Telematics
Computer Systems

Freie Universität Berlin

Computer Systems and Telematics — Distributed, Embedded Systems

Bachelorarbeit

Design und Implementierung eines mobilen Referenzsystems für die Indoorlokalisierung

Benjamin Aschenbrenner

Matr. 4292264

Simon Schmitt

Matr. 4287788

Betreuer: Prof. Dr. rer. nat. Mesut Güneş

Betreuender Assistent: Dipl.-Inf. Heiko Will

Institut für Informatik, Freie Universität Berlin, Deutschland

24. Juli 2011

Text so ok, oder lieber zwei Texte? Unterschreibt so auch jeder dass der andere nicht abgeschrieben hat? Wir versichern, dass wir die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, sind als solche gekennzeichnet. Die Zeichnungen oder Abbildungen sind von uns selbst erstellt worden oder mit entsprechenden Quellennachweisen versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner Prüfungsbehörde eingereicht worden.

Berlin, den 24. Juli 2011

(Benjamin Aschenbrenner) (Simon Schmitt)

Zusammenfassung

Zusammenfassung

Das Forschungsprojekt Indoorlokalisierung der Arbeitsgruppe Computer Systems and Telematics beschäftigt sich mit der Positionsbestimmung in GPS-freien Umgebungen anhand von Signallaufzeiten zwischen mobilen Sensorknoten. Der Schwerpunkt dieser Bachelorarbeit liegt darin, ein möglichst genaues Testsystem für die Indoorlokalisierung anhand von visuellen Daten zu entwickeln. Dabei werden Microsoft Kinects eingesetzt, durch die es möglich wird, Räume dreidimensional zu erfassen.

Abstract

The indoor localization research project of the computer systems and telematics departement works on positioning determination in gps-less areas using signal propagation delays between mobile sensory nodes. The main focus of this bachelor thesis is to develop an accurate testing system for indoor localization based on visual data. In doing so, Microsoft Kinects are used, which enable three dimensional room capturing.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Glossar

Coordinate Frame Ein Bezugssystem, repräsentiert durch einen String. xv–xvii, 7, 8

Fixed Frame Ein Coordinate Frame, welcher in rviz als globales Bezugssystem gewählt wird. 7

Interface Definition Language Eine Sprache, die für den Austausch von Informationen zwischen zwei Programmen sorgt, wobei beide Programme diese Sprache beherrschen müssen. xvii, 6

Kinect Ein Gerät der Firma Microsoft, das über eine USB-Schnittstelle 2D und 3D Informationen aus einem gewissen Sichtbereich zur Verfügung stellt. 8, 9

Launch Datei Eine XML-Datei in der festgelegt wird, welche Launch Dateien (rekursiv), Stacks oder Packages mit einer konkreten Konfiguration gestartet werden sollen. 6, 7

Manifest Eine XML-Datei, die ein Package oder Stack beschreibt und Abhängigkeiten offenlegt. xv, xvi, 6, 7

Message Eine strikt getypte Datenstruktur, die primitive Datentypen, geschachtelte Messages und Arrays dieser beiden erlaubt [1, S. 3]. xv, xvi, 6–8

Node Ein Softwaremodul oder Prozess, der zur Laufzeit mit anderen Nodes über Messages innerhalb des Robot Operating Systems kommuniziert [1, S. 3]. xv, xvi, 6–8

Odometrie Daten, die aufgrund der Aktivierung von Antriebssystemen mit einer bestimmten Wahrscheinlichkeit die Bewegungsrichtung und -strecke wiederspiegeln. 2, 3

Package Eine Ordnerstruktur, die ein oder mehrere Nodes einschließt und ein Manifest besitzt [1, S. 4]. xv, xvi, 6–8

Pointcloud Eine Datenstruktur, die u.a. (x,y,z)-Punkte speichert. 8, 9

Robot Operating System Ein Framework für Roboter das Hardwareabstraktion, eine Paketverwaltung, ein Nachrichtensystem und die Möglichkeit zum Betreiben auf mehreren Computern liefert. xv, xvii, 5

rviz Ein Node, der Messages visualisieren kann. xv, xvi, 7, 9

Service Ein synchrone Informationsquelle eines Nodes, repräsentiert durch einen Namen, einer Message als Übergabeparameter, und einer Message als Rückgabewert [1, S. 3]. 6, 8

Stack Eine Ordnerstruktur, die üblicherweise mehrere Packages einschließt und ein Manifest besitzt [1, S. 5]. xv, 6

Target Frame Ein Coordinate Frame, welcher in rviz als Kameraperspektive eingestellt wird. 7

Topic Eine asynchrone Informationsquelle und -senke welche durch einen String repräsentiert wird, über die Nodes Messages senden oder abonnieren können [1, S. 3]. 6–8

Akronyme

Frame Coordinate Frame. 7

GPS Global Positioning System. 2

IDL Interface Definition Language. 6

ROS Robot Operating System. 5–8

KAPITEL 1

Einleitung

1.1 Motivation

Systeme zur Positionsbestimmung werden für zahlreiche Zwecke genutzt und deren Bedeutung wächst parallel zur Verbreitung immer neuer sogenannter *location based services* und deren wachsender Nutzung. Für Anwendungen im Freien haben sich Satelliten gestützte Systeme, welche hohe Genauigkeit bieten, etabliert. Als bekanntes Beispiel sei hier das *NAVSTAR-GPS* genannt, welches sich auch im zivil nutzen lässt. Allerdings ergeben sich viele Anwendungsumgebungen, in denen derartige Systeme gar nicht, bzw. nur ungenau funktionieren oder bewusst aus Kostengründen gemieden werden. Dies sind typischerweise Umgebungen in denen die Satellitensignale zu stark gedämpft werden oder vor allem durch Reflexionen bedingte Laufzeitverschiebungen, sich negativ auf die Genauigkeit auswirken, wie z.B.:

- innerhalb von Gebäuden ("indoor")
- im Untergrund (Tunnel, Höhlen u.ä.)
- im Bereich dicht bebauter urbaner Gebiete (Mehrwegeausbreitung)

Um in solchen Umgebungen dennoch Lokalisierung zu ermöglichen wurden und werden viele theoretische Konzepte und konkrete Systeme entwickelt. Einen Überblick hierzu bietet Quelle anbringen (mobile entity localization and tracking in GPS less enviroments - Buch) Quelle anbringen (mobile entity localization and tracking in GPS less enviroments - Buch) Auch in der Arbeitsgruppe *Computer Systems & Telematics*, an der *FU-Berlin*, wurde dem Problem der indoor Lokalisierung mit der Entwicklung eines *Wireless Sensor Network (WSN)* basiertem Systems im Rahmen des Forschungsprojektes *FeuerWhere*, begegnet. Dieses Projekt entstand u.a. in Kooperation mit der Berliner Feuerwehr. ist das wichtig zu wissen an dieser Stelle? Ziel bei der Entwicklung war ein flexibles indoor Lokalisierungssystem zu schaffen, welches mit low-cost Komponenten bzw. ohne Spezialhardware konstruiert wurde. Im Kern ist das System in der Lage die Entfernung zwischen involvierten Sensorknoten zu bestimmen und dadurch Rückschlüsse auf deren Position zu ermöglichen. In dem WSN unterscheidet man zwei Arten von Knoten, mobile Knoten und Anker Knoten. Diese unterscheiden sich nur dadurch, dass die Position eines Anker Knotens bekannt ist. Bei einer hinreichenden

Zahl von Anker Knoten im WSN kann dann per Trilateration bzw. Multilateration die Position eines mobilen Knotens ermittelt werden. add figure principle of trilateration ? Die Entfernungsmessung zwischen zwei Knoten geschieht hierbei durch Laufzeitmessungen von per Funk gesendeten Round Trip Time (RTT) Paketen, wodurch eine teure sowie aufwendige Zeit-Synchronisierung zwischen den Knoten entfällt, da bei der Messung der RTT nur ein Knoten die Zeit berechnet. Diese Laufzeitmessungen sind jedoch durch in der Hardware auftretenden Jitter und in *non-line of sight (NLOS)* Umgebungen auftretende Mehrwegeausbreitung fehlerbehaftet. Die genaue Funktionsweise und Untersuchung der Auftretenden Fehler ist beschrieben in. hier würde ich natürlich gerne Heiko's paper reffen. Frage: Ist das schon erlaubt? Um diesen Fehler zu untersuchen, ist es sehr nützlich, ein möglichst genaues aber ebenso flexibles Testsystem Referenzsystem? zur Verfügung zu haben, welches mögliche Anpassungen, Konfigurationen und Einsatzszenarien des indoor Lokalisierungssystems, in Hinblick auf dessen Genauigkeit, evaluierbar macht. Der Implementierung und Analyse eines solchen Referenzsystems widmet sich diese Arbeit.

1.2 Lösungsansätze

Es gibt zahlreiche Möglichkeiten der Lokalisierung. Damit die Entfernungsmessung der Sensorknoten evaluiert werden kann, werden unabhängige Daten benötigt. Das heißt, dass nicht auf das bereits bestehende System zurückgegriffen werden kann. Im Folgenden werden einige andere Möglichkeiten beschrieben und ihre Vor- sowie Nachteile bezüglich der Anwendung in einer Referenzimplementierung erörtert.

Die einfachste Möglichkeit wäre, die Positionsbestimmung manuell durchzuführen. Dabei könnte eine Testmessung auf einer zuvor festgelegten Strecke durchgeführt werden. Eine Messung der Strecke per Hand würde zwar verlässliche tatsächliche Positionsdaten liefern, jedoch wäre der zu betreibende Aufwand für viele verschiedene Strecken enorm. Um eine robuste Evaluierung realisieren zu können, ist es jedoch gerade von Nöten, Messungen auf verschiedenen Strecken umzusetzen. Theoretisch denkbar wäre eine Verwendung des Global Positioning Systems (GPSs), das eine dynamische Positionsbestimmung erlaubt. Jedoch wäre man dann auf das GPSs Signal angewiesen, und könnte keine Ortung in Gebäuden durchführen. Nötig wären Versuchsaufbauten im Freien. Nur könnten hier nicht die gleichen Bedingunen erreicht werden, die tatsächlich innerhalb eines geschlossenen Gebäudes herrschen. Durchaus von Interesse wären dagegen Odometriedaten. Odometrie wird von einem fahrenden Objekt von dessen Antriebssystem geliefert. Dieses kann mit einer gewissen Wahrscheinlichkeit messen, dass das Objekt aufgrund der Aktivierung gewisser Vortriebsmechanismen eine bestimmte Wegstrecke zurückgelegt hat. Ebenfalls bestimmbar muss die Richtung sein, in die die Bewegung ausgeführt wurde. Allerdings kann diese Methode nicht absolut fehlerfrei sein. Dadurch summieren sich kleinste Fehler mit der Zeit so weit auf, dass eine Positionsbestimmung unmöglich wird. Wird jedoch diese Art der Lokalisierung mit Anderen kombiniert, erhält man sehr wohl eine gute probabilistische Position. Eine weitere Möglichkeit wäre eine ständige Positionsbestimmung anhand von Kameras. Dabei können Bewegungen zwischen zwei aufgenommenen Bildern erfasst werden. Handelt es sich um dreidimensionale Bildinformationen, können schon heute relativ robuste Aussagen über eine zurückgelegte Wegstrecke getroffen werden. Allerdings handelt es sich hierbei um ein aktuelles Forschungsgebiet, in dem es oft noch keine optimalen Lösungen für dabei auf-

tretende Probleme gibt. Diese Art der Lokalisierung entspricht sinnhaft in etwa der der Odometriedaten. Es handelt sich hier um visuelle Odometrie. Deshalb können sich auch hier theoretisch Fehler aufaddieren. Dennoch kann diese Methode ungleich genauer sein, da sie lediglich von der Güte der Daten und den verwendeten Algorithmen abhängt. So kann ein solches System beispielsweise auf eine zurückgelegte Wegstrecke zurückblicken, und anhand des neuen Blickwinkels auf bereits gesammelte Daten eben diese optimieren, um bessere Aussagen über den gerade zurückgelegten Weg treffen zu können.

Leider handelt es sich bei den genannten Möglichkeiten entweder um Daten die für sich betrachtet zu ungenau erscheinen, oder um komplexe Algorithmen, die viel Rechen- und Speicheraufwand erfordern. Deshalb sind unweigerlich bewährte Kombinationen obiger Verfahren am besten geeignet, um eine möglichst robuste Lokalisierung im Gebäude durchführen zu können.

1.3 Aufgabenstellung

Die Aufgabe des Referenzsystems besteht darin, möglichst genaue Positionsinformationen relativ zur Startposition und -laufrichtung zu liefern. Diese Daten müssen zuverlässiger, als die gegenwärtige Konfiguration der Sensorknoten sein, sodass eine Optimierung der Entfernungsmessung dieser stattfinden kann.

Die Daten werden so erhoben, dass eine visuelle Auswertung sowohl während der Ausführung des Tests als auch später stattfinden kann. Dabei muss erkennbar sein, zu welcher Zeit welche Positionsdaten von welchem System vorlagen.

Als Basis wird ein Roboter dienen, der durch das Gebäude fährt. Die Steuerung erfolgt zunächst per Tastatur, wobei alternative Steuerungsmöglichkeiten getestet werden sollen. In diesem Kontext sollte der Roboter möglichst autonom durch Flure navigieren und eine gefahrene Strecke bei Bedarf wiederholen können.

Damit die Daten zunächst visuell ausgewertet werden können, muss eine Karte der Teststrecke erstellt werden, die es theoretisch erlaubt, sowohl die Positionsdaten der Sensorknoten, als auch die des Referenzsystems anzuzeigen.

Aufbau der Arbeit nicht doch ans Ende von 1.1 übernehmen? Die Arbeit ist wie folgt aufgebaut: In Kapitel 2 wird zunächst das generelle Konzept, sowie die verwendete Soft- und Hardware erläutert. Anschließend wird im Detail auf die Implementierungen dieser Arbeit wie auch auf verwendete Algorithmen eingegangen. Diese wird in Kapitel 3 anhand mehrerer Testläufe analysiert. Im Kapitel 4 wird abschließend sowohl ein Fazit der Arbeit als auch ein Ausblick gegeben.

KAPITEL 2

Umsetzung

2.1 Konzept

lorem ipsum!

2.2 Soft- und Hardware

Im Folgenden werden die wichtigsten Konzepte näher erläutert.

2.2.1 Microsoft Kinect

Auflösung, Reichweite, Genauigkeit in Abhängigkeit zur Entfernung, generelle Funktionsweise, schatten nicht vermeidbar(aussage wird referenziert) openni_kinect stack erwähnen/referenzieren! Überleitung zu ROS, wir brauchen ROS!

2.2.2 Robot Operating System

Robot Operating System (ROS) ist ein Open-Source Betriebssystem, welches viele Probleme und Aspekte verteilter, komplexer Software bezüglich der Anwendungsentwicklung für Roboter kapselt. Falls nicht anders gekennzeichnet, dient [1] in diesem Abschnitt als Quelle.

Im eigentlichen Sinne ist ROS kein Betriebssystem. Vielmehr handelt es sich um ein leichtgewichtiges C++ Framework, das in einem bestehenden Betriebssystem ausgeführt wird. Dabei verwendet es eine Peer-to-peer Kommunikationsarchitekur, mit Hilfe derer einzelne Programme in verschiedenen Programmiersprachen unabhängig voneinander kommunizieren können. ROS legt zudem großen Wert auf eine Tool-basierte Funktionsweise. Das heißt, dass alles strikt in Module gegliedert ist, wodurch das Framework selbst sowie die einzelnen Module besonders gut getestet werden können.

In dieser Arbeit dient ROS als eine robuste Architektur zum Transport verschiedener Daten zwischen Programmen, die bei Bedarf auch auf verschiedenen Computern im Netzwerk ausgeführt werden können. Wie in den Kapiteln 2.3 und 2.4 näher beschrieben, bietet ROS

bereits einen großen Funktionsumfang bezüglich des Umgangs mit Robotern. Im Folgenden wird eine kurze Einführung in die generelle Architektur sowie in in dieser Arbeit benötigte Tools gegeben.

Architektur

Einzelne Module beziehungsweise Programme in einer ROS Umgebung werden Nodes bezeichnet. Diese können beliebg oft unter Angabe eines Namensraumes gestartet werden. Das heißt ein Node kann generisch Aufgaben lösen, ohne zu wissen, ob er beispielsweise gerade den rechten oder linken Arm eines Roboters repräsentiert.

Damit ein Node unter ROS übersetzt beziehungsweise ausgeführt werden kann, muss er einem Package zugeordnet sein. Ein Package kann mehrere Nodes beinhalten. Es definiert durch eine Manifest Datei, welche Abhängigkeiten es zu anderen Packages besitzt. Mehrere Packages können zu einem Stack zusammengeschlossen sein, beispielsweise wenn sie eine gemeinsame Aufgabe erfüllen, indem sie diese in Teilaspekte zerlegen. Ein solcher Stack besitzt dann ebenfalls ein Manifest.

Innerhalb der Packages oder Stacks können Launch Dateien angelegt werden, die das Starten eines ganzen Systems mit mehreren Nodes erleichtern. Dabei kann in einer solchen Datei auch definiert sein, dass bestimmte Nodes nicht lokal, sondern entfernt auf einem anderen Computer gestartet werden. Außerdem können hier Parameter an die Nodes übergeben werden, die sich üblicherweise nur selten ändern. Launch Dateien können zudem andere Launch Dateien rekursiv inkludieren und dabei auch Parameter überschreiben(Inernet-)Quelle dafür angeben, da nicht in angegebener Quelle). Verzichtet man auf eine Launch Datei, dann müssen die Nodes einzeln gestartet und auch wieder beendet werden. Durch die Launch Datei können alle gestarteten Nodes über STRG+C zusammen beendet werden.

Nodes kommunizieren über das ROS interne Kommunikationsnetzwerk miteinander. Das heißt, wenn ein Node Informationen anderen Nodes zur Verfügung stellt, veröffentlicht er diese über ein bestimmtes Topic. Andere Nodes können dieses Topic abonnieren. Dabei haben Topics immer einen bestimmten Namen, der sowohl durch Nodes, die Informationen veröffentlichen, als auch von Nodes die Informationen abonnieren, die sie benötigen um ihre Funktionalität zu erfüllen, festgelegt sein kann. Um einen Konflikt zu vermeiden, muss in einem konkreten System in der Launch Datei eine Abbildung zwischen Topic Benennungen stattfinden. Nodes können zudem sehen, ob ihr eigenes Topic abonniert wurde. Dadurch lässt sich Rechenzeit sparen, sollte kein Abonnent vorhanden sein.

Informationen werden in Messages zu einem Topic veröffentlicht. Sie werden zunächst in einer platformunabhängigen Sprache definiert, einer Interface Definition Language (IDL). Diese Abstraktion erlaubt es, in wenigen Zeilen eine Message zu definieren. Entsprechende Code Generatoren in verschiedenen Sprachen erzeugen automatisch aus dieser Definition Klassen, welche durchaus hunderte Zeilen Code umfassen können. Dadurch fällt es dem Programmierer sehr viel einfacher, in ROS sprachenunabhängig neue zuvor nicht bekannte Messages zu definieren. Viele oft benötigte Messages werden beispielsweise bereits durch den *common_msgs* Stack zur Verfügung gestellt. Zusätzlich zur Topic Architektur gibt es in ROS Services, welche einen optionalen Parameter erhalten, und üblicherweise einen Rückgabewert liefern. Durch diese kann eine synchrone Kommunikation zwischen Nodes stattfinden.

Tools

Quellen finden!

Ein Package wird über das Tool *rosmake PACKAGE_NAME* kompiliert. Dabei werden falls nötig alle Abhängigkeiten die in der Manifest Datei eingetragen wurden ebenfalls rekursiv kompiliert. Ein Package mit mehreren Nodes besitzt somit mehrere ausführbare Dateien, die über *rosrun PACKAGE_NAME NODE_NAME* einzeln, über *roslaunch PACKAGE_NAME LAUNCH_DATEI_NAME* zusammen, gestartet werden.

Damit gestartete Nodes sich in ROS finden, gibt es einen Namensdienst, den sogenannten Master. Alle Nodes melden sich an diesem Dienst an, und erhalten hierüber Direktverbindungen zu anderen Nodes. Dieser Dienst wird entweder über die Konsole mit dem Befehl *roscore*, oder implizit durch Benutzung einer Launch Datei gestartet. Ein spezieller Node namens *rosout* wird immer gemeinsam mit dem Master gestartet. Er bekommt automatisch Log Nachrichten aller Nodes. Diese können an zentraler Stelle mit dem Tool *rxconsole* angezeigt werden.

Da eine Topographie mit vielen Nodes sehr unübersichtlich sein kann, bietet ROS ein Tool namens *rxgraph* an. Dieses Tool zeigt alle Nodes visuell als Knoten und verbindet diese, sollten sie Topics einander abonnieren.

Um einzelne Nodes oder ein ganzes System zu testen, kann es sinnvoll sein, bestimmte Daten beispielsweise einer Kamera oder eines Roboters, bei einem normalen Test aufzunehmen und anschließend so wieder abzuspielen, dass der eigene noch zu entwickelnde Node mit diesen Daten als Eingabe getestet werden kann. ROS stellt hierfür das Tool *rosbag* zur Verfügung. Mittels *rosbag record TOPIC_NAME(N) -O DATEI* werden ein oder mehrere Topics abonniert und in einer einzelnen Datei aufgezeichnet. Über *rosbag play DATEI(-EN)* werden eventuell mehrere solche Dateien gleichzeitig wieder abgespielt. Dabei werden die Zeitstempel beachtet, die in einer jeden aufgezeichneten Message enthalten sind. Die Reihenfolge, in der die Messages während des Aufzeichnens erstellt wurden, entspricht also auch der Abspielreihenfolge.

Messages transportieren üblicherweise Daten, die visualisiert werden können. ROS hält hierfür einen Node namens rviz bereit. Dieser versteht bereits einige Messages, kann aber auch bei Bedarf durch Plug-Ins erweitert werden. In dieser Arbeit wird rviz beispielsweise dafür eingesetzt, eine Karte und einen Roboter darin anzuzeigen. Hierbei wird deutlich, dass rviz die Transformation zwischen Karte und Roboter kennen muss. Dafür sorgt ein sehr wichtiges Package namens *tf*. Jede Message ist in einem bestimmten Koordinatensystem entstanden, einem sogenannten Coordinate Frame, welcher in einer Message durch einen String repräsentiert wird. Der *tf*-Node abonniert nun alle *tf-Messages*, die zu einem Topic namens */tf* beispielsweise von Sensoren veröffentlicht werden. Andere Nodes, die die Transformation zwischen zwei Coordinate Frames (Frames) benötigen, wie beispielsweise rviz zum Visualisieren der Messages, können nun anhand der gesammelten Daten des *tf-Nodes* diese direkt ablesen. Diese Transformationen werden in einer Baumstruktur vorgehalten. Es muss also zwischen zwei Frames immer genau eine Hierarchie geben. In rviz selbst, kann der Anwender einen globalen Fixed Frame wählen, in den alle sonstigen Frames unter Beachtung ihrer Transformation zu diesem gezeichnet werden, typischerweise also die Wurzel des *tf*-Baumes. Zusätzlich lässt sich ein Target Frame auswählen, den der Benutzer verfolgen möchte. Wird also eine Karte als Target Frame gewählt, bewegt sich der Roboter. Wird der Roboter selbst

Message	Topic	Beschreibung
sensor_msgs/CameraInfo	/camera/depth/camera_info	Parameter der Infrarot Kamera
stereo_msgs/DisparityImage	/camera/depth/disparity	was ist das, brauchen wir das?
sensor_msgs/Image	/camera/depth/image	Tiefenbild, enthält die Entfernungen in Metern
sensor_msgs/PointCloud2	/camera/depth/points	farblose Pointcloud
sensor_msgs/CameraInfo	/camera/rgb/camera_info	Paramter der RGB Kamera
sensor_msgs/Image	/camera/rgb/image_color	RGB Bild
sensor_msgs/Image	/camera/rgb/image_mono	Bild in Graustufen
sensor_msgs/PointCloud2	/camera/rgb/points	farbige Pointcloud

Tabelle 2.1: Übersicht der veröffentlichten Informationen von *openni_camera*.

ausgewählt, bewegt sich die Karte während der Roboter immer in der Mitte bleibt.

2.2.3 OpenNI_Kinect Stack

Der OpenNI_Kinect Stack integriert unter Anderem die in 2.2.1 beschriebene Kinect in ROS. Er erlaubt neben Skeleton Tracking und Gesture Recognition auch die direkte Verwendung der Rohdaten der Kinect durch das Package *openni_camera*. Wie in [2] beschrieben, werden durch das Starten des *openni_camera* Nodes u. a. die in der Tabelle **??** (steht hier immer noch ein doppeltes fragezeichen?) beschriebenen Messages zu den jeweiligen Topics veröffentlicht. Über jeweils einen Service *set_camera_info* erlaubt das Package zusätzlich eine Konfiguration und Kalibrierung der Infrarot und RGB Kamera.

Eine Message des Typs *PointCloud2* wird hierbei dazu verwendet, 3D Informationen zu veröffentlichen[1]. Diese Message besitzt wie in [3] näher beschrieben u.a. folgende Komponenten:

Header Sequenznummer, Zeitstempel und Informationen über den Coordinate Frame

Fields Liste aller Felder eines Punktes

Row_Step Länge einer Zeile

Data Liste aller Punkte

In Abbildung 2.1 sind zwei solche Pointclouds aus zwei verschiedenen Winkeln dargestellt. Auf der linken Seite ist nahezu die Sicht der Kinect zu sehen, während auf der rechten Seite der Sichtwinkel auf die Wolke stark geändert wurde. Beide Bilder wurden zu unterschiedlichen Zeiten aufgenommen, ohne die Position der Kinect zu verändern. Dadurch ist ein unterschiedliches Rauschen an den Rändern der Farbbereiche gut zu erkennen.

Farbbereiche-Entfernungen erklären

[1] siehe **??**

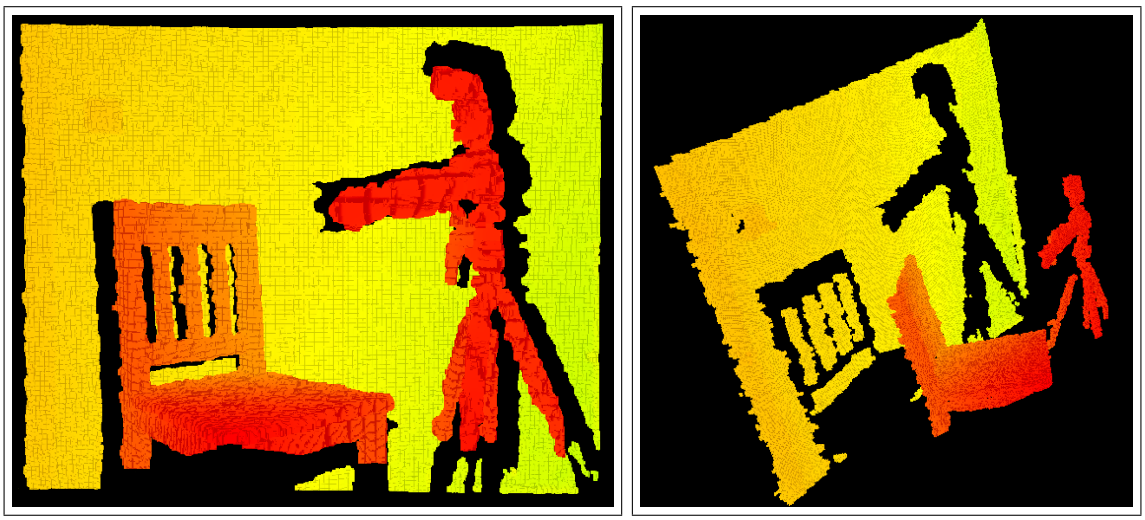

Abbildung 2.1: Zwei Pointclouds der Kinect, visualisiert durch rviz.

2.2.4 Turtle Bot

lorem ipsum!

2.3 Pathfinder

lorem ipsum!

2.4 NavStack

lorem ipsum!

KAPITEL 3

Analyse

3.1 Testlauf

lorem ipsum!

3.2 Genauigkeit

lorem ipsum!

3.3 Fazit

lorem ipsum!

KAPITEL 4

Ausblick

lorem ipsum!

KAPITEL 5

Anhang

lorem ipsum!

Literaturverzeichnis

[1] Morgan Quigley, Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system, 2009.

[2] Openni camera package. `http://www.ros.org/wiki/openni_camera`, July 2011.

[3] Pointcloud2 message. `http://www.ros.org/doc/api/sensor_msgs/html/msg/PointCloud2.html`, July 2011.