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

17. 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 17. 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

Interface Definition Language Eine Sprache, die für den Austausch von Informationen zwischen zwei Programmen sorgt, wobei beide Programme diese Sprache beherrschen müssen. xvii, 4

Launch Datei Eine XML-Datei in der festgelegt wird, welche Launch Dateien (rekursiv), Stacks oder Packages mit einer konkreten Konfiguration gestartet werden sollen. 4

Manifest Eine XML-Datei, die ein Package oder Stack beschreibt und Abhängigkeiten offenlegt. xv, 4

Message Eine strikt getypte Datenstruktur, die primitive Datentypen, geschachtelte Messages und Arrays dieser beiden erlaubt [1, S. 3]. xv, 4

Node Ein Softwaremodul oder Prozess, der zur Laufzeit mit anderen Nodes über Messages innerhalb des Robot Operating Systems (ROSs) kommuniziert [1, S. 3]. xv, 4

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

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

Stack Eine Ordnerstruktur, die üblicherweise mehrere Packages einschließt und ein Manifest besitzt [1, S. 5]. xv, 4

Topic Eine asynchrone Informationsquelle und -senke welche durch einen String repräsentiert wird, über die Nodes Messages senden oder abonnieren können [1, S. 3]. 4

Akronyme

IDL Interface Definition Language. 4

ROS Robot Operating System. xv, 3, 4

KAPITEL 1

Einleitung

1.1 Motivation

lorem ipsum!

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 Testbed zurückgegriffen werden kann. Im Folgenden werden einige andere Möglichkeiten beschrieben und ihre Vor- sowie Nachteile bezüglich der Anwendung in einer Testimplementierung 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, das eine dynamische Positionsbestimmung erlaubt. Jedoch wäre man dann auf das GPS 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 sagen, 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 serwohl 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 auftretende Probleme gibt. Diese Art der Lokalisierung entspricht sinnhaft in etwa der der Odometriedaten. Es handelt sich hier quasi 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 Testsystems 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.

Außerdem muss es möglich sein, eine mathematische Analyse der Abweichungen der Sensordaten im Vergleich zu den Daten unseres Systems zu erstellen. Diese Arbeit soll jedoch hierauf noch keinen Fokus legen, da zunächst die zu verwendenten Systeme evaluiert werden sollen.

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 unseres Testsystems anzuzeigen.

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 Überleitung zu ROS, wir brauchen ROS!

2.2.2 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 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 beziehungsweise die einzelnen Module besonders gut getestet werden können.

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 zerlegt haben. Ein solcher Stack besitzt dann ebenfalls eine Manifest Datei.

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. 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 zwei solche Nodes miteinander kompatibel zu starten, muss in einem konkreten System in der Launch Datei eine Abbildung zwischen Topic Benennungen stattfinden.

Informationen werden in Messages zu einem Topic veröffentlicht. Diese 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.

- service - master - strikt in module (global clock, logger) - printf to rosout (single console)

Tools

- rosmake (rekursiv) - playbag - logger - roslaunch (hosts) - rxgraph - rviz

Generelle Funktionalität

- tf

2.2.3 OpenNI Stack

Pointclouds

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.