Stacks.

**msg** Die Datenstruktur von Nachrichten (*messages*).

**srv** Anfrage- und Antwortdatenstrukturen für Dienstleistungen (*services*).

## 3.3. Der Computation Graph

Der Computation Graph ist ein Peer-to-Peer Netzwerk von ROS Prozessen, welche gemeinsam eine Aufgabe erfüllen (siehe Abbildung 3.1). Der Graph bezieht seine Daten aus folgenden Komponenten:

**Nodes** Knoten sind Prozesse, welche Berechnungen durchführen. ROS ist modular ausgelegt, deshalb arbeiten in einem Robotersystem normalerweise viele Knoten zusammen. Ein Knoten setzt auf einer ROS client library wie roscpp oder rospy auf. Knoten kommunizieren direkt untereinander, die Verbindungsinformationen erhalten sie vom Master. Das Standardkommunikationsprotokoll ist TCPROS.

**Master** Über den ROS Master können Namen registriert oder nachgeschlagen werden. Ohne den Master wären die Knoten nicht in der Lage zu kommunizieren.

**Parameter Server** Der Parameterserver ist Teil des Masters und speichert Daten zu einem Schlüssel an zentraler Stelle (Key-Value Store).

**Messages** Knoten kommunizieren über Nachrichten. Eine Nachricht ist eine Datenstruktur mit typisierten Feldern. Standard Datentypen (Integer, Float, Boolean etc.) und Arrays derselben werden unterstützt. Nachrichten können beliebig verschachtelte Strukturen enthalten.

**Topics** Nachrichten werden über einen Publisher-Subscriber Mechanismus verteilt. Ein Knoten veröffentlicht Nachrichten zu einem Thema (publish to a topic) identifiziert über einen Namen. Ein anderer Knoten, welcher an diesen Nachrichten interessiert ist, abonniert das Thema (subscribe). Es können mehrere Knoten gleichzeitig als Publisher und Subscriber fungieren und ein Knoten kann mehrere Themen veröffentlichen oder abonnieren. Allgemein gesagt wissen Publisher und Subscriber nichts voneinander. So werden Erzeugung und Konsum von Informationen entkoppelt.

**Services** Der Publisher-Subscriber Mechanismus erlaubt eine flexible, einseitige n-zu-n Kommunikation. Diese ist jedoch für eine Anfrage-Antwort Interaktion zwischen den Knoten nicht geeignet. Für diese Art der Kommunikation bieten Knoten Services an, identifiziert durch einen Namen. Der aufrufende Knoten sendet eine Anfrage und wartet auf die Antwort.

**Bags** Bags dienen der Ablage und Wiedergabe von ROS Nachrichten. Mit Bags können beispielsweise Sensordaten aufgezeichnet werden welche für die Entwicklung nötig sind, jedoch grossen Aufwand zur Aufzeichnung benötigen.

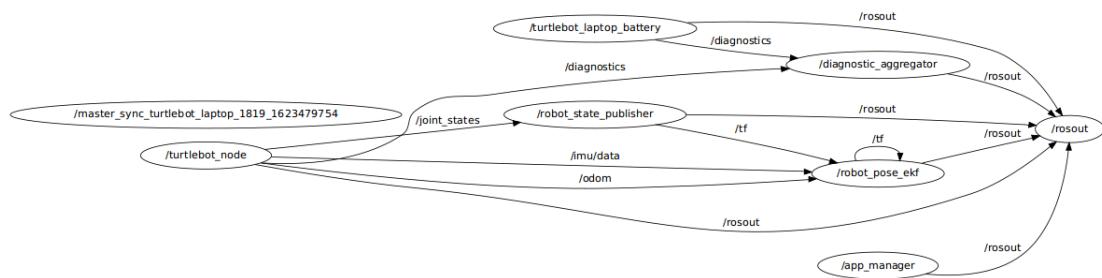


Abbildung 3.1.: Minimaler Graph eines Turtlebot-Systems

## 3.4. Overlays

Üblicherweise wird ROS unter Linux mit Hilfe der Paketmanager (Apt, Dpkg) installiert. Die vorkompliierten Quellen werden dabei im schreibgeschützten Verzeichnis für Drittsoftware (*/opt*) abgelegt. Für Anpassungen wird ein Overlay-Verzeichnis erstellt (siehe Abbildung 3.2). Hinweise zur Erstellung von Overlays befinden sich im Anhang B.2.3.

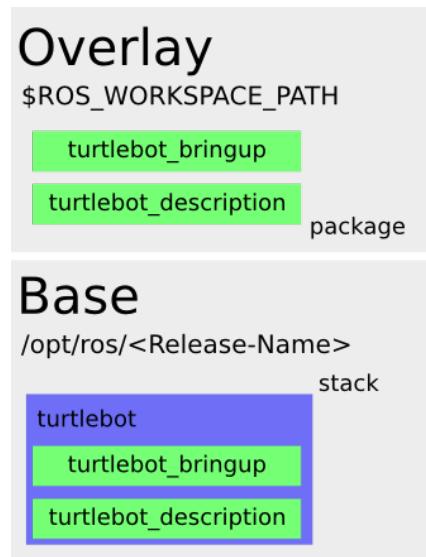


Abbildung 3.2.: Das Overlay überschreibt Teile des Turtlebot Stacks

# 4. Simultaneous Localization and Mapping

Der eingesetzte SLAM Algorithmus [3] - genannt GMapping - ist ein *Rao-Blackwellized Particle Filter*<sup>1</sup>.

## 4.1. Particle Filter

Partikelfilter gehören zu den sequenziellen Monte-Carlo Methoden zur Schätzung unbekannter Zustände mit Hilfe von Wahrscheinlichkeiten. Für Systeme mit linearer oder gauss'scher Verteilung werden Kalman-Filter eingesetzt. Ansonsten erzielen Partikelfilter bessere Resultate.

Man nehme eine Menge von Partikeln, welche mögliche Zustände verwalten. Anhand der Messung des aktuellen Zustandes wird eine Verteilung für die zukünftige Änderung berechnet (*Proposal Distribution*). Jedes Partikel wird neu geschätzt (*Prediction*) und gewichtet (*Importance Sampling*). Die Gewichtung entscheidet darüber, welche Partikel für die nächste Sequenz verwendet werden (*Resampling*). Das Verfahren ist grafisch in Abbildung 4.1 dargestellt.

Im Fall von GMapping stellen die Zustände Robotertrajektorien und daraus resultierende Karten dar. Die bestmögliche Messung des Zustandes erfolgt über die Odometrie sowie einen Laser Scanner. Durch Vergleich der Tiefeninformationen des Lasers werden Wahrscheinlichkeitszonen für die Schätzung der neuen Zustände identifiziert (siehe Abbildung 4.2). Für die Gewichtung wird die Odometrie sowie die Abweichung des Tiefenbildes berechnet. Um den Rechenaufwand zu verringern werden die Partikel erst reproduziert, wenn sie den Zustand ungenügend repräsentieren<sup>2</sup>. Je weniger Informationen für den Vergleich der Tiefenbilder zu Verfügung stehen, desto stärker ist der Einfluss der Odometrie auf die Gewichtung.

---

<sup>1</sup>Unter Rao-Blackwellization versteht man die Berechnung einer Karte aufgrund der Schätzung der Robotertrajektorie.

<sup>2</sup>Je grösser die Varianz der Gewichte ist, desto schlechter repräsentieren die Partikel den Zustand.

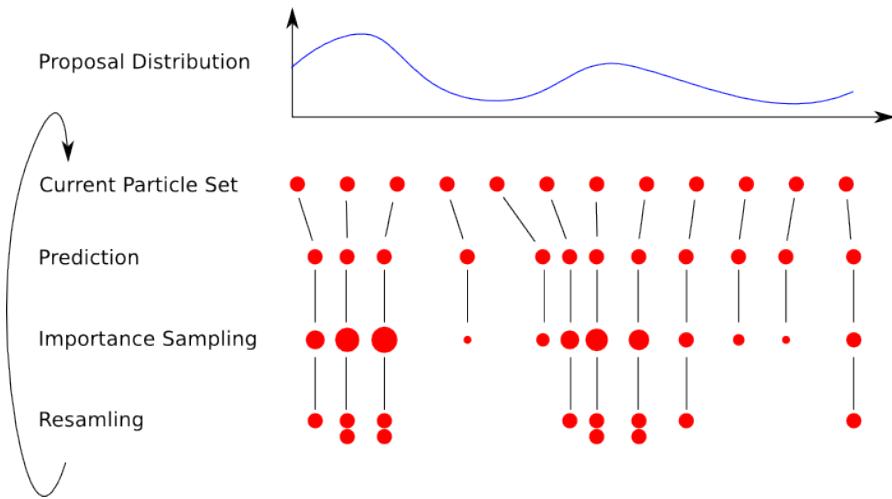


Abbildung 4.1.: 2D Particle Filter grafisch dargestellt

Die Autoren von GMapping arbeiteten mit einem Pioneer2 Roboter und waren in der Lage Gelände bis zu  $250\text{ m}^2$  mit einer Auflösung von 1cm bis 5cm zu kartografieren.

Ausstattung:

- SICK PLS Laser Scanner (180° Sichtfeld, 4m Reichweite)
- Pentium 4

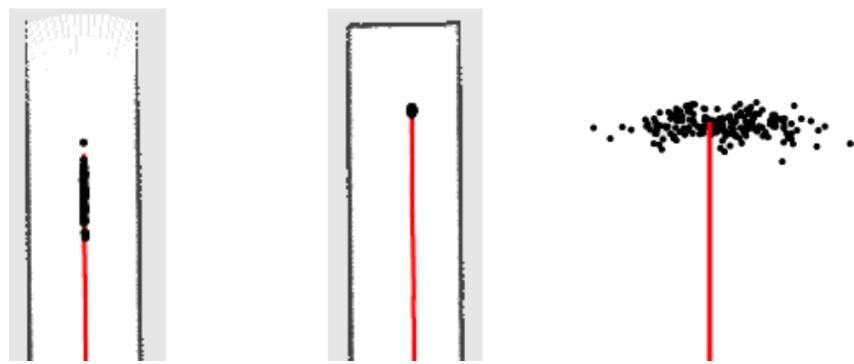


Abbildung 4.2.: Durch die Tiefeninformation wird die Schätzung der Partikel fokussiert.

# 5. Inbetriebnahme

Detaillierte Informationen zur Installation und Konfiguration der Software sind im Anhang B.

## 5.1. Lieferprobleme

Um Kompatibilitätsprobleme mit Betriebsspannungen und Anschlüssen zu umgehen, war geplant den Turtlebot beim europäischen Vertreiber<sup>1</sup> zu beziehen. Der Versand wurde jedoch Mitte 2012 eingestellt. Andere Händler aus den USA lieferten nicht nach Europa aufgrund von Bestimmungen für den Vertrieb bleihaltiger Materialien (ROHS [9]). Ein Ersatzaufbau mit Hilfe eines Roomba Staubsaugerroboters war wegen fehlendem Gyroskop ungeeignet. Aufgrund der Verwendung des Roboters für die Forschung konnte beim Bundesamt für Umwelt eine Einfuhrbewilligung eingeholt werden.

Clearpath Robotics - der Lieferant des aktuellen Systems - stellte für Mitte November eine Version mit verbesserter Odometrie in Aussicht. Der Lieferbeginn wurde jedoch auf Ende Januar 2013 verschoben, weshalb die neue Version für die vorliegende Arbeit nicht eingesetzt werden konnte.

## 5.2. Arbotix Pincher Arm

Für den Einsatz des Roboterarmes<sup>2</sup> sowie der Dreh- und Kippvorrichtung wurde der Turtlebot umgebaut (siehe Abbildung 5.1). Aufgrund des grossen Hebelarms schwingt die Kippvorrichtung während Beschleunigungsvorgängen im Bereich weniger Zentimeter. Bei Stillstand erfolgt jedoch die Bewegung der Kinect auf den Millimeter genau und ermöglicht präzise 360° Aufnahmen. Die detaillierte Inbetriebnahme des Armes ist im Anhang A beschrieben.

---

<sup>1</sup>[www.turtlebot.eu](http://www.turtlebot.eu)

<sup>2</sup><http://www.trossenrobotics.com/p/PhantomX-Pincher-Robot-Arm.aspx>

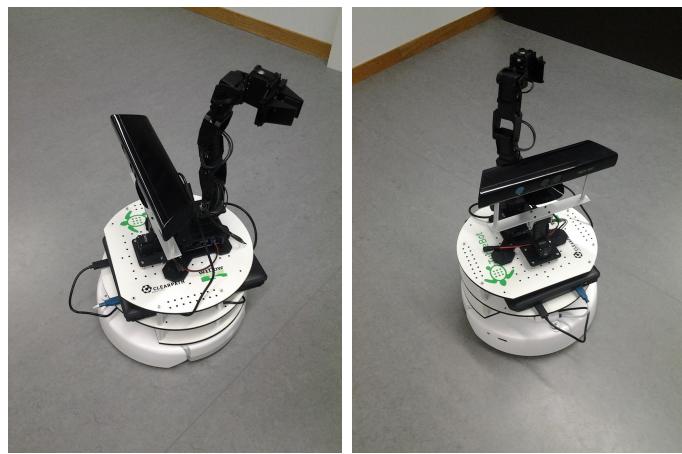


Abbildung 5.1.: Modifikation mit Arm, Dreh- und Kippvorrichtung

Die Ansteuerung der Servos erfolgt über die ROS-Stacks *simple-arms* und *arbotix*. Durch Verschieben interaktiver Marker können einerseits die Greiferposition, andererseits die einzelnen Achsen kontrolliert werden (siehe Abbildung 5.2). Die inverse Kinematik für die Armbewegung wird von ROS berechnet. Möglichkeiten für die Konfiguration von Safety-Zonen (Verhinderung von Kollisionen mit der Kinect) sind nicht vorhanden.

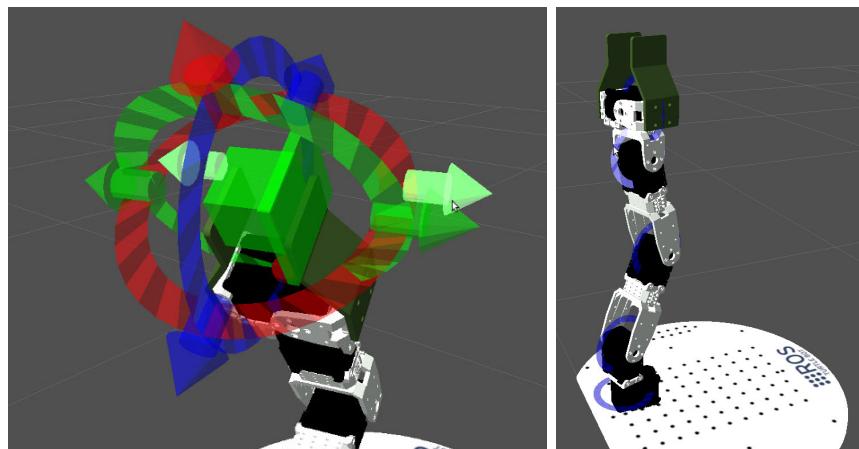


Abbildung 5.2.: Arm Ansteuerung  
links Greifer, rechts Achsen

Der iCreate Unterbau kann bis zu 1.5 A an periphere Geräte liefern. Die Kinect benötigt davon 1 A. Der Hersteller des Turtlebot rät davon ab den Roboterarm ebenfalls über den iCreate zu speisen. Aufgrund der begrenzten Zeit wurde deshalb im weiteren Verlauf des Projektes auf den Arm verzichtet.

### 5.3. Kalibrierung

Für die Kalibrierung der Odometrie ist in ROS ein Hilfsprogramm vorhanden. Eine Wand dient der Ermittlung der Nullposition zwischen Referenzdrehungen des Roboters mit unterschiedlichen Geschwindigkeiten [7].

Trotz erfolgreicher Kalibrierung scheiterten erste Kartografieversuche. Aufgrund der Positionierung der Kinect im vorderen Bereich des Roboters änderte sich das Koordinatensystem der Aufnahmen für die Weitergabe an den SLAM Algorithmus. Durch die Anpassung des RobotermODELLES (siehe Anhang B.3) konnte das Resultat verbessert werden. Um die Qualität der Kalibrierung visuell zu prüfen, kann die Punktewolke mit Ausblendeverzögerung dargestellt werden. Während der Bewegung des Roboters überlagern sich so die Tiefenbilder (siehe Abbildung 5.3).

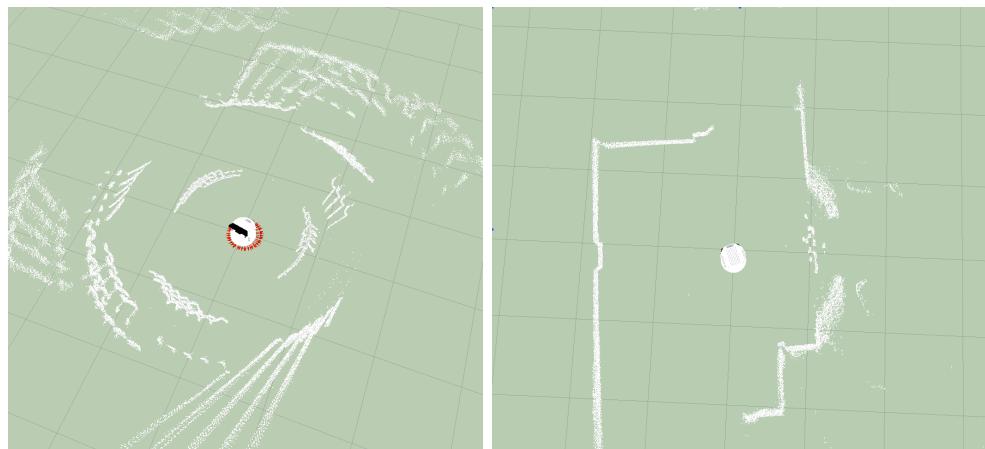


Abbildung 5.3.: Kalibrierungsqualität  
links schlecht, rechts gut

Eine Winkelabweichung der Odometrie von ca.  $5^\circ$  auf eine Umdrehung erschien dem Autor stets noch gross. Es wurden Versuche auf unterschiedlichen Bodenbelägen (Teppich, Parkett, Linoleum) und mit verschiedenen Drehgeschwindigkeiten durchgeführt. Es konnte jedoch keine signifikante Verbesserung festgestellt werden.

Damit für die Korrelation von Bildmerkmalen grössere Überlappungen entstehen, wurde der Turtlebot in die Originalversion zurückgebaut (siehe Abbildung 5.4).

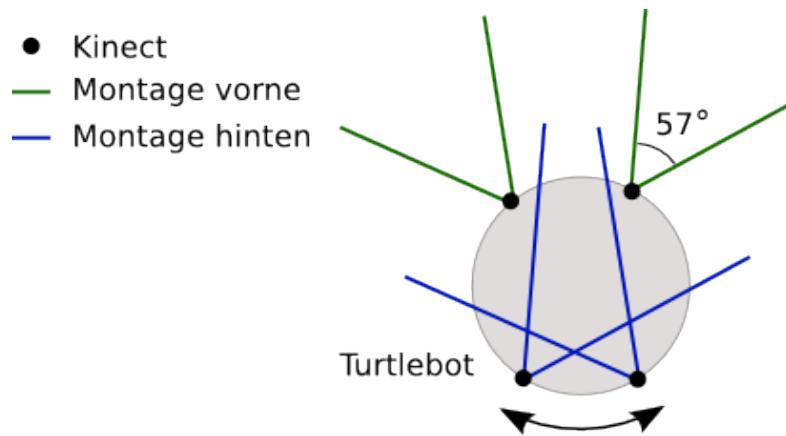


Abbildung 5.4.: Überlappung der Kameraaufnahmen bei unterschiedlicher Montageposition

## 5.4. Netzwerk

Aufgrund von Verbindungsproblemen mit diversen Wireless-Netzen des NTB traten Fehler bei der Kartografierung auf. Auf Anweisung des Informatik-Support wurde das System auf den neuesten Stand aktualisiert. Dies reduzierte die Ausfälle der Verbindung, benötigte jedoch eine neuere Version von ROS. Diese enthielt wiederum Fehler im Kinect Stack. Durch die Verwendung der aktuellsten Entwicklungsquellen konnten diese behoben werden.

Die Verkabelung des Netzwerkes brachte eine robustere Erfassung, die Qualität der Kartografierung verbesserte sich jedoch in geringem Masse.

# 6. Erfassung

## 6.1. Aufnahmestrategien

Für die Erfassung wurden verschiedene Methoden evaluiert (siehe Abbildung 6.1).

**Schnelldurchlauf** Der Roboter wird nahe den Wänden entlang durch den Raum navigiert. Als bald sich der Grundriss abzeichnet, wird die Drehrichtung geändert, um bis anhin verborgene Details aufzuzeichnen.

**Zick-Zack** Um lokal mehr Merkmale zu erhalten, wird der Roboter im Zick-Zack bewegt.

**Zellen** Nach der Idee der lokalen Optimierung wird jeweils ein Bereich in der Nähe des Roboters (Zelle) so gut als möglich aufgezeichnet. Sobald eine Zelle genügende Qualität erreicht, wird eine benachbarte Zelle aufgenommen.

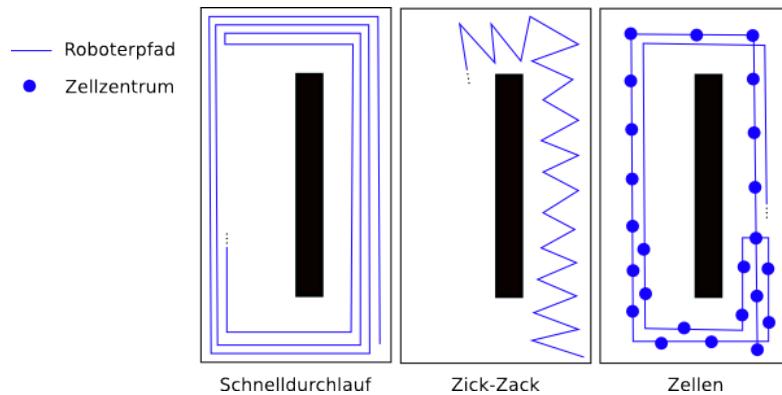


Abbildung 6.1.: Darstellung verschiedener Aufnahmestrategien

## 6.2. Aufnahmeobjekte

Zum Vergleich wurde die Erfassung in zwei Wohnungen, sowie einem Schulgebäude getestet (siehe Bilderserie 6.2 bis 6.4)<sup>1</sup>.

● Stuhl, Papierkorb, Karton usw.

■ Kasten > 2 m

□ Tisch, Kommode, Lavabo etc.

Abbildung 6.2.: Legende der Grundrisse

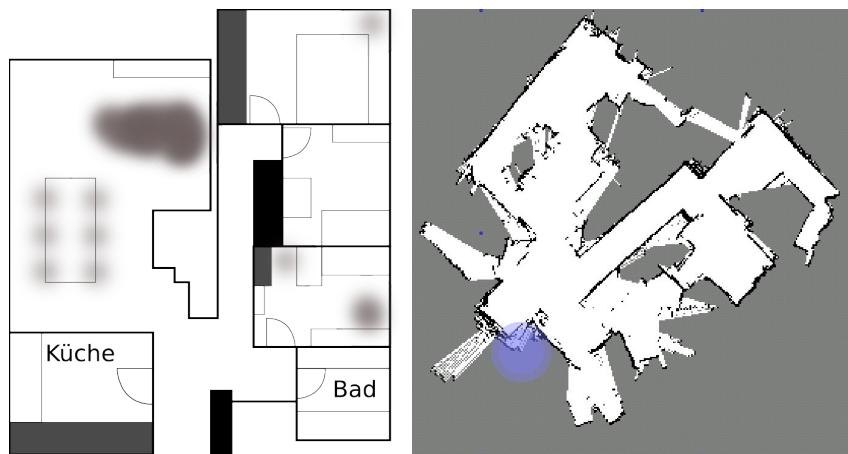


Abbildung 6.3.: Grundriss mit Innenausstattung einer Wohnung  
Küche und Bad wurden wegen zu hohen Absätzen nicht befahren. Wohnzimmer und Schlafzimmer weisen aufgrund der Möbel Lücken auf.

---

<sup>1</sup>Die Erfassung der zweiten Wohnung lieferte ähnliche Resultate und wird deshalb nicht mehr aufgeführt.

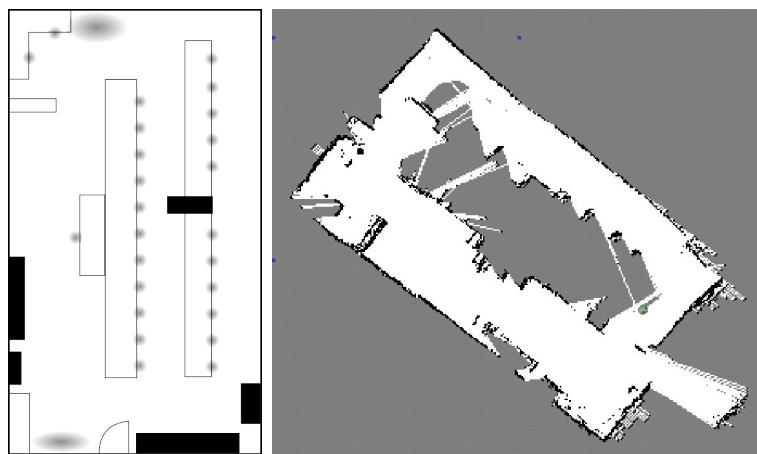


Abbildung 6.4.: Grundriss mit Innenausstattung eines Schulzimmers

### 6.3. Ergebnisse

Die besten Resultate wurden mit der Zellenstrategie erzielt. Der Winkelfehler der Positionsmessung wird kompensiert, indem die Karte aufgrund von Merkmalen verbunden werden kann. Fehlen diese Informationen ist die Geometrie meist verzogen (siehe Abbildung 6.5). In kleinen Räumen funktioniert die Aufnahme besser, weil das Tiefenbild mehr Anhaltspunkte aufweist. Bei Verbindungsabschluss, Durchdrehen der Räder (Fuge, Teppichrand), Stößen (Absatz, Aufprall) oder Überlastung des Rechners wird die Positionsbestimmung verfälscht und die Karte wird unbrauchbar (siehe Abbildung 6.6).

Die grössten Einschränkungen bei der Aufnahme:

- Aufgrund der kleinen Räder kann der Roboter Absätze nicht überwinden.
- Durch Möbel ist ein Grossteil des Raumes für die Kinect nicht sichtbar, was das Zusammenfügen von Raumteilen verunmöglicht.

Weitere Fehler entstehen aufgrund der Limitierungen des Infrarotsensors der Kinect [8]. Diese wurden im vorliegenden Fall verursacht durch Spiegelschränke (siehe Abbildung 6.7), Fenster, glänzende Objekte sowie Heizungslamellen (siehe Abbildung 6.8).

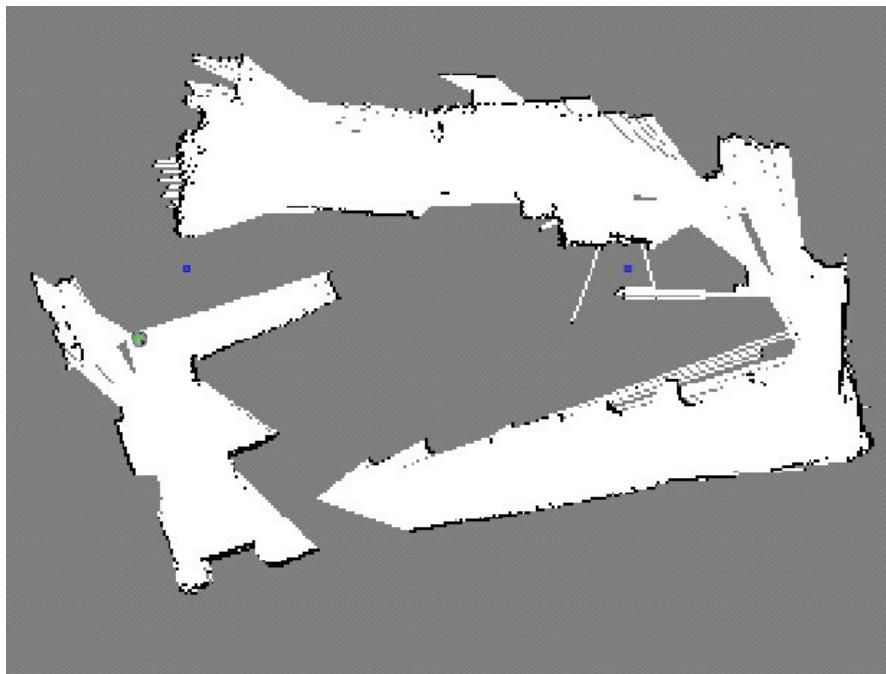


Abbildung 6.5.: Misslungenes Zusammenführen der Raumgeometrie

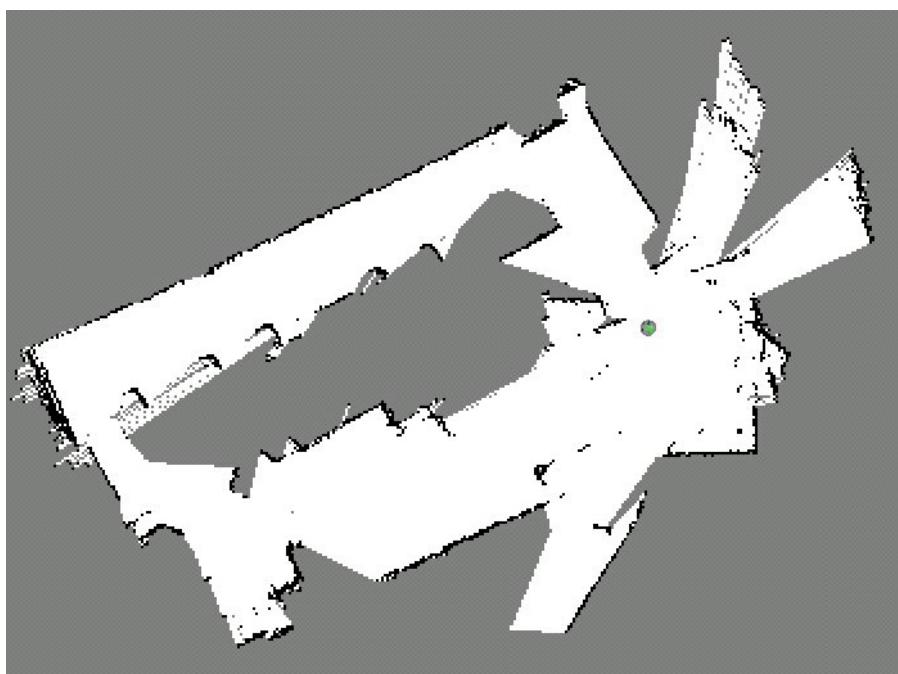


Abbildung 6.6.: Fehler nach Ausfall der Verbindung  
Die anfänglich gute Karte wird durch den Ausfall zerstört.



Abbildung 6.7.: Fiktiver Leerraum aufgrund von Spiegelungen (rot eingefärbt)

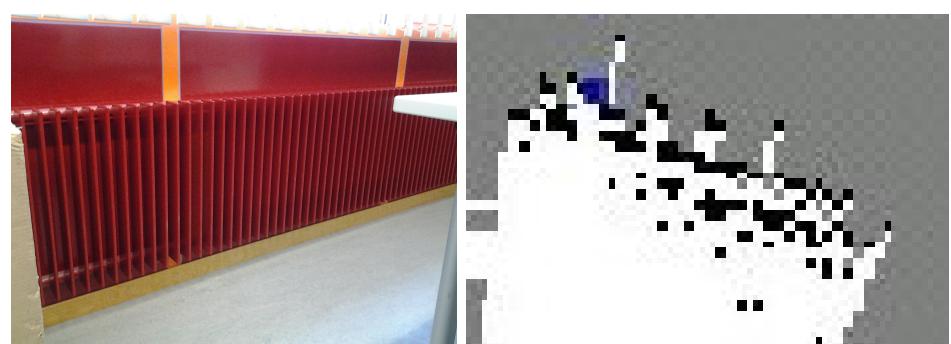


Abbildung 6.8.: Ungenaue Kante aufgrund von Heizungslamellen

## 6.4. Erweiterung 3D

Für die Innenraumerfassung ist weiter das dreidimensionale Modell von Bedeutung. Verschiedene Ansätze werden folgend erläutert.

### 6.4.1. 3D Photogrammetry

Im Rahmen einer Masterthesis am Institut Informatik des NTB wird zurzeit eine Software zur Erfassung von Gebäudehüllen entwickelt. Der eingesetzte Algorithmus verknüpft Merkmale von Bildern unterschiedlicher Perspektive um daraus die Kameraposition sowie die Position der Merkmale zu bestimmen. Theoretisch ist dieser Algorithmus auch für die Aufnahme von Innenräumen einsetzbar.

Der Test erfolgte mit den Bildern der Kinect als Eingangsdaten. Die Software ist in der Lage Kamerapositionen zu berechnen, aufgrund der kurzen Distanz zwischen den Bildern ist jedoch die Positionsbestimmung der Merkmale durch Triangulation ungenau (siehe Abbildung 6.9).

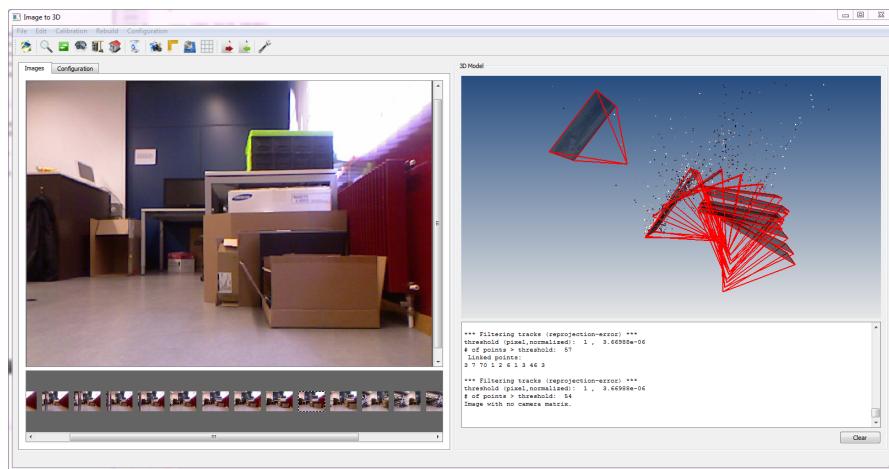


Abbildung 6.9.: Ausgabe der 3D Photogrammetrie Software

### 6.4.2. SLAM and Octomap

Mit Octrees<sup>2</sup> können 3D Modelle effizient gespeichert werden. ROS besitzt eine Anbindung an die Octomap Bibliothek<sup>3</sup>. Empfangene Punktewolken werden als

<sup>2</sup><http://de.wikipedia.org/wiki/Octree>

<sup>3</sup><http://octomap.github.com/>

besetzte Voxel in das 3D Modell eingetragen (siehe Abbildung 6.10). Als Basis für die Positionierung der Eingangsdaten dient eine per SLAM aufgenommene Karte. Erfolgt die 3D Modellierung gleichzeitig wie die Kartografierung, so enthält das 3D Modell Fehler aufgrund dynamischer Optimierungen der Karte. Diese Fehlerquelle kann durch die vorgängige Aufnahme der Karte vermindert werden (siehe Abbildung 6.11). Durch eine Kopplung der Punktwolken mit dem zugehörigen Kartenausschnitt könnten auch Änderungen der Karte ins 3D Modell propagiert werden.

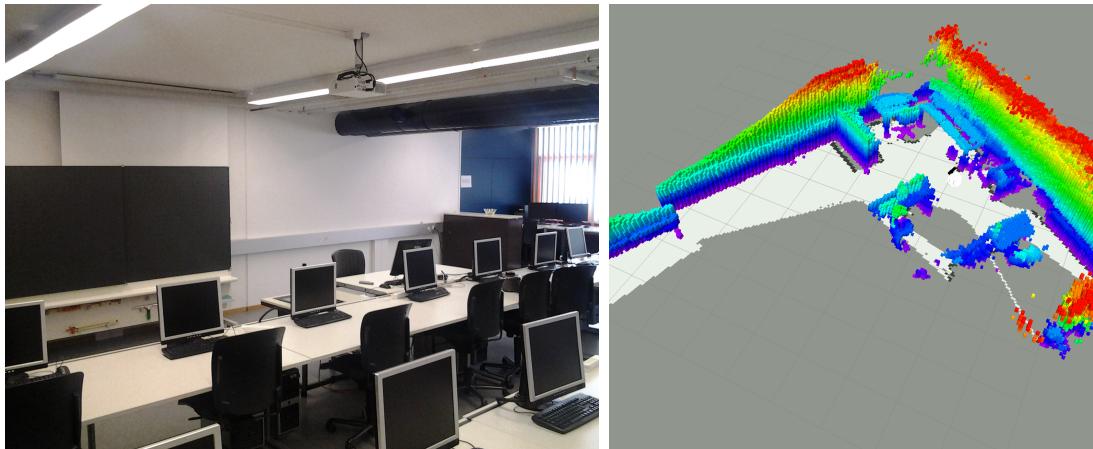


Abbildung 6.10.: Reale Szene im Vergleich zur Octomap bei Stillstand des Roboters mit einer Auflösung von 5 cm.

### 6.4.3. RGBDSLAM

Während des ersten Vertiefungsprojekts [8] wurde RGBDSLAM der Universität Freiburg eingesetzt. Da im ROS-Forum einige Beiträge zur Verwendung auf Robotern zu finden ist, sollte auch dieser Algorithmus erprobt werden. Innert nützlicher Frist konnte jedoch mit der verfügbaren Infrastruktur (Rechner und Wireless-Netzwerk) keine Aufnahme erstellt werden<sup>4</sup>.

---

<sup>4</sup>Die Software war lauffähig, die Berechnung des 3D Modells schlug jedoch fehl.

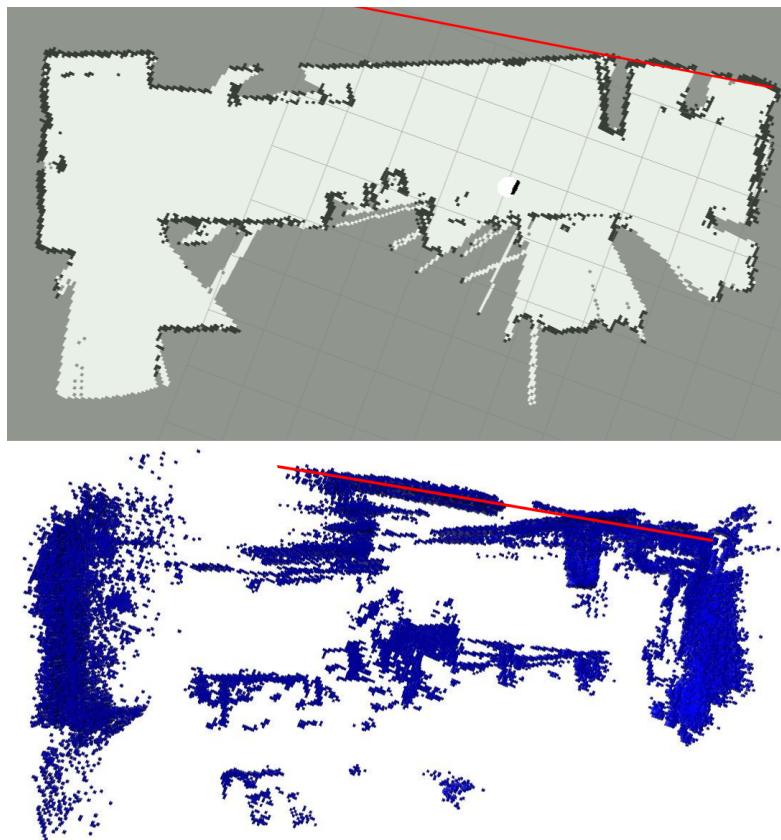


Abbildung 6.11.: Octomap mit bestehender Karte (Ansicht von oben)  
Die Octomap enthält Artefakte aufgrund eines Kartenverzugs in der rechten Ecke  
(Angezeigt durch eine rote Linie).

## 6.5. Fazit

Odometrie und Fahrwerk des Turtlebot sind für die Innenraumerfassung ungenügend. Entgegen den Erwartungen führt die ausschliessliche Verwendung der Odometrie nicht zu brauchbaren Ergebnissen. Die kommende Version des Roboters mit verbesserter Wegmessung führt womöglich zu präziseren Karten, scheitert jedoch ebenfalls an den anderen Fehlerquellen (siehe Abschnitt 6.3). Der Winkelfehler des Gyroskops pflanzt sich während der Messung fort und wird durch Erkennung optischer Merkmale kompensiert. Bei fehlenden Merkmalen ist jedoch die Odometrie massgebend. Ein Lösungsansatz wäre die Messung der absoluten Roboterausrichtung mit einem Magnetometer (Kompasssensor), um den Winkelfehler ohne Merkmale auszugleichen.

Keine der dreidimensionalen Erfassungsmethoden überzeugt. Für die 3D Photo-

grammetrie müsste die Aufnahmetechnik angepasst werden<sup>5</sup>, RGBDSLAM benötigt leistungsstärkere Hardware, Grafikkartenunterstützung und eine grosse Übertragungsbandbreite. Einzig der Einsatz von Octomap führte zu ansatzweise brauchbaren Ergebnissen. Hier fehlt vor allem die Kopplung zwischen 2D und 3D sowie eine bessere Odometrie.

---

<sup>5</sup>Grössere Bewegung der Kameraposition gegenüber den Bildmerkmalen

## 7. Schlussfolgerung

Die grosse Vorlaufzeit bei der Bestellung der Hardware hat sich bewährt. Trotz einer Lieferverzögerung von mehreren Monaten konnte die Arbeit im gewünschten Zeitraum durchgeführt werden. Aufgrund vieler Beispiele im Internet wurde angenommen, dass der Turtlebot mehrere Peripheriegeräte speisen kann. Durch Schilderung der Anforderungen hätte der Lieferant auf diese Probleme hindeuten und allenfalls eine Lösung für die Verwendung der Roboterarme anbieten können.

Trotz einiger Einarbeitungszeit erweist sich das Ökosystem rund um ROS als sehr ausgereift. Durch die lose Kopplung der Knoten können bestehende Funktionalitäten schnell verknüpft werden. So erfolgte zum Beispiel die Anbindung der Octomap-Bibliothek in weniger als einem halben Tag. Es existieren nützliche Werkzeuge zur Analyse des laufenden Systems sowie zur Handhabung der Pakete und zur Darstellung verschiedenster Daten der aktuellen Anwendung. Die schnelle Entwicklung der Komponenten führt teilweise zu Inkompatibilitäten. Vor allem der OpenNI-Stack zur Anbindung von Kamerasystemen (Kinect) änderte sich seit der letzten Version enorm. Weiter verwenden die Entwickler unterschiedliche Hardware (Roboter, Sensoren, Steuerrechner), weshalb die Funktionalität der Komponenten nicht auf allen Systemen gewährleistet ist. Aufgrund fehlender Erfahrung mit ROS wurde dieser Mehraufwand nicht einkalkuliert. Die Tutorials erweckten den Eindruck als sei die Turtlebot Plattform ausgereift und ab Stange lauffähig. Dank der guten Dokumentation und der schnellen Hilfe der Community konnten dennoch viele Probleme in nützlicher Frist behoben werden.

Entgegen den Erwartungen lieferte der Turtlebot kaum brauchbare Karten für die Innenraumerfassung. Die Fortpflanzung des Winkelfehlers von einigen Grad wurde unterschätzt. Nur mit viel Kalibrationsaufwand und Erfahrung in der Aufnahme gelang die Kartografierung. Durch Studium des verwendeten SLAM Verfahrens konnten die Einstellungen verbessert und die Genauigkeit von Objekten (Streuung der Hindernislinien) erhöht werden. Weiter ist dadurch nachvollziehbar, weshalb die Erfassung in kleineren Räumen besser gelingt. Möchte die Innenraumerfassung mit einem mobilen Roboter weiter verfolgt werden, so wäre wohl ein umgekehrter Ansatz erfolgsversprechend. Anstatt ein Minimalsystem zu optimieren, könnte ein lauffähiges, überdimensioniertes System [1] auf das Notwendigste reduziert werden.

## 7.1. Interessante Quellen

**The 3D Toolkit** In Zusammenarbeit der Universität Osnabrück und Bremen entstand ein Softwarepacket, welches in der Lage ist, dreidimensionale Punktwolken zu Modellen zu verarbeiten und vor allem durch hochauflösenden Beispielbilder besticht (siehe Abbildung 7.1).

<http://slam6d.sourceforge.net/>

**Kinect on a Quadrocopter** Die Mitglieder des Projektes *Visual Odometry for GPS-Denied Flight and mapping using a Kinect* entwickelten unter der Leitung des Massachusetts Institute of Technology (MIT) einen Quadrokopter der mit Hilfe der Kinect in der Lage ist dreidimensionale Odometriedaten zu berechnen. Darauf aufbauend konnte mit RGBDSLAM ein Modell der besuchten Umgebung erstellt werden (siehe Abbildung 7.2) [2].

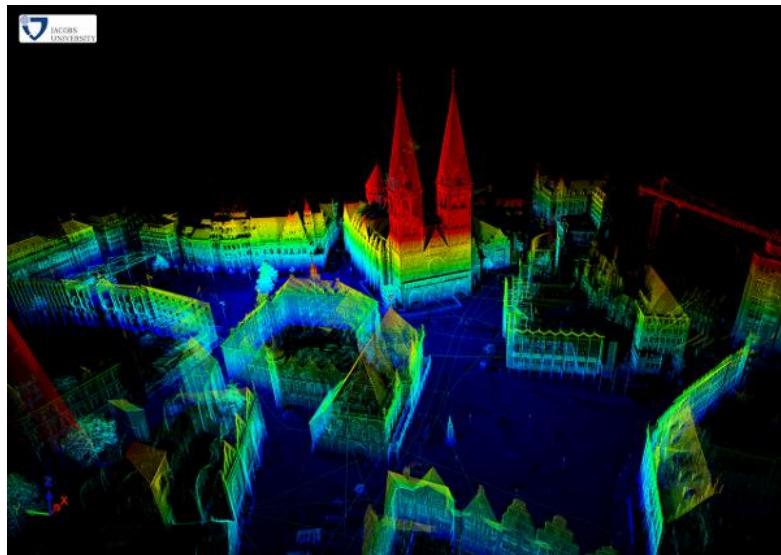


Abbildung 7.1.: Einleitungsbild des 3D Toolkit

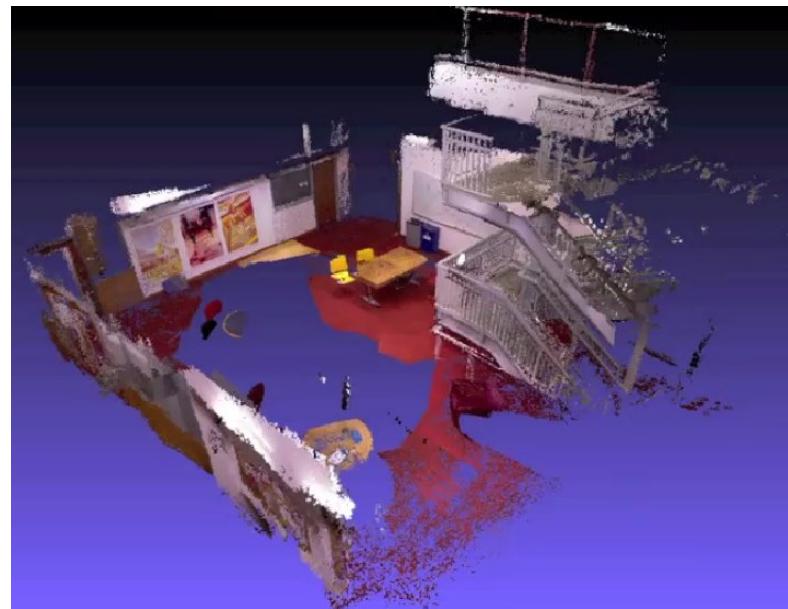


Abbildung 7.2.: RGBDSLAM mittels eines Quadrokopters

# Literaturverzeichnis

- [1] ANDERSON, S. ; MACTAVISH, K.: *3D visual SLAM with mobile robots.* <http://www.ros.org/news/2011/03/3d-visual-slam-with-mobile-robots.html>. – [Online; Stand 15. Januar 2013]
- [2] BACHRACH, Abraham ; PRENTICE, Samuel ; HE, Ruijie ; HENRY, Peter ; HUANG, Albert S. ; KRAININ, Michael ; MATURANA, Daniel ; FOX, Dieter ; ROY, Nicholas: Estimation, planning, and mapping for autonomous flight using an RGB-D camera in GPS-denied environments. In: *International Journal of Robotic Research* 31 (2012), September, Nr. 11, 1320–1343. <http://dx.doi.org/10.1177/0278364912455256>. – DOI 10.1177/0278364912455256. – ISSN 0278-3649
- [3] GRISETTI, G. ; STACHNISS, C. ; BURGARD, W.: Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. In: *IEEE Transactions on Robotics* 23 (2007), February, Nr. 1, S. 34–46. <http://dx.doi.org/10.1109/TRO.2006.889486>. – DOI 10.1109/TRO.2006.889486. – ISSN 1552-3098
- [4] ROS.ORG: *ROS - Robot Operating System.* <http://www.ros.org/wiki/de>. – [Online; Stand 15. Januar 2013]
- [5] ROS.ORG: *Turtlebot.* <http://ros.org/wiki/Robots/TurtleBot>. – [Online; Stand 15. Januar 2013]
- [6] ROS.ORG: *Turtlebot Arm.* [http://www.ros.org/wiki/turtlebot\\_arm](http://www.ros.org/wiki/turtlebot_arm). – [Online; Stand 15. Januar 2013]
- [7] ROS.ORG: *Turtlebot Calibration.* [http://www.ros.org/wiki/turtlebot\\_calibration/Tutorials/Calibrate%20odomentry%20and%20Gyro](http://www.ros.org/wiki/turtlebot_calibration/Tutorials/Calibrate%20odomentry%20and%20Gyro). – [Online; Stand 15. Januar 2013]
- [8] SACCIOLTO, Fabian: *Ersteinführung der Kinect am Institut INF*. Interstaatliche Hochschule für Technik NTB Buchs, September 2012
- [9] WIKIPEDIA: *Richtlinie 2002/95/EG (RoHS)* — Wikipedia, Die freie Enzyklopädie. [http://de.wikipedia.org/w/index.php?title=Richtlinie\\_2002/95/EG\\_\(RoHS\)&oldid=112123350](http://de.wikipedia.org/w/index.php?title=Richtlinie_2002/95/EG_(RoHS)&oldid=112123350). Version: 2012. – [Online; Stand 16. Januar 2013]

- [10] WIKIPEDIA: *Simultaneous Localization and Mapping* — Wikipedia, Die freie Enzyklopädie. [http://de.wikipedia.org/w/index.php?title=Simultaneous\\_Localization\\_and\\_Mapping&oldid=111593125](http://de.wikipedia.org/w/index.php?title=Simultaneous_Localization_and_Mapping&oldid=111593125). Version: 2012. – [Online; Stand 15. Januar 2013]

# A. Inbetriebnahme des Roboterarms

Die Installation und Inbetriebnahme der Roboterarme erfolgte hauptsächlich mit Hilfe des ROS Tutorials [6].

## A.1. Arduino IDE

Für die Konfiguration des Arbotix Controllerboards wird die Arduino Entwicklungsumgebung eingesetzt <http://arduino.cc> (Version 1.03). Die Erweiterungen für Arbotix sind unter [www.code.google.com/p/arbotix](http://www.code.google.com/p/arbotix) zu finden, die Inhalte des Arbotix Unterordners müssen in den Sketchbook Order von Arduino entpackt werden<sup>1</sup>.

Mit Arduino kann anschliessen das Pypose Sketchbook geöffnet werden. Nach Auswahl von angeschlossener Hardware (Arbotix), Seriellport und Download<sup>2</sup> können die Servo-ID's über Tools > Terminal angepasst werden. Jede ID am Bus muss dabei eindeutig sein. Wenn nötig werden die Servos einzeln angeschlossen. Die ID's des Armes sind dabei vorgegeben, diejenigen von der Dreh- und Kippvorrichtung wurden mit 6 (Drehen) und 7 (Kippen) konfiguriert.

Mit dem Pose Editor können die Ansteuerungslimiten ausgelesen sowie Stellungen des Armes abgespeichert werden.

Für die Ansteuerung via ROS ist das gleichnamige Sketchbook auf den Controller zu laden. Die Makros *USE\_BASE* sowie *USE\_HW\_SERVOS* müssen für den Turtlebot auskommentiert werden.

---

<sup>1</sup>Der Hardware Ordner muss direkter Unterordner des Sketchbooks sein, ansonsten wird die Arbotix-Plattform im Downloadmenü nicht angezeigt.

<sup>2</sup>Bei Fehler *Programmer is not responding* nützt es das Projekt nach dem Kompilieren zu speichern.

# B. Installation Entwicklungsrechner

## B.1. Wireless unter Linux

Für die Verbindung mit dem eduroam Netzwerk wurde vom Autor eine Anleitung im NTB-Intranet erstellt. In Systemen mit einfacherem Netzwerksetup könnte die manuelle Konfiguration von Nutzen sein<sup>1</sup>.

## B.2. Turtlebot ROS

Die Installation von ROS und die Inbetriebnahme des Turtlebot ist in [5] beschrieben. Dabei sind einige Spezialfälle zu beachten.

### B.2.1. SSH

Für die Verbindung zwischen Steuerrechner und Entwicklungssystem sollte ein Secure-Shell Server (SSH) auf dem Steuerrechner installiert werden.

### B.2.2. Chrony

Zur besseren Zeitsynchronisierung wird der Chrony-Dienst eingesetzt. Standardmäßig arbeitet dieser mit den Servern von Debian, welche hauptsächlich in Amerika stationiert sind. Unter */etc/chrony.conf* sind deshalb die Debian Server mit folgenden Einträgen auszutauschen.

```
server 0.ch.pool.ntp.org
server 1.ch.pool.ntp.org
server 2.ch.pool.ntp.org
server 3.ch.pool.ntp.org
```

---

<sup>1</sup>[Tutorial: Advanced Networking Setup](#)

Des Weiteren ist für das Update der ETH-Timeserver zu verwenden.

```
sudo ntpdate swisstime.ethz.ch
```

### B.2.3. Overlay

Um mit den neusten Entwicklungsquellen zu arbeiten besteht die Möglichkeit ein Overlay zu erzeugen. In ROS Fuerte sind dafür verschiedene Werkzeuge vorhanden (siehe <http://www.ros.org/wiki/fuerte/Installation/Overlays>).

Unter ROS Electric müssen dazu Umgebungsvariablen angepasst werden (Vorzugsweise in *.bashrc*).

```
source /opt/ros/fuerte/setup.bash
export ROS_PACKAGE_PATH=<Overlay-Verzeichnis>:$ROS_PACKAGE_PATH
export ROS_WORKSPACE=<Overlay-Verzeichnis>
```

Zuweilen funktioniert die Navigation zu einem gewünschten ROS-Stack trotz korrekt konfigurierten Umgebungsvariablen nicht. Möglicherweise ist der Stack Cache ungültig. Dies kann mittels der folgenden Kommandos behoben werden.

```
rospack profile
rosstack profile
```

### B.2.4. Udev Regeln

Werden mehrere USB-Geräte am Rechner angeschlossen, ist die Identifikationsnummer abhängig von der Anschlussreihenfolge. Um im Programm auf definierte Schnittstellen zuzugreifen, werden deshalb udev-Regeln (Userspace device) benötigt. Mit Hilfe von *udevadm* können die genauen Angaben der angeschlossenen Geräte inspiziert werden.

Für das verwendete System wurde die Datei */etc/udev/rules.d/53-usb.rules* mit folgendem Inhalt erstellt:

```
KERNEL=="ttyUSB[0-9]*", SUBSYSTEMS=="usb", ATTRS{interface}=="UC232R", NAME="ttyUSBO", SYMLINK="usb_roomba"
KERNEL=="ttyUSB[0-9]*", SUBSYSTEMS=="usb", ATTRS{interface}=="FT232R USB UART", NAME="ttyUSB1", SYMLINK="usb_arbotix"
```

## B.2.5. Batteriekennung

Die ursprüngliche Version des Turtlebot verwendete einen anderen Steuerungsrechner. Die Batterie ist jedoch nicht unter derselben Kennung (siehe Abschnitt B.2.4) ansprechbar und liefert die Werte in anderen Einheiten. Auf der Statusanzeige des Turtlebot erscheint deshalb eine Fehlermeldung. Unter *ROS Answers: Unable to check laptop battery state* ist eine Anleitung zur Lösung zu finden.

## B.3. Anpassung Robotermodell

Für die Berechnung der Koordinatentransformationen wird das Robotermodell im *Unified Robot Description Model* (URDF) abgelegt. Darin werden Verbindungen, Massen und Darstellungs- und Kollisionsvolumen der Einzelteile in einer Baumstruktur modelliert. Die aktuellen Transformationen sind über die /tf Topic verfügbar. Somit können Visualisierungen das Robotermodell entsprechend darstellen oder simulieren.

Die URDF Daten werden beim Start der Knoten via die Xacro Scriptsprache dynamisch erzeugt. Durch den modularen Aufbau sind so Änderungen schnell umzusetzen.

Die Xacro Skripte für den Turtlebot befinden sich unter  
*turtlebot/turtlebot\_description/urdf*.

## B.4. Kalibrierung

Das verwendete System enthält ein Gyroskop mit einer Auflösung von  $250^{\circ}\text{s}^{-1}$ . Dies kann über folgenden Eintrag vorgenommen werden:

*/etc/ros/<ROS-Version>/turtlebot.launch*

```
<param name="turtlebot_node/gyro_measurement_range" value="250"/>
```

Diese Werte ergaben gute Ergebnisse:

```
gyro_scale_correction = 1.6319
odom_angular_scale_correction = 1.037813
```

Laut Hersteller meldet das Gyroskop einen Fehlalarm:  
*Gyro sensor : Bad Gyro Calibration Offset*

Weitere Informationen zur Kalibrierung der Odometrie und zur Erfassung von Karten

*Tutorial: Navigation Tuning Guide*

*ROS-Answers: Trouble getting an accurate map*

Die Kamera verwendet ohne Kalibrierung Standardparameter, welche bereits zu guten Ergebnissen führen. Informationen zur Kalibrierung der Kamera befinden sich im ROS-Wiki

*Tutorial: Intrinsic Calibration*

Für das Tiefenbild wird die Topic `/camera/ir` verwendet. Die Kalibrierung ist auflösungsabhängig.

## B.5. OpenNI

Während der Arbeit änderten sich die OpenNI Treiber stark. Die Registrierung von Tiefen- und Farbinformation funktionierte in der zuletzt verwendeten Version nicht. Weiter werden viele unnötige Dienste gestartet (Komprimierte Formate, Infrarotbilder) und der Treiber hängt sich von Zeit zu Zeit auf. Die Einbindung der neuen Bibliothek<sup>2</sup> erfolgt durch Verwendung des neuen Launch-Files (`openni_launch openni.launch`) und Einsatz einer dieser Topics:

`/camera/depth/points`

`/camera/depth_registered/points`

## B.6. Extraktion von Bilddaten

Für den 3D Photogrammetry Algorithmus des Instituts INF wurde der Datenstrom der Kinect mit `rosbag` aufgenommen und anhand dieses Artikels extrahiert:

*Tutorial: Exporting image and video data*

## B.7. Android Remote

Die Einrichtung der Fernsteuerung über ein Android Device<sup>3</sup> erfolgt mit Hilfe des ROS App Choosers.

<http://www.ros.org/wiki/turtlebot/Tutorials/AndroidControl>

---

<sup>2</sup>*Migration Guide*

<sup>3</sup>Für den Test wurde ein Samsung Galaxy Tab 10.2 verwendet.

Die Ablage der Karten in einer MongoDB-Datenbank benötigte zusätzliche Anpassungen an den Launch-Files sowie den map-store Stack. Die Anpassungen erfolgten aufgrund der Beispieldateien der verwendeten Stacks. Werden die Launch-Files von einem Entwicklungsrechner gestartet, so kann deren korrekte Funktion vorab überprüft werden.