Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Christoph Schmiedecke

Aktor-Sensor-Simulation für Assistenzroboter

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Christoph Schmiedecke

Aktor-Sensor-Simulation für Assistenzroboter

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter : Prof. Dr. Wolfgang Fohl

Abgegeben am 06.12.2013

Christoph Schmiedecke

Thema der Masterarbeit

Aktor-Sensor-Simulation für Assistenzroboter

Stichworte

Assistenzroboter, Serviceroboter, Simulation, Gazebo, ROS, Aktor-Sensor-Simulation, Simulation von Assistenzrobotern, Simulation von Servicerobotern

Zusammenfassung

Gegenstand dieser Arbeit ist die Erstellung einer Simulation des im Robot Vision Lab der HAW Hamburg eingesetzten Roboters Scitos G5. Hierfür wird ein Modell in der Simulationsumgebung Gazebo erstellt, das die Aktoren und Sensoren des Roboters abbildet. Die Steuerung des Robotermodells erfolgt über implementierte Komponenten des Robot Operating Systems (ROS), welche bereits bei dem echten Roboter eingesetzt werden. Anschließend erfolgt eine Validierung des Modells gegen den realen Roboter, um zu entscheiden ob die Simulation für die weitere Entwicklung des Roboters verwendet werden kann.

Christoph Schmiedecke

Title of the paper

Actuator-sensor-simulation for assistant robots

Keywords

assistant robots, service robots, simulation, Gazebo, ROS, actuator-sensor-simulation, simulation of assistant robots, simulation of service robots

Abstract

The subject of this work is the development of a simulation of the robot Scitos G5 which is used in the Robot Vision Lab of the HAW Hamburg. A model of the robot including its sensors and actuators is created in the simulation environment Gazebo. The simulation model is controlled through Robot Operating System (ROS) components, which are also used for the real robot. The model is validated against the real robot in order to decide if the simulation model can be used for the further development of the robot.

Inhaltsverzeichnis

1 Einführung

Die Robotik ist heute aus vielen Bereichen des Lebens nicht mehr wegzudenken. Während herkömmliche Industrieroboter auch weiterhin neue Tätigkeitsbereiche für sich erschließen, bieten die neuesten Entwicklungen im Bereich der Informationstechnik und der Sensorik Einsatzmöglichkeiten, die über die traditionellen Bereiche der Automobil- und Elektroindustrie hinausgehen (vgl. International Federation of Robotics, 2013a). Zuverlässige Bin-Picking Verfahren[1] ermöglichen unter Einsatz von Sensoren und intelligenten Steuerungen eine Vielzahl von Montageprozessen und teilautonome Roboter unterstützen Menschen in Werkshallen z.B. beim Transport oder der Werkzeugführung (vgl. N-238784, 2013, S. 43).

Einen ebenfalls in den letzten Jahren stark gewachsenen Bereich der Robotik bilden die Service- oder Assistenzroboter (vgl. International Federation of Robotics, 2013b). Sie dienen z.B. als Unterstützung im Haushalt (vgl. (Siddhartha u. a., 2010) und (Reiser u. a., 2013)), im medizinischen Umfeld bei der Unterstützung von Operationen (vgl. Sun u. a., 2007), bei der Wiederherstellung motorischer Fähigkeiten nach einem Schlaganfall (vgl. Colombo u. a., 2000) oder im Servicebereich . Das Zukunftspotenzial der Robotik und dessen Marktchancen wurde bereits 1994 in einer Studie des Fraunhofer IPA (Hägele, 1994) prognostiziert und in der Folgestudie EFFIROB von 2010 (Hägele u. a., 2011) bestätigt. In dieser wurde sowohl die technische als auch die betriebswirtschaftliche Sicht von Servicerobotern in verschiedenen Szenarien und über verschiedene Lebensbereiche hinweg untersucht, z.B. die Unterstützung von Pflegepersonal bei der Bewegung von Patienten, das Ernten von Bodenfrüchten in der Landwirtschaft, als Produktionsassistenz in der Autoindustrie. Für die Steuerung der Roboter wurden besonders Herausforderungen bei der Wahrnehmung der Umgebung, der Navigation im Raum, der physischen Interaktion mit Objekten und dem Verhalten im Fehlerfall, vor allem in Bezug auf die Sicherheit im Umgang mit Personen, identifiziert. Weitere Herausforderungen sind hohe Kosten für die benötigte Hardware und für die Softwareentwicklung, welche einem schnelleren Wachstum entgegenstehen.

Ein generelles, allgemein bekanntes Problem der Robotik ist die Einbringung neuer Funktionen, Szenarien und Werkstücke in produktive Umgebungen. Die Anpassung an eine neue Situation erfolgte bisher verstärkt „online“, d.h. die Maschine oder sogar eine komplette Produktionsstraße muss ihren Betrieb einstellen, bis die neuen Bewegungen und Arbeitsschritte von einem Programmierer manuell eingestellt worden sind. Gängige Verfahren der online Programmierung sind „Teach-In“, hier werden die Gelenke des Roboters in die gewünschte Stellung gebracht und diese abgespeichert, und „Playback“, hierbei wird der Arm manuell vom Programmierer entlang der

[1] Bin-Picking oder der „Griff in die Kiste“ beschreibt eine Aufgabenstellung, bei dem ein Roboterarm ein Werkstück in einem Behälter mit meist unsortierten Werkstücken lokalisieren (Objektlokalisierung), greifen (Greiftechnik) und für weitere Aufgaben (Objektmanipulation) verwenden kann.

gewünschten Bewegungsbahn geführt und diese abgespeichert. Die entstehenden Ausfallzeiten bedeuten Stillstand und verursachen Kosten. Dies ist nicht wünschenswert.

Eine weitere Möglichkeit zur Anpassung des Roboters an die neue Situation bieten Verfahren, die einen Stillstand und die Ausfallzeiten der Produktion möglichst gering halten sollen. Bei diesen Verfahren werden die neuen Programme zuerst „offline“ am PC erstellt und anschließend online auf die Maschine aufgebracht und angepasst (vgl. Pires, 2007). Abhängig von Größe und Komplexität der Arbeitsumgebung und der Aufgabenstellung, insbesondere beim Einsatz von Servicerobotern, sind diese Verfahren nur begrenzt einsetzbar.

Industrieroboter arbeiten standardmäßig in einer statischen, für sie zugeschnittenen Umgebung (Arbeitszelle) in denen sich nur der Roboter und die zu bearbeitenden Werkstücke befinden. Die Umgebung wird dem Roboter und seiner jeweiligen Aufgabe schon beim Design angepasst. Das reicht von Anlagen mit einem Roboter bis hin zu ganzen Anlagenstraßen mit vielen Robotern, wie sie z.B. in der Automobilindustrie verwendet werden.

Serviceroboter hingegen arbeiten im direkten Umfeld des Menschen in sich ständig ändernden hochdynamischen Umgebungen. Menschen bewegen sich, verschieben Gegenstände oder fügen neue hinzu. Dies alles sind Dinge, mit denen der Roboter umgehen und sich zwischen ihnen zurechtfinden können muss.

Systeme müssen zukünftig also robust, flexibel und leicht zu bedienen sein, um eine rasche Anpassung an Veränderungen und Anforderungen der Umgebung und der Aufgabenstellung zu gewährleisten.

1.1 Motivation

Eine gute Möglichkeit zur Entwicklung und Planung komplexerer Abläufe, wie sie bei Servicerobotern auftreten, bietet eine computergestützte Simulation. Der Roboter und seine Umgebung werden im Rechner abgebildet, was eine flexible und gefahrlose Entwicklung, Erprobung und Optimierung für verschiedenste Aufgaben und Strategien in beliebigen Umgebungen ermöglicht. Eine Erweiterung des Roboters um zusätzliche Komponenten oder die Konzipierung komplett neuer Maschinen ist ohne direkten Zugriff auf die Hardware umsetzbar.

Ein Großteil der Probleme, die im Normalfall erst beim Aufbau der Arbeitszelle bzw. beim Einsatz des Roboters in einer realen Umgebung sichtbar werden, können frühzeitig erkannt und verhindert werden. Dadurch können neue Module und Ablaufpläne mit minimalen Anpassungen und damit verbundenen kurzen Online-Phasen in das reale System eingebracht werden.

Eine Simulation ist aber keine genaue Kopie der Realität (vgl. Siciliano und Khatib, 2008). Es handelt sich um eine idealisierte Umgebung, die gegebenenfalls nach den gewünschten Ergebnissen aufgebaut wird, dessen Qualität unter anderem von der Genauigkeit des verwendeten

Modells (des Roboters, der Robotersensoren, der Arbeitsumgebung), der Abbildung von Dynamik, Toleranzen und Abnutzung im realen System und von mechanischen Fehlern z.B. durch Wärme usw. abhängt.

Um zu überprüfen, ob eine Simulation für die Entwicklung eines Roboters verwendet werden kann, muss eine Bewertung der Simulation vorgenommen und diese anhand von praktischen Tests gegen ein reales System validiert werden. Als Validierungssystem steht im Zuge dieser Arbeit der im Robot Vison Lab der HAW Hamburg vorhandene Einarm-Roboter Scitos G5 zur Verfügung (vgl. Abbildung 1.1).

Abbildung 1.1: SCITOS G5-Plattform

1.2 Zielsetzung

Ziel dieser Arbeit ist zum einen die Simulation des Roboters Scitos G5 aus dem Robot Vision Lab. Im speziellen geht es um die Simulation der relevanten Aktoren[2] und Sensoren[3], und den zum aktuellen Zeitpunkt bereits umgesetzten Funktionalitäten (Fahren, Armbewegung u.a.). Zum anderen erfolgt eine Validierung der Simulation gegen den realen Roboter anhand verschiedener Szenarien. Übereinstimmungen und Unterschiede in Bezug auf Messdaten der unterschiedlichen Sensoren und welche Abweichungen es zwischen der Realität und der Simulation gibt werden evaluiert. Abschließend erfolgt eine Bewertung über die Qualität und Verwendbarkeit der Simulation in Hinblick auf eine Verwendung im Robot Vision Lab der HAW Hamburg.

Als Simulationsumgebung wird die Simulationsumgebung Gazebo (siehe Kapitel 2.3) verwendet, die für die Abbildung der Aktoren und Sensoren und der Roboterfunktionalität verantwortlich ist. Die Realisierung der Software-Funktionen des simulierten Roboters, die eine Benutzung dessen ermöglichen, erfolgt mittels ROS (siehe Kapitel 2.2), da die bereits realisierte Software-Funktionalität zur Steuerung des Scitos G5 in Form von ROS-Modulen vorliegt und somit diese Module mit geringen Anpassungen übernommen werden können.

[2] Aktoren: Im engeren Sinne die Motoren und Antriebe, die den Roboter bewegen. Im weiteren Sinne Dinge, die den Roboter mit seiner Umgebung interagieren lassen (Roboterarm, Manipulator, mobile Plattform).

[3] Sensoren ermöglichen dem Roboter seine Umgebung zu erfassen. Man unterscheidet zwischen verschiedenen Typen von Sensoren wie beispielsweise Kameras, Laserscanner und Kontaktsensoren.

Für den zu entwickelnden simulierten Roboter spielt die Abbildung der exakten physischen Realität des Validierungssystems nur eine begrenzte Rolle. So wird bei der Simulation die Dynamik des Systems außen vorgelassen bzw. nur insoweit implementiert, wie es für eine optische und funktionale Handhabung des simulierten Roboters im Robot Vision Lab nötig ist. Das heißt die Motoren und Antriebe des realen Roboters werden nicht in allen Einzelheiten nachgebaut. Die Bewegung und Steuerung des simulierten Roboters ähnelt in seinem Verhalten dem des realen Roboters.

1.3 Gliederung

Die Arbeit ist in folgende Abschnitte unterteilt:

- **Kapitel 2 - Grundlagen**

  Vorstellung des zu simulierenden Roboters inkl. seiner Aktoren und Sensoren sowie seiner Funktionen. Erläuterung der Simulationsumgebung Gazebo und des für die Steuerung des Roboters eingesetzten Frameworks Robot Operation System (ROS). Darüber hinaus wird noch ein Ausblick auf alternative Simulationsumgebungen gegeben.

- **Kapitel 3 - Anforderungen und Konzeption**

  Aufzeigen der Anforderungen für die zu entwickelnde Simulation und Erstellung eines Konzepts für deren praktische Umsetzung in Hinblick auf die vorhandene Zielsetzung. Betrachtung der zur Verfügung stehenden Modellierungsmöglichkeiten für die Abbildung des Roboters und seiner Funktionen, der Umsetzung von Sensorfunktionalität und der Visualisierung von Sensordaten und welche Einschränkungen bzw. Abweichungen in Bezug auf den realen Roboter zu beachten sind.

- **Kapitel 4 - Realisierung**

  Erstellung des Roboter-Modells, der zentralen Steuerungs-Plugins für die Kontrolle des Roboters und Entwicklung der benötigten Steuerung für die Kontrolle der Roboter-Gelenke. Realisierung der Roboter-Sensoren über Gazebo-Plugins und Implementierung der benötigten Schnittstellen für die Kommunikation mit den verwendeten ROS-Komponenten. Übersicht über die realisierten Module und die Abweichungen zum realen Roboter.

- **Kapitel 5 - Validierung**

  Validierung der Simulation gegen den realen Roboter anhand verschiedener Szenarien und Beurteilung der Verwendbarkeit der Simulation als Grundlage für die weitere Entwicklung des realen Roboters.

- **Kapitel 6 - Zusammenfassung und Ausblick**

  Zusammenfassung der entstandenen Simulation und der Erkenntnisse aus der Validierung sowie mögliche Erweiterungen der Simulation in zukünftigen Arbeiten.

2 Grundlagen

Dieses Kapitel gibt einen Überblick über das reale Referenzmodell des simulierten Roboters Scitos G5, im weiteren Verlauf der Arbeit „Scitos“ genannt, mit seinen Aktoren (Greifarm mit Endeffektor [1], Antrieb der Plattform) und Sensoren (Laserscanner, Kinect-Kamera, Bumper), die verwendeten Softwarekomponenten für die Realisierung der Simulation werden vorgestellt und die Bereitstellung der Funktionalität des Roboters und der Kommunikation der einzelnen Komponenten wird erläutert. Zusätzlich wird ein kurzer Ausblick auf weitere Simulationssoftware gegeben und es werden mögliche Alternativen zu Gazebo vorgestellt.

2.1 Roboter

Im Folgenden werden der Aufbau und die Komponenten des Scitos (siehe Abbildung 1.1) beschrieben. Es werden die für die Realisierung der Simulation wichtigen Komponenten und ihre Eigenschaften betrachtet. Die weiteren Bauteile der Plattform (wie Gehäuse und Rechner) und des Arms (wie Verbinder und Gelenkgehäuse) stellen im weitesten Sinne nur „funktionslose“ Bauteile für die Simulation dar und werden erst zum Zeitpunkt der Modellerstellung betrachtet.

Der Roboter lässt sich in die zwei für die Simulation wichtigen Komponentengruppen Aktoren und Sensoren aufteilen.

2.1.1 Aktoren

In Falle des Scitos bilden die Aktoren den Antrieb der mobilen Plattform, für die Bewegung des Roboters in seiner Umgebung und den Greifarm mit seinen Gelenken, für die Durchführung von Manipulationsaufgaben.

[1] Der Endeffektor ist die Hand oder das Werkzeug des Roboterarms. Mit ihm kann er Aktionen ausführen, wie z.B. das Greifen von Gegenständen. Die Bandbreite reicht hierbei von einfachen Werkzeugen und Greifern mit zwei Fingern bis hin zu komplexen mit Sensoren ausgerüsteten Händen.

2.1.1.1 Mobile Roboterplattform - Scitos G5

Die mobile Plattform des Roboters (siehe Abbildung 2.1) besteht aus einer geschlossenen Basis, in der hauptsächlich die Batterie und das dazugehörige Ladegerät, der Antrieb (Aktor) sowie Sonar- und Kontaktsensoren (Bumper) enthalten sind. Zur Befestigung weiterer Komponenten gibt es einen Aufbau, innerhalb dessen sich auch der Computer befindet, der die Hauptsteuerung des Roboters übernimmt. Der Greifarm ist oberhalb des Aufbaus montiert.

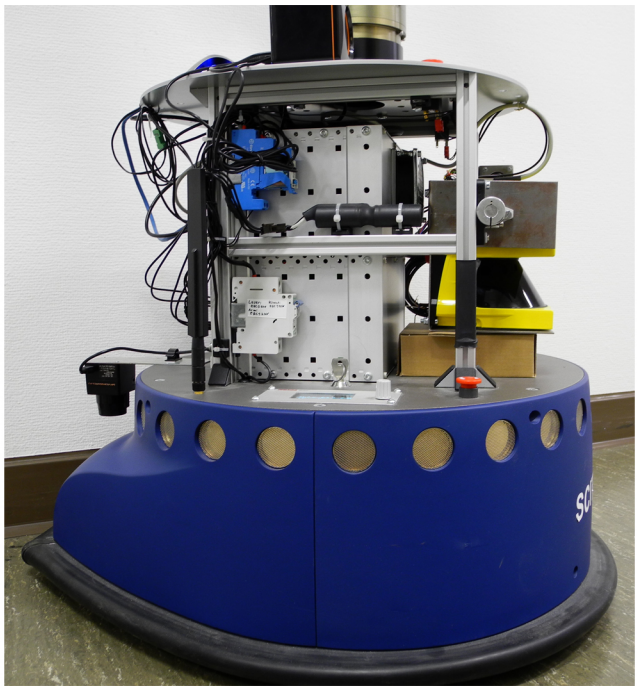


Abbildung 2.1: Mobile Plattform des Scitos G5.

2.1.1.2 Antrieb der mobilen Roboterplattform

Der Antrieb der Roboterplattform ist ein Differenzialantrieb, welcher durch zwei Getriebemotoren mit je einem Antriebsrad realisiert ist, d.h. jedes Rad kann unabhängig von dem jeweils anderen angetrieben werden. Die Antriebsachse liegt nicht im Mittelpunkt der Plattform, sondern ist nach vorne versetzt. Im Heck der Plattform befindet sich ein Laufrad. Durch den verschobenen Drehpunkt muss bei der Bewegung der Plattform die ausladende Bewegung des Hecks beachtet werden. Eine Drehung um den geometrischen Mittelpunkt des Roboters ist nicht möglich, nur um die Antriebsachse.

Die wichtigsten Daten der Plattform für die Simulation sind im Anhang A.1 aufgelistet. Alle weiteren Daten wie die Gehäuseabmessungen, der Versatz der Drehachse und die Position des Laufrads können der Anleitung des Scitos (vgl. G5 Manual, 2010) entnommen werden.

2.1.1.3 Schunk-Greifarm

Bei dem auf der mobilen Plattform montierten Roboterarm (siehe Abbildung 2.2) handelt es sich um einen 5 DOF[2] Arm. Dieser besteht aus Gelenkmodulen und Verbindungsstücken der Firma Schunk[3]. Die Gelenke werden im aktiven Fall über Motoren bewegt und im Falle eines Fehlers oder Stromausfalls über Magnetbremsen in ihrer Position gehalten.

Die für die Simulation wichtigen Daten wie Gelenkbegrenzungen und Drehgeschwindigkeiten sind im Anhang A.2 aufgelistet. Die Abmaße der Verbindungsstücke sowie die genauen Abstände und Positionen der Gelenke und Verbinder, können dem Benutzerhandbuch und der technischen Zeichnung (vgl. Schunk-Arm Manual, 2010) entnommen werden.

Abbildung 2.2: Roboterarm des Scitos G5

[2]DOF = Dergee of Freedom (Freiheitsgrad): Beschreibt die Bewegungsfreiheit des Roboterarms
[3]SCHUNK GmbH & Co. KG – http://www.de.schunk.com

2.1.2 Sensoren

Der Scitos G5 verfügt über Sonar- und Kontaktsensoren (Bumper) mit denen der Nahbereich des Roboters untersucht wird und ein direkter Kontakt der Plattform mit der Umgebung registriert wird. Zusätzlich verfügt der Roboter über zwei Laserscanner, die vorne und hinten an der mobilen Plattform angebracht sind und in einer zweidimensionalen Ebene die Umgebung erfassen, sowie über eine Kinect-Kamera, die auf dem Sockel des Endeffektors befestigt ist und Farb- und Tiefenbilder generiert.

Im Folgenden wird ein kurzer Überblick über die Funktionen und die Möglichkeiten der einzelnen Sensortypen gegeben.

Die genauen Abstände und Positionen der Sensoren am Roboter können den technischen Zeichnungen im Robot Vision Lab entnommen werden. Detaillierte Informationen zu den einzelnen Sensoren können den jeweiligen Produktbeschreibungen der Hersteller (vgl. (Leuze Spezifikationen, 2013), (Hokuyo Spezifikationen, 2013) und (Kinect Spezifikationen, 2013)) sowie dem Roboter Benutzerhandbuch (vgl. Schunk-Arm Manual, 2010) entnommen werden.

- **Laserscanner**

  Ein Laserscanner ist ein meist horizontal ausgerichteter Sensor, der anhand von Laufzeit- oder Phasenmessungen die Umgebung in einem bestimmten Winkelbereich abtastet und die Entfernung zu Objekten misst. Die erfassten Daten können zur Erstellung einer Umgebungskarte und damit auch zur Navigation und Kollisionserkennung verwendet werden. Die Reichweite und Genauigkeit dieser Sensoren ist deutlich besser als beispielsweise die von Sonarsensoren.

  In der aktuellen Konfiguration verfügt der Roboter über zwei Laserscanner. Bei dem vorderen Laserscanner handelt es um einen Leuze RS4-2E und bei dem hinten montierten Laserscanner um einen Hokuyo URG-04LX-UG01 (siehe Abbildung 2.1).

- **Kamera**

  Es gibt verschiedene Arten von Kameras die dem Roboter je nach Typ ein zweidimensionales Bild oder dreidimensionales Bild (Tiefenbildkamera) seiner Umgebung vermitteln.

  In der aktuellen Konfiguration verfügt der Roboter über eine Kinect-Kamera. Diese arbeitet intern mit einer Infrarot-Kamera in Kombination mit einem Infrarotlaser und mit einer Farbkamera. Die Kinect-Kamea ist somit in der Lage Tiefeninformationen über die Umgebung zu liefern und Farbbilder zu generieren. Über eine Kombination der beiden Bilder ist eine Darstellung der Tiefeninformation mit Farbwert möglich. Weitere Informationen zur Verwendung der Kinect-Kamera am Scitos G5 bietet die Ausarbeitung von Struss (2013).

- **Bumper**

  Bumper oder auch Kontaktsensoren ermöglichen es dem Roboter konkrete Kollisionen mit der Umgebung zu erkennen. Werden die Sensoren berührt oder eingedrückt, wird dies registriert und der Roboter kann eine gewünschte, programmierte Reaktion ausführen.

  Der Roboter verfügt über einen umlaufenden Ring von Bumpersensoren, die entlang des unteren Gehäuserands der mobilen Plattform angebracht sind. Lösen diese bei einer Kollision ein Signal aus, so blockiert der Antrieb des Roboters so lange, bis die Kollision beseitigt worden ist, etwa durch das manuelle Eingreifen durch den Menschen.

- **Sonar**

  Sonarsensoren senden Ultraschallwellen aus, die an Hindernissen reflektiert werden. Die Reflektionen werden wieder aufgefangen und können zur Hinderniserkennung oder zur Erstellung und zum Abgleich mit einer Umgebungskarte verwendet werden. Im Gegensatz zum Laserscanner sind die Sonarsensoren aber nur in einem kurzen Abstand zur Roboterbasis verwendbar und weniger genau.

  Die verwendeten Sonarsensoren sind ringförmig an der mobilen Plattform des Roboters befestigt und ermöglichen zwar eine Hinderniserkennung in alle Richtungen, die gelieferten Informationen sind aber nicht sehr genau und erlauben auf Grund ihrer Implementierung auch nur eine begrenzte nachträgliche Filterung.

2.2 Robot Operating System (ROS)

Die Entwicklung neuer Serviceroboter-Anwendungen bedeutet häufig einen großen Aufwand in Bezug auf die Integration von Hardware- und Softwarekomponenten (vgl. N-238784, 2013, S. 46). Viele Funktionen und Anwendungen werden neu geschrieben, obwohl gleiche oder kaum veränderte Hardwarekomponenten eingesetzt werden.

Wünschenswert ist ein System, welches eine Komponenten-basierte Entwicklung unter Wiederverwendung bereits bestehender Hardware- und Softwarekomponenten ermöglicht. Ein solches System verspricht einen großen Fortschritt bei der Entwicklung von neuen Servicerobotern und Serviceanwendungen (vgl. N-238784, 2013, S. 46).

Das Robot Operation System (ROS) ist ein Framework, welches diese Anforderungen erfüllt. Es ist eines der am weitest verbreitetsten auf dem Markt (vgl. Hägele u. a., 2011, S. 40). Dies und die Tatsache, dass die bereits vorhandenen Software-Module für die Steuerung und die Handhabung des Scitos G5 mit ROS realisiert worden sind, spricht für die Verwendung von ROS in dieser Arbeit.

Bei ROS handelt es sich um ein open-source Framework für Roboter, welches 2007 am Artificial Intelligence Labaratory der Stanford Universität im Zuge des Projekts STAIR[4] entwickelt worden ist und dessen Weiterentwicklung seither durch das Robotikunternehmen Willow Garage[5] vorangetrieben wurde. Seit diesem Jahr wird diese Aufgabe von der Open Source Robotics Foundation[6] (OSRF) übernommen.

Im Folgenden werden die Aspekte ROS-Kommunikationsschicht, Koordinatensystemverwaltung in ROS und die Installation von ROS näher erläutert.

2.2.1 ROS-Kommunikationsschicht

ROS beinhaltet Bibliotheken, Schnittstellen und Werkzeuge für die Entwicklung verschiedenster Arten von Robotern. Den Kern bilden hierbei die sogenannten **Nodes**. Diese bieten verschiedenen **Services** und **Topics** an, über welche der Nachrichten-basierte Austausch von Daten zwischen den Nodes erfolgt. Die Vermittlung der Nodes, Services und Topics untereinander erfolgt über einen zentralen ROS-Master.

2.2.1.1 ROS-Node

Ein ROS-Node[7] ist ein Prozess, der innerhalb der ROS-Umgebung eine bestimmte Funktion kapselt. ROS-Nodes kommunizieren in der ROS-Umgebung untereinander um Daten auszutauschen. Zur Kommunikation stehen Topics und Services zu Verfügung, die im Anschluss erklärt werden.

Für die Teilnahme an der Kommunikation ist eine Anmeldung am ROS-Master nötig. Beim Start meldet sich ein Node mit den von ihm angebotenen Topics und Services am zentralen ROS-Master an. Dieser fungiert als Vermittler und speichert die Registrierung von Nodes, Topics und Services.

2.2.1.2 ROS-Topic

Topics[8] sind namentlich identifizierte Datenkanäle, über welche Nodes miteinander kommunizieren und Daten austauschen. Jedes Topic hat einen festen Nachrichtentyp, welcher durch den sendenden Node festgelegt wird. Hierbei können zu jeder Zeit mehrere Nodes Daten senden (publishen) und empfangen (subscriben). Der Versand von Nachrichten über Topics erfolgt asynchron.

[4] STAIR (STanford Artitificial Intelligence Robot) – http://stair.stanford.edu
[5] Willow Garage, Inc – http://www.willowgarage.com
[6] Open Source Robotics Foundation: Offizielle Webseite – http://www.osrfoundation.org/
[7] ROS Nodes – http://wiki.ros.org/Nodes
[8] ROS Topics – http://wiki.ros.org/Topics

2.2.1.3 ROS-Service

ROS-Services[9] bilden ein dynamisches Remote-Procedure-Call (RPC) Grundgerüst. Die RPC-Kommunikation erfolgt synchron. Der Aufrufer des Service wird so lange blockiert, bis seine Anfrage mit einer Ergebnisnachricht beantwortet worden ist.

Vermittlung und Kommunikation zwischen Nodes

Nodes kennen ihre Gesprächspartner nicht. Sie subscriben zu einem Topic, dessen Daten sie interessieren. Daten, die Nodes produzieren, publishen sie über ein Topic und machen sie damit für anderen Nodes zugänglich.

Interessiert sich ein Node für ein Topic, so stellt er eine Anfrage an den ROS-Master, der ihn an den entsprechenden Node vermittelt. Die anschließende Verbindung und der Datenaustausch erfolgt danach direkt zwischen den Nodes (vgl. Abbildung 2.3). Der ROS-Master greift nur dann informierend ein, wenn sich an den registrierten Informationen Änderungen ergeben.

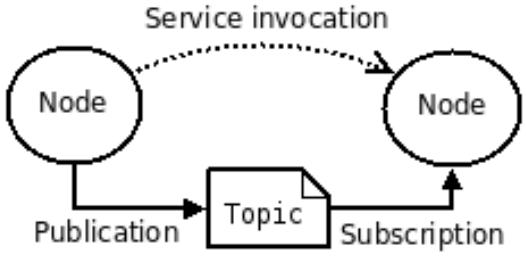


Abbildung 2.3: Einfache Darstellung des ROS-Kommunikationskonzepts[10] zwischen zwei Nodes basierend auf Topics und Services (Grafik-Quelle: Ros.org).

2.2.2 Koordinatensystemverwaltung in ROS

Viele ROS-Module verfügen über ein eigenes Koordinatensystem, dessen Zusammenspiel im Folgenden erläutert wird.

2.2.2.1 ROS-Coordinate Frame

Ein Frame ist ein Koordinatensystem (z.B. von Sensoren, dem Roboterarm, der Roboterbasis). Die Ausrichtung erfolgt rechtshändig (x vorne, y links, z nach oben). Die Koordinatensysteme müssen in Relation zueinander gesetzt werden können. Dies ist für die korrekte Anordnung der Komponenten im Raum und eine entsprechende grafische Darstellung notwendig. Diese Koordinatentransformationen werden über ROS-Transforms verwaltet.

[9] ROS Services – http://wiki.ros.org/Services

2.2.2.2 ROS-Transform

Ein ROS-Transform[11] beschreibt die Relation zwischen zwei Frames (vgl. Abbildung 2.4). Frames werden hierbei ausgehend von einem Root-Frame in einem Baum angeordnet. ROS-Transforms sind vollwertige Transformationen im dreidimensionalen Raum und bestehen aus einer Translations- und einer Rotationskomponente. Sie beschreiben die Position relativ zum jeweiligen Parent-Frame.

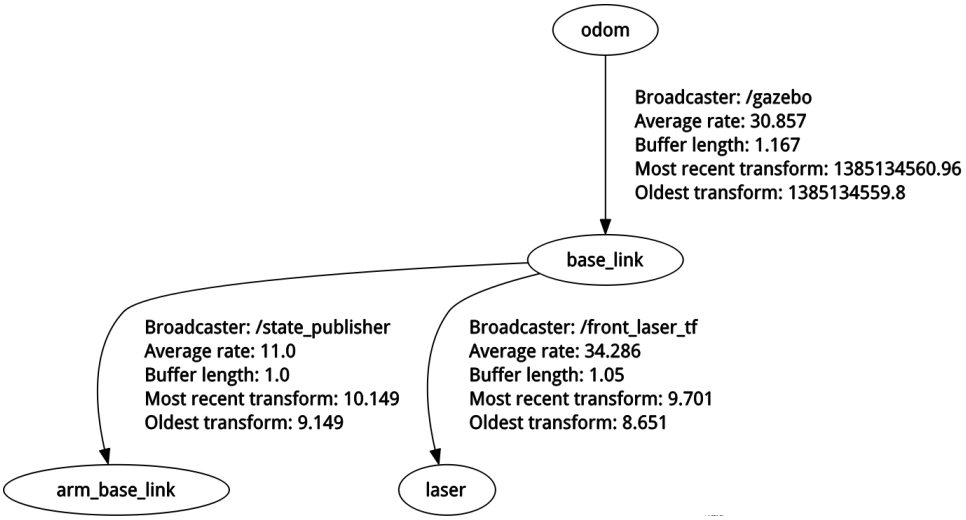


Abbildung 2.4: Beispiel eines ROS-Transform Baums mit den Koordinatensystem-Transformationen von odom, base_link, arm_base_link und laser und wie diese zueinander im Verhältnis stehen. Darüber hinaus erfolgt die Angabe von Daten zu Diagnose-Zwecken wie beispielsweise die Update-Rate der jeweiligen Koordinatensystem-Transformationsnachrichten und wann die älteste bzw. neueste Transformationsnachricht empfangen worden ist.

2.2.3 Installation von ROS

Aktuell ist ROS nur unter Unix-basierten Plattformen verwendbar. Die empfohlene Distribution ist Ubuntu. Die verwendete ROS Distribution ist ROS Groovy[12]. Die Installation erfolgt mit der Variante „Desktop-Full-Install“ von der offiziellen ROS Webseite[13]. Im Zuge der Realisierung der Simulation werden noch weitere ROS-Tools und -Bibliotheken verwendet, auf die an gegebener Stelle eingegangen wird.

[11] ROS Transforms: TF – http://wiki.ros.org/tf
[12] ROS Groovy: Übersicht – http://wiki.ros.org/groovy
[13] ROS Groovy: Installation – http://wiki.ros.org/groovy/Installation/Ubuntu

2.3 Gazebo Simulator

Gazebo[14] ist eine dynamische dreidimensionale multi-robot-Simulationsumgebung. Sie ermöglicht die Simulation von Robotern, Sensoren und Objekten sowie deren Entwicklung und Interaktion in beliebigen Szenarien. Ein Schwerpunkt liegt in der Abbildung eines realistischen Sensorfeedbacks. Die Interaktion von Objekten erfolgt physikalisch korrekt. Die Steuerung der Roboter erfolgt über Plugins, welche einen direkten Zugriff auf alle Aspekte der Simulationsumgebung inklusive der Physik-Engine, der Grafikbibliotheken und der Sensorgeneration ermöglichen.

Gazebo kann z.B. für folgende Aufgaben verwendet werden (vgl. Koenig und Howard, 2004):

- Rapid Prototyping: Parallele Entwicklung von Hardware und Software (u.a. mit Schnittstellen zu ROS und Solidworks).
- Reverse Engineering: Zur Bestimmung und Abbildung der Funktionalität komplexer Systeme, bei denen nicht auf Anhieb gesagt werden kann, wie diese funktionieren. Ein Beispiel hierfür ist der Segway (vgl. Koenig und Howard, 2004, S. 2153). Ein einfacher simulierter Zweirad-Antrieb hätte den realen Antrieb nicht korrekt wieder gegeben, da dieser durch Kippen und Neigen des Segways selbst gesteuert wird.
- Algorithmus-Design: Testen von Algorithmen in passenden Szenarien.
- Sensortests: Diese erfordern normalerweise echte Hardware.
- Szenarien-basierte Tests: Testen von Roboterverhalten und -verfahren in verschiedenen Situationen und Umgebungen.
- Verwendung in Robotik-Wettbewerben: Gazebo bildet die Basis für den Simulator DRCSim der DARPA Robotik Challenge[15] .

2.3.1 Entwicklung von Gazebo

Als Teil des Player Stage Projects[16] wurde Gazebo am USC Robotics Research Lab[17] entwickelt. Im Zuge des Projekts wurde es als mutli-robot-Simulator für 2D-Umgebungen konzipiert und verwendet. Das vermehrte Auftreten von all-terrain Robotern machte eine dynamische dreidimensionale Simulationsumgebung für Outdoor-Roboter notwendig (vgl. Koenig und Howard, 2004). Den Schwerpunkt bildet hierbei der Ansatz zur Schaffung einer realistischen Welt mit einer genauen Abbildung von Aktoren und Sensoren sowie deren physikalischem Verhalten.

[14] Gazebo Simulator – http://gazebosim.org/

[15] DARPA Robotic Challange: Offizielle Seite des Roboter-Wettbewerbs – http://www.theroboticschallenge.org/default.aspx

[16] Player Stage Project – http://playerstage.sourceforge.net

[17] USC Robotics Research Lab – http://robotics.usc.edu/?l=Projects:PlayerStageGazebo

Seit 2011 ist Gazebo unabhängig von dem Player Stage Projekt und wird durch die Open Source Robotics Foundation[18] unterstützt.

Seit Begin der Entwicklungen, Gazebo den Zugriff auf ROS-Funktionalitäten zu ermöglichen, gab es direkte Abhängigkeiten von Gazebo zu ROS. Dabei wurde das ROS-Paket simulator_gazebo bereitgestellt, welches Gazebo passend zur verwendeten ROS-Version installiert. Die daraus resultierenden jahrelangen Erfahrungen sowie die Unterstützung von ROS-Funktionalitäten in Gazebo und den Möglichkeiten von Gazebo selbst, führten zur Wahl von Gazebo als Simulationsumgebung für diese Masterarbeit. Eine Folge der Abhängigkeiten von Gazebo in Bezug auf die Einbindung von ROS war jedoch auch, dass die eigentliche Entwicklung von Gazebo der Entwicklung mit ROS-Unterstützung davon lief. In dem Bestreben, sich von der Abhängigkeit direkter ROS-Komponenten zu lösen, gab es im Laufe dieser Masterarbeit eine Vielzahl an Umstellungen von Seiten Gazebos, welche das Anpassen und Umschreiben der in dieser Arbeit entwickelten Plugins und Steuerungskomponenten nötig machte. Ab der aktuellen Gazebo Version (1.9) und ROS-Hydro gibt es keine direkte Abhängigkeit mehr zwischen Gazebo und ROS. Damit sollten die mit der Umstellung verbunden gewesenen Schwierigkeiten für zukünftige Benutzer von Gazebo keine Rolle mehr spielen.

2.3.2 Aufbau von Gazebo

Gazebo besteht aus zwei Hauptkomponenten: dem Gazebo-Server und dem Gazebo-Client. Der Gazebo-Server stellt die eigentliche Funktionalität von Gazebo bereit. Der Gazebo-Client ist für die Darstellung der Simulation und die Benutzer-Interaktion zuständig. Die Funktionalität dieser beiden Komponenten wird durch verschiedene Bibliotheken gewährleistet (vgl. Abbildung 2.5). Als Basis jeder Simulation dient die Definition einer Systemwelt. Diese kann den gewünschten physikalischen Gegebenheiten angepasst und es können beliebige Gegenstände, wie z.B. Roboter oder Hindernisse, eingefügt werden. Die Steuerung und Funktionalität von Objekten wird durch Plugins ermöglicht. Alle Elemente in Gazebo werden durch das Simulation Description Format (SDF) beschrieben, auf das im weiteren Verauf der Arbeit näher eingegangen wird (siehe Abschnitt 3.1.1).

2.3.3 Gazebo-Server

Der Gazebo-Server ist für die Berechnung der physikalischen Simulation und die Generierung der Sensordaten zuständig. Weiterhin verwaltet er die Namensauflösung und das Topic-Mangement, ähnlich der Funktion des ROS-Masters.

[18] OSRF (Open Source Robotics Foundation) – http://osrfoundation.org/

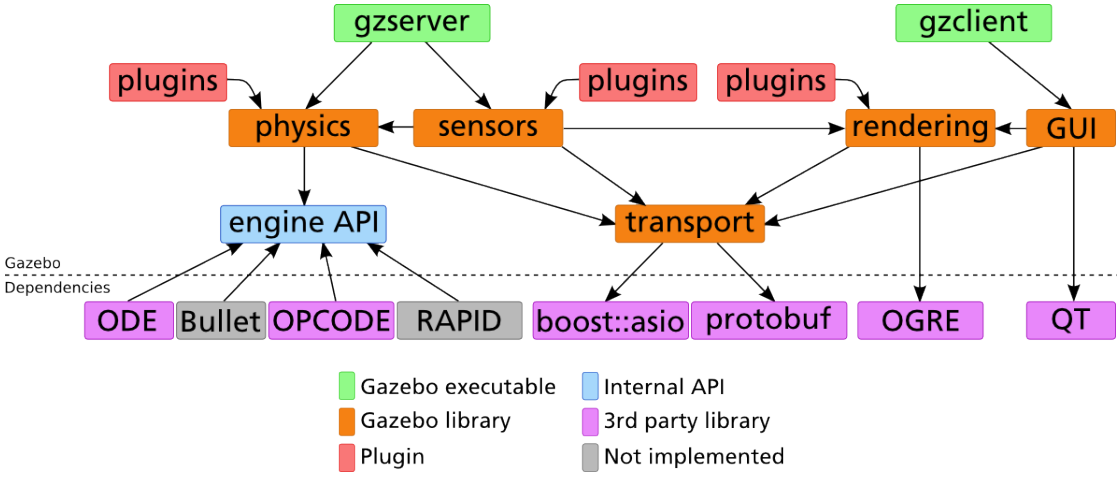


Abbildung 2.5: Gazebo-Architektur mit den beiden Hauptkomponenten (Gazebo-Server und-Gazebo Client), den Gazebo-Bibliotheken für die Abbildung der Physik, der Sensoren, der internen Kommunikation und der grafischen Darstellung sowie der Anbindung an externen Bibliotheken. (Grafik-Quelle: Gazebosim.org)

Als Physik-Engine wird die weit verbreitete open source engine Open Dynamics Engine[19] verwendet. Deren Ziel ist die Simulation der Dynamik und Kinematik von gegliederten, zusammenhängenden Körpern.

2.3.4 Gazebo-Client

Die Hauptaufgabe des Gazebo-Clients ist die grafische Darstellung des aktuellen Simulationsstands. Darüber hinaus bietet sie dem Benutzer die Möglichkeit mit der Simulation zu interagieren, beispielsweise durch das Einfügen und Entfernen von Objekten oder durch das Pausieren der Simulation.

Für die Visualisierung wird OpenGL[20] und GLUT[21] verwendet. Da die Berechnung des dynamischen Verhaltens viel Rechenleistung erfordert, soll diese nicht zusätzlich durch eine langsame oder aufwändige Oberfläche ausgelastet werden. OpenGL ist eine Standardbibliothek für die Erstellung von 2D- und 3D-Anwendungen. Durch die Auslagerung vieler Operationen in den Bereich der Grafikkarte wird die CPU entlastet. GLUT ist ein einfaches Interface für die Programmierung von Window-basierten Applikationen. Es ermöglicht die Anpassung und Erweiterung der vorhandenen GUI an die Bedürfnisse der eigenen Simulation. Auf diese wird jedoch nicht weiter eingegangen, da für die Steuerung des Roboters eine andere GUI verwendet wird (vgl. Abschnitt 4.4.4).

[19] ODE (Open Dynamics Engine) – http://opende.sourceforge.net/
[20] OpenGL – http://www.opengl.org/
[21] GLUT: The OpenGL Utility Toolkit – http://opende.sourceforge.net/

2.3.5 Gazebo-Interprozesskommunikation

Gazebos Interprozesskommunikation besteht aus einer Kombination von Googles Protobuf[22] Nachrichten und einer socketbasierten Kommunikation und imitiert das Verhalten von ROS-Nodes. Eine Simulation (der Server) veröffentlicht (published) Daten über den aktuellen Inhalt (z.B. die aktuelle Position eines Roboter, Sensordaten etc.). Die GUI (Client) abonniert (subscribed) diese Daten und bildet sie grafisch ab.

2.3.6 Gazebo-Plugins

Plugins sind von zentraler Bedeutung und bieten dem Programmierer eine direkte Möglichkeit alle Aspekte von Gazebo zu kontrollieren und auszulesen. Aspekte sind unter anderem die Welt der Simulation sowie vorhandene Modelle und Sensoren. Alle Steuerungsprogramme, aber auch die Abbildung von Sensoren, werden in Form von Plugins realisiert. Diese werden dynamisch zur Laufzeit geladen und ausgeführt. Die Kommunikation zwischen Gazebo und ROS findet ebenfalls über Plugins statt. Der Aufbau und die wichtigsten Funktionen eines Plugins werden in Abschnitt 3.4.2 behandelt.

2.3.7 Zeitmessung in Gazebo

Die Zeitmessung in Gazebo erfolgt anhand einer Simulationszeit. Diese beginnt bei jedem Start einer Simulation bei Null. Zusätzlich zur Simulationszeit wird die tatsächlich vergangene Zeit (Real Time) und das Verhältnis von Simulationszeit zu Real Time (Real Time Factor), also wie schnell die Simulation im Verhältnis zur Real Time läuft, gemessen. Die Simulationszeit stellt sicher, dass die gelieferten Simulationsergebnisse und das physikalische Verhalten korrekt abgebildet wird, wenn die Simulation auf Grund langsamer Hardware oder komplexer Umgebungen nicht in Echtzeit durchgeführt werden kann.

2.3.8 Gazebo-ROS-Kommunikation

Ab der Gazebo Version 1.9 besitzt Gazebo selbst keine Abhängigkeiten mehr zu ROS. Um ROS-Funktionalitäten unter Gazebo nutzen zu können bedarf es des Pakets `gazebo_ros_pkgs` (vgl. Abbildung 2.6). Dieses Paket kapselt Gazebo-Server und Gazebo-Client mit zwei internen Plugins. Durch diese Kapselung werden die benötigten Interfaces und Funktionen für Nachrichten, Services usw. zur Verfügung gestellt und Gazebo erscheint in der ROS-Umgebung als Node „gazebo“. Die Kapselung erfolgt automatisch durch einen abgeänderten Startbefehl von Gazebo.

[22] Google Protobuf (Protocol Buffer): Eine Sprachen- und Plattformunabhängiges Datenformat zur Serialisierung von strukturierten Daten. – http://code.google.com/p/protobuf

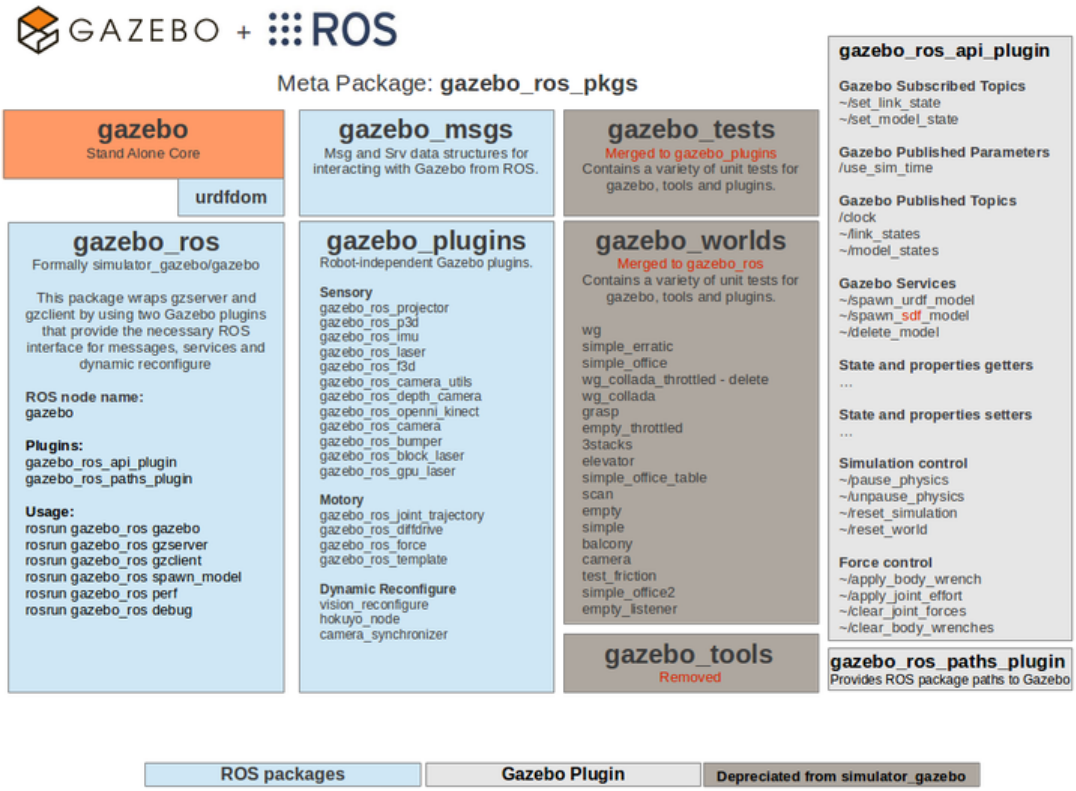


Abbildung 2.6: Gazebo-ROS-API für die Kommunikation zwischen Gazebo und ROS. `gazebo_ros` ist für die Kapselung des Gazebo-Servers und des -Clients zuständig, so dass diese als Node „gazebo“ in ROS erscheinen. `gazebo_msgs` ermöglicht die Kommunikation auf Basis von ROS-Nachrichten und -Datenstrukturen und `gazebo_plugins` simulieren bestimmte Funktionalitäten, wie Sensoren, in der Gazebo-Welt. (Grafik-Quelle: Gazebosim.org)

2.3.9 Hardware-Software-Simulation in Gazebo

Allgemein basiert die Simulation in Gazebo auf einem Weltenmodell. Dieses kann beliebig den benötigten Anforderungen an ein gewünschtes Test-Szenario angepasst werden.

Alle Komponenten in Gazebo, inklusive der Welt, der Gegenstände und der Hardware des Roboters mit ihren physikalischen Eigenschaften, werden mit der Beschreibungssprache SDF (Simulation Description Format) beschrieben. Im groben wird in SDF zwischen Verbindungselementen (Links), wie etwa dem Armteil eines Roboters, und Gelenken (Joints), die zwei Links miteinander verketten, unterschieden. Den Links und Joints können verschiedene Attribute wie Form, Aussehen, Kollisionsbereich u.a. zugewiesen werden. Die Software-Module zur Realisierung von Steuerungen und Sensoren erfolgt durch Plugins.

2.3.10 Installation von Gazebo

Zum aktuellen Zeitpunkt ist Gazebo nur unter Unix-basierten Plattformen verfügbar. Als bevorzugte Distribution wird Ubuntu empfohlen. Eine Unterstützung von Windows und Mac OS ist für die Zukunft geplant. Die Installation von Gazebo in der Version 1.9 mit ROS-Support wurde nach der Anleitung im Gazebo-Wiki durchgeführt[23].

2.4 Alternative Simulationsumgebungen

Auch wenn in dieser Arbeit eine Kombination von ROS, mit der die Software-Steuerung des realen Roboters realisiert ist, und Gazebo, das über umfangreiche Simulationsmöglichkeiten verfügt und native Schnittstellen zu ROS bietet, zur Realisierung der Simulation eingesetzt wird (siehe Kapitel 2.2 und 2.3.1), so gibt es noch viele weitere, teils kommerzielle oder herstellerseitige Simulationsumgebungen, die in ihren speziellen Einsatzgebieten Gazebo durchaus überlegen sind. Im Bereich Industrieroboter bieten beispielsweise die Hersteller KUKA und Dürr eigene Programme (KUKA.sim[24] und Eco Screen 3D-OnSite[25]) für eine originalgetreue Abbildung von Roboterzellen und die Planung der Arbeitsabläufe. Simulationsumgebungen frei beweglicher Roboter mit ROS-Support sind z.B. Stage[26], das auf 2D-Umgebungen begrenzt ist, v-rep [27], welches sich zum Zeitpunkt der Masterarbeit im Übergang von einer kommerziellen zu einer freien Software befand und nur stark begrenzten Support ermöglichte, oder die kommerzielle Anwendung Webots[28].

Im Folgenden wird etwas genauer auf zwei weitere Möglichkeiten zur Simulation eingegangen. Zum einen auf das Robotics Developer Studio von Microsoft, das nicht nur eine Simulationsumgebung bietet, sondern auch eine eigene Software-Architektur für Roboter. Damit ist es als eigenständige Alternative zu ROS in Kombination mit einer Simulationssoftware wie Gazebo anzusehen. Zum anderen wird ein Einblick in die an der HAW entwickelte Simulationsumgebung HAW-Sim gegeben, welche ihren Fokus auf eine reale Abbildung von Kamerasensoren und eine sensorgeführte Steuerung von Robotern setzt.

[23] Gazebo Wiki: Installation von Gazebo – http://gazebosim.org/wiki/Install
[24] Simulator der Firma KUKA: KUKA.sim – http://www.kuka-robotics.com/germany/de/products/software/simulation/
[25] Simulator der Firma Dürr: Eco Screen 3D-OnSite – http://www.durr.com/de/unternehmen/aktuelles/single/steuerungstechnik-auf-dem-neuesten-stand-einfacher-schneller-und-effektiver
[26] Open Source 2D Simulator: Stage – http://playerstage.sourceforge.net/index.php?src=stage
[27] Simulator der Firma Coppelia Robotics: v-rep – http://www.coppeliarobotics.com/
[28] Simulator der Firma Cyber Botics: Webots – http://www.cyberbotics.com/

2.4.1 Microsoft Robotics Developer Studio

Microsoft Robotics Developer Studio (RDS) bietet umfangreiche Tools für die Erstellung und das Testen von Robotern. Es beinhaltet ein Programmiermodel für die Entwicklung asynchroner zustandsgesteuerter Anwendungen. Den Schwerpunkt bilden die drei Hauptkomponenten Concurrency and Coordinate Runtime (CCR), welches die Handhabung von Datenströmen über Threads stark vereinfacht, die Decentralized Software Services (DSS), für eine einfache Kommunikation zwischen einem Roboter und verschiedenen PCs sowie der Visual Programming Language (VPL), einer visuellen Programmiersprachen, mit denen per drag-and-drop Anwendungen erstellt werden können. Zusätzlich beinhaltet das Microsoft RDS eine 3D-Simulationsumgebung für die Darstellung und das Testen der erstellen Roboter.

2.4.2 HAW-Sim

Die HAW-Sim ist eine Simulationsumgebung für Sensor-geführte Roboter, die im Zuge der Masterarbeit von Pohlmann (2013) entstanden und in Matlab und OpenSceneGraph realisiert ist. Matlab übernimmt alle nötigen Berechnungen während OpenSceneGraph die Darstellung und das Sensorfeedback liefert. Mit ihr sollen Roboter simuliert werden und Algorithmen für dessen Steuerung entwickelt und getestet werden können. Einen Schwerpunkt der Arbeit bildete hierbei die möglichst reale Abbildung eines Kamerasensors und der von ihr getätigten Aufnahmen.

Mit der HAW-Sim lässt sich zum aktuellen Stand der Entwicklung ein Roboterarm (Katana 6M189) mit einem Kamerasensor abbilden. Der Arm kann über Zielwerte bewegt werden und die Kamera liefert dabei eine sehr realitätsnahe Abbildung der Umgebung. Die Kommunikation mit der HAW-Sim erfolgt über Netzwerknachrichten. Die HAW-Sim lässt sich um beliebige eigene Roboter, Sensoren und Steuerungsverfahren erweitern und ermöglicht auf Grund ihres Kommunikationskonzeptes eine Anbindung an externe Anwendungen wie z.B. ROS.

Der in dieser Arbeit zu simulierende Roboter verfügt über eine Vielzahl an Sensoren und verwendet unterschiedliche Steuerungsverfahren. Weil diese Art von Sensorik noch nicht in die HAW-SIM integriert wurde, wird auf den Einsatz der HAW-Sim verzichtet.

3 Anforderungen und Konzeption

Im Folgenden werden die Anforderungen bezüglich des zu erstellenden Modells, der gewünschten Funktionen und der Möglichkeiten zur Steuerung des zu simulierenden Roboters sowie die Visualisierung der Sensordaten beschrieben. Darüber hinaus wird der Simulationsumfang bestimmt, d.h. welche Komponenten realisiert werden müssen oder vernachlässigt werden können und wie die benötigte Funktionalität, speziell die der Antriebe und Sensoren, in Gazbeo realisiert wird.

3.1 Robotermodell

Dieser Abschnitt befasst sich mit der Auswahl der verwendeten Beschreibungssprache zur Erstellung des Robotermodells und den Entscheidungen über einen modularen Aufbau des zu simulierenden Robotermodells (Abbildung der Hardwarekomponenten). Eine Betrachtung der Realisierungsmöglichkeiten für die Softwarekomponenten des Roboters und eine Abgrenzung der Simulation zum realen Roboter und zu dessen Verhalten erfolgt im Abschnitt 3.4.

3.1.1 Roboter-Beschreibungssprachen

Aus vorherigen Arbeiten zum Scitos existiert bereits eine Modellbeschreibung des Roboters. Warum dieses Modell für die Simulation in Gazebo nicht ausreichend ist und daher nicht verwendet werden kann, wird nachfolgend erklärt.

Sowohl Gazebo als auch ROS bieten Beschreibungssprachen zur Modellierung von Robotern. Gazebo hat mit dem SDF[1] (Simulation Description Format) ein eigenes Beschreibungsformat für Simulationsumgebungen geschaffen. Bei ROS erfolgt die Modellierung mit URDF[2] (Unified Robotic Description Format).

URDF

URDF ist ein XML-basiertes Format zur Beschreibung von Robotermodellen. Dieses ist in ihren Möglichkeiten jedoch recht eingeschränkt und bietet lediglich die Abbildung kinematischer und dynamischer Eigenschaften eines einzelnen Roboters mit seinen Links und Joints und ist auf die Beschreibung von Robotern beschränkt.

[1] Gazebo's Modellierungssprache SDF (Simulation Description Format) – http://gazebosim.org/sdf.html
[2] ROS's Modellierungssprache URDF (Unified Robotic Description Format) – http://wiki.ros.org/urdf

SDF

SDF ist ebenfalls ein XML-basiertes Format, geht jedoch über die reine Beschreibung eines Roboters hinaus. Es bietet eine Beschreibungsmöglichkeit vom Welt-Level (der Simulationsumgebung) bis hin zum Roboter-Level. Mit ihr können alle für eine Simulation benötigten Informationen und Einstellungen zur Physik abgebildet und beeinflusst werden.

Entscheidung für eine Beschreibungssprache

In der Anfangsphase dieser Arbeit hat Gazebo SDF als nativ unterstütztes Roboterbeschreibungsformat eingeführt und damit URDF abgelöst. URDF-Modelle konnten zwar weiterhin in Gazebo verwendet werden, da diese beim Laden intern in das SDF Format konvertiert wurden, die Umwandlung des vorhandenen Modells erwies sich in diesem Zusammenhang aber als nur begrenzt verwendbar. Die Integration benötigter Parameter für die Beschreibung der dynamischen Eigenschaften des Roboters konnte nicht wie benötigt durchgeführt werden. Aus diesem Grund wurde entschieden, das Modell des Roboters neu und nach SDF-Konventionen zu erstellen.

Ungeachtet dieser Tatsache wird das URDF-Modell in Zusammenhang mit dieser Arbeit verwendet, da es für die visuelle Repräsentation des Roboters in RViz (vgl. Abschnitt 3.2.4), über das bereits eine grafische Steuerung des realen Roboters möglich ist, benötigt wird.

3.1.1.1 Beispiele für den Aufbau eines Modells in SDF und die Einbindung von Plugins

Für die Erstellung eines Modells stehen in SDF die Basiskomponenten *Link* und *Joint* zur Verfügung. Links definieren alle Bauteile des Roboters. Joints verbinden Links unter Angabe kinematischer und dynamischer Eigenschaften und repräsentieren die Gelenkfunktionalität des Roboters.

Beispiel für einen Modellaufbau mit SDF

Der folgende Code zeigt den exemplarischen Aufbau eines Modells mit SDF. Das Modell besteht aus zwei Links, die über ein Gelenk verbunden sind. Zusätzlich wird ein Plugin für die Steuerung des Modells eingebunden.

```
1 <model name="modellname">
2
3     <link name="link1">
4     <!-- Hier werden alle Parameter des Links beschrieben. -->
5     <!-- Dies umfasst die Position und Ausrichtung, das
          Aussehen und die Kollisionszonen, sowie die
          Beschreibungen zur Abbildung der Dynamik. -->
6     </link>
7
8     <link name="link2">
9     <!-- Parameter -->
```

```
10      </link>
11
12      <joint name="link1_link2_joint" type="revolute">
13      <!-- Angabe der Links die miteinander verbunden werden
           sollen und Angabe der Drehachse des Gelenks -->
14      </joint>
15
16      <!-- Einbindung eines Plugins zur Steuerung des Modells -->
17      <plugin name="my_control_plugin" filename="libmy_plugin.so"></plugin>
18
19  </model>
```

Beispiel für die Einbindung eines Sensorplugins mit SDF:

Der folgende Code zeigt die exemplarische Einbindung eines Sensorplugins an dessen repräsentativen Modell-Link.

```
1   <link name='sensor_link'>
2
3       <sensor name="sensorname" type="sensortyp">
4       <!-- Angabe der Sensorparameter für die Abbildung und
           Realisierung in Gazebo -->
5
6           <!-- Laden eines Sensorplugins -->
7           <plugin name="my_plugin" filename="libmy_plugin.so">
8           <!-- Angabe der vom Plugin benötigten Parameter z.B.
               über welche ROS Topics die Daten publiziert werden
               sollen. -->
9           </plugin>
10
11      </sensor>
12
13  </link>
```

3.1.2 Grundüberlegungen für einen modularen Aufbau

Ein entwickeltes Robotermodell muss eine einfache und flexible Anpassung an mögliche zukünftige Änderungen und Erweiterungen des realen Roboters ermöglichen. Solche Erweiterungen können beispielsweise zusätzliche Gelenke im Arm, die Verwendung eines anderen Greifers oder eines alternativen Fahrgestells oder weiterer Sensoren darstellen. Aus diesem Grund wird das Robotermodell in Form von zwei Modulen realisiert, eins für die mobile Plattform und eins

für den Greifarm. Ein weiterer Vorteil einer solchen modularen Bauweise sind die modulspezifischen Entwicklungs- und Testmöglichkeiten, d.h. jedes Modul kann unabhängig von den anderen im Rahmen seiner Möglichkeiten getestet werden. So benötigen Navigationsverfahren nicht zwangsläufig einen Roboterarm.

Eine Unterteilung von Sensoren in Module ist insoweit durch die Beschreibung in SDF und Gazebo gegeben, als das deren Funktionalität in Form von Plugins realisiert wird. Das heißt für einen neuen Sensor muss das jeweilige Modell lediglich um einen repräsentativen Link für den Sensor erweitert und das Plugin an diesen Link gebunden werden.

3.2 Zu realisierende Funktionen des realen Roboters

Vor der Realisierung ist ein Blick auf den realen Roboter nötig, um eine Übersicht der bereits implementierten Funktionen des Roboters zu erhalten und um zu bestimmen, welche Funktionalität in der Simulation umgesetzt werden muss. Es wird zwischen Basisfunktionen, die eine einfache manuelle Steuerung des Roboters erlauben, und erweiterten Funktionen, die eine komplexere Bewegung durch Vorgabe eines Zielpunktes ermöglichen, unterschieden. Zusätzlich können Aufgaben anhand eines Aktionsplaners ausgewählt werden. Im Abschnitt 3.3 findet eine Betrachtung der Implementierung dieser Funktionen beim Scitos statt um festzustellen, welche ROS-Komponenten in der Simulation umgesetzt werden müssen.

3.2.1 Basisfunktionen

Basisfunktionen ermöglichen eine einfache manuelle Steuerung des Roboters und des Roboterarms. Hierbei geht es hauptsächlich um die Ansteuerung des Antriebs der Basis und der Gelenke des Roboterarms.

Zum aktuellen Zeitpunkt verfügt der Scitos über folgende Basisfunktionen:

- Manuelle Steuerung der Plattform über die Tastatur, über den Versand von Translations- und Rotationsgeschwindigkeiten via Topic-Nachrichten.
- Manuelle Steuerung des Arms über die Tastatur, über den Versand von Positionen via Topic-Nachrichten.
- Manuelle Steuerung der Plattform über einen Gamecontroller, über den Versand von Geschwindigkeiten via Topic-Nachrichten.
- Manuelle Steuerung des Arms über einen Gamecontroller, über den Versand von Geschwindigkeiten via Topic-Nachrichten.

Mit der Implementierung dieser Funktionen ließe sich der Roboter in der virtuellen Umgebung frei steuern.

3.2.2 Erweiterte Funktionen

Die Möglichkeiten eines manuellen Eingreifens in die Steuerung des realen und des simulierten Roboters ist von Vorteil, wenn dieser sich während seiner Fahrt in eine Situation gebracht hat, aus der er sich mit den implementierten Verhaltensmustern (z.B. aufgrund eines Minimal-Abstands zu Hindernissen während der Bewegung) nicht mehr selbstständig befreien kann. Ziel weiterer Entwicklungsschritte ist ein komplett autonomes Verhalten des Roboters zu erhalten. Dieses bedarf komplexer automatisierter Steuerungs- und Verhaltensmuster.

Bereits umgesetzte automatisierte Steuerungsmöglichkeiten des Scitos:

Zum Zeitpunkt dieser Arbeit ist eine Steuerung des Scitos im Raum über die Vorgabe einer Zielposition möglich. Hierbei wird das Ziel in einer Karte markiert. Der Roboter plant eine Trajektorie zum Ziel und setzt diese in Bewegungskommandos für den Plattformantrieb um. Zusätzlich findet eine Kollisionserkennung und -vermeidung statischer und dynamischer Hindernisse unter Verwendung der aktuellen Robotersensoren statt. Wird der geplante Weg versperrt, versucht der Roboter einen alternativen Weg zum Ziel zu finden, so weit dies möglich ist.

Auch wenn die Vorgabe der Zielposition noch manuell über die Umgebungskarte erfolgen muss, finden alle weiteren benötigten Schritte zur Erreichung der Zielposition bereits automatisch statt. In weiteren zukünftigen Szenarien soll der Roboter seine Zielpositionen für die Durchführung komplexer Aufgaben selbstständig bestimmen (z.B. Objektlokalisierung, Bestimmung der Objektposition, Berechnung eines Pfades zum Objekt und anschließende Manipulation durch den Roboterarm).

3.2.3 Planungs-Funktionen

In der Masterarbeit von Kolbe (2013) wurde ein zielorientierter Aktionsplaner für den Roboter entwickelt. Dieser ermöglicht es dem Benutzer eine Aufgabe auszuwählen, welche anschließend vom Roboter automatisch durch geschickte Auswahl und Ausführung von Aktionen durchführt wird. Die Auswahlmöglichkeiten basieren auf den aktuellen Fähigkeiten des Roboters. Für Entwicklungs- und Testzwecke ist eine Verwendung dieser Anwendung wünschenswert.

3.2.4 Steuerung und Visualisierung

Gazebo bietet eine API zur Programmierung eigener Steuerungs- und Ausgabekonsolen für die Erweiterung der vorhandenen GUI. Auf diese Möglichkeit wird zugunsten der Verwendung von RViz[3] verzichtet. RViz ist ein 3D-Visualisierungstool von ROS, das auch für die visuelle Steuerung und Darstellung des realen Roboters verwendet wird. Es ermöglicht die Darstellung des Robotermodells, der Sensordaten (Laserscan, Kamerabild etc.) und der grafischen Auswahl von

[3] ROS RViz: 3D-Visualisierungstool von ROS – http://wiki.ros.org/rviz

Navigationszielen über die Umgebungskarte. Die benötigten Informationen werden über ROS-Nachrichtenströme (Topics) ausgetauscht.

3.3 Verwendung bestehender ROS-Komponenten des realen Systems

Der Scitos verwendet bereits einige ROS-Komponenten für die Umsetzung der in Abschnitt 3.2 beschriebenen Funktionen. Diese Komponenten sollen durch die Kopplung von Gazebo mit ROS ebenfalls bei dem simulierten Roboter verwendet werden. Voraussetzung hierfür ist die Implementierung der benötigten Schnittstellen der jeweiligen ROS-Komponenten, die im Kapitel Realisierung durchgeführt wird.

Die folgenden ROS-Komponenten werden benötigt:

- teleop_twist_keyboard - Manuelle Steuerung der Roboterplattform per Tastatur.
- teleop_twist_keyboard_arm_cam - Erweiterte Form des teleopt_keyboards mit der Möglichkeit zur Ansteuerung der Gelenke des Roboterarms.
- slam_gmapping - Erstellung der Umgebungskarte des Roboters aus Laserscan-Daten und zur Positionsbestimmung des Roboters innerhalb der Karte.
- move_base - Generierung einer Trajektorie für eine automatische Bewegung des Roboters vom aktuellen Punkt zu einem gegebenen Ziel.
- arm_navigation - Visuelle Vorgabe einer Armpose für den Roboterarm. Anschließend erfolgt eine automatische Generierung der Trajektorie für die Bewegung. Auf Wunsch können diese Steuerungsdaten an den Arm gesendet werden.
- teleop_arm_controller - Steuerung des Roboters und des Arm über den Controller der PlayStation 3.

3.4 Simulationsumgebung

Bevor mit der Realisierung begonnen werden kann, muss der Umfang der Simulation bestimmt werden. Welche Dinge müssen realisiert oder können vernachlässigt werden, wie wird die benötigte Funktionalität, speziell die der Antriebe und Sensoren, realisiert, und welche Abweichungen gibt es zum realen System.

3.4.1 Realismus und Simulationstiefe

Gazebo ist eine dynamische Simulationsumgebung, d.h. alle Objekte unterliegen der Einwirkung physikalischer Kräfte. Durch diese Eigenschaft lassen sich realitätsnahe Simulationen generieren. Zum aktuellen Entwicklungsstand des Scitos, bei dem es im weitesten Sinne um Handlungs- und Aufgabenplanung, Bewegungs- und Greifstrategien sowie Objekt- und Hinderniserkennung geht, ist eine vollständige und reale Abbildung der Dynamik in der Simulation nicht ausschlaggebend. Gleichzeitig bleibt das Potenzial für spätere Entwicklungen, bei denen ein dynamisches Objektverhalten benötigt wird, erhalten.

Es wird vielmehr eine Simulation des Roboters benötigt, die alle wichtigen Parameter bezüglich der Dynamik berücksichtigt, um ein optisch ansprechendes und im Verhalten vernünftiges System zu erhalten, das in seinem Bewegungsverhalten und in den Sensordaten dem realen Roboter entspricht.

3.4.2 Realisierung von Softwarekomponenten in Gazebo

Softwarekomponenten werden in Gazebo generell über Plugins realisiert (vgl. Grundlagenkapitel 2.3.6), mit denen alle Aspekte der Simulationsumgebung (Welt, Roboter, Sensoren) beeinflusst und verändert werden können. Auf Grundlage der Überlegungen zur Erstellung eines modularen Modells (vgl. Abschnitt 3.1.2) werden für die Abbildung der Roboterfunktionalität und dessen Steuerung zwei Plugins benötigt, eines für die mobile Plattform und eines für den Roboterarm. Eine Hauptaufgabe dieser Plugins liegt in der Abbildung der Antriebe des jeweiligen Moduls, welche im nächsten Abschnitt beschrieben wird.

Beispiel für das Grundgerüst eines Plugins

Das folgende Beispiel beschreibt den generellen Aufbau eines minimalen Plugins:

```
1  #include <gazebo/gazebo.hh>
2
3  // Alle Plugins müssen sich im Gazebo Namespace befinden
4  namespace gazebo
5  {
6    // Klassenname und Typ des Plugins
7    // Es gibt vier Typen: World, Model, Sensor, System
8    class MyControlPlugin : public ModelPlugin
9    {
10
11     // Die Load Funktion ist die einzige erforderliche Funktion
          eines Plugins.
12     // Die Funktion bekommt ein SDF-Objekt übergeben das alle
          Informationen über das SDF-Objekt enthält, an welches das
          Plugin gebunden ist.
```

```
13      public: void Load(physics::ModelPtr _parent, sdf::ElementPtr
            _sdf)
14            {
15                // Initialisierung, z.B. für die Anbindung an ROS
16                // Aufruf weiterer Funktionen oder von Threads
17            }
18    };
19    // Registrierung des Plugins per Makro
20    GZ_REGISTER_MODEL_PLUGIN(MyControlPlugin)
21  }
```

Neben der Load-Funktion bietet Gazebo zusätzlich noch eine weitere die „OnUpdate“ genannt wird. OnUpdate ermöglicht eine Steuerung des Modells, ohne dass ein zusätzlicher Thread benötigt wird. Der Aufruf erfolgt im Aktualisierungstakt der Welt. Genauere Informationen bezüglich der Taktung und des zeitlichen Verhaltens sind der Gazebo-Dokumentation nicht zu entnehmen.

```
1  // Wird durch das Update Start Event der Welt aufgerufen
2  public: void OnUpdate()
3  {
4      // Realisierung der Steuerung des Modells
5      // z.B. auslesen der aktuellen Position, setzen von Gelenkwinkeln
6  }
```

3.4.3 Realisierung der Antriebe

Wie im Abschnitt (3.4.1) über Realismus und Simulationstiefe beschrieben, wird jedes Objekt in Gazebo von physikalischen Kräften beeinflusst. Das direkte Einstellen und Halten von Positionen für die Gelenke des Greifarms ist nicht möglich. Antriebe und Motoren, die in der Simulation durch Gelenke repräsentiert werden, verhalten sich nur dann wie gewünscht, wenn man diese aktiv regelt.

Gazebo bietet eine Basisklasse für einen PID-Regler. Dieser berechnet aus der gegebenen Differenz der aktuellen Position (Ist-Wert) und der Zielposition (Soll-Wert) und der vergangenen Zeit seit dem letzten Regulierungszyklus die zu setzende Kraft für den nächsten Regulierungsschritt. Die Abbildung der Antriebs- und Motorenfunktionalität findet durch die Implementierung eines PID-Reglers für jedes Antriebsgelenk statt. Durch die verschiedenen zur Verfügung stehenden Möglichkeiten zur Steuerung der Antriebe und Gelenke des Scitos (Vorgaben in Form von Geschwindigkeit, Beschleunigung oder Zielwinkel), müssen ggf. verschiedenen Varianten des PIDs umgesetzt werden. Die Bindung der Regler an die Links des Modells erfolgt in Form von Plugins.

3.4.4 Realisierung der Sensoren

Die Realisierung der Funktionalität der Robotersensoren findet ebenfalls durch den Einsatz von Plugins statt. Gazebo bietet bereits eine Vielzahl an Sensorimplementierungen, sodass diese nicht selbst geschrieben werden müssen. Die Sensoren werden als Teil des Robotermodells modelliert. Hierbei wird das jeweilige Plugin dem entsprechenden Sensor-Link im Modell zugewiesen. Beim Laden des Robotermodells zum Simulationsstart werden ebenfalls alle Plugins geladen und nutzbar. Der simulierte Roboter benötigt zwei Lasersensoren sowie einen Kamerasensor und einen Bumpersensor.

3.4.5 Simulations- und Echtzeit in Gazebo und ROS

Um Zeitverläufe darzustellen und nachvollziehbar zu machen verwendet Gazebo eine eigene Simulationszeit (vgl. Abschnitt 2.3.7). Dies ist vor allem zu beachten, wenn ROS-Komponenten verwendet werden sollen. Als Standardzeit verwenden diese normalerweise *ros_time*, welche sich nach der Zeit des ausführenden PCs richtet. Mit dem Parameter *use_sim_time*[4] in den Startdateien der ROS-Komponenten werden diese angewiesen, die von Gazebo veröffentlichte Zeit zu verwenden. Wird Gazebo neu gestartet, so wird auch die Zeit zurückgesetzt und alle anderen ROS-Komponenten müssen neu gestartet werden, da es sonst zu Problemen auf Grund alter Nachrichten kommen kann (Old Clock Messages Problematik).

Ein Aspekt, der zumindest kurz erwähnt werden sollte, ist die Simulation in Echtzeit. Gazebo kann, je nach Leistung des ausführenden Rechners, schneller oder langsamer als Echtzeit laufen. ROS ist nicht als echtzeitfähiges Framework ausgelegt. Der PR2-Roboter von Willow Garage, der sowohl in Gazebo als auch in ROS unterstützt wird, bietet eine Art Echtzeit-Controller, mit dem ein solches Verhalten abgebildet werden kann. Eine solche Betrachtung würde aber den Rahmen dieser Arbeit sprengen.

3.4.6 Abweichungen vom realen Roboter

Abgesehen von den Beschränkungen bzgl. der Realitätsnähe des Robotermodells (vgl. Abschnitt 3.4.1) gibt es noch weitere Abweichungen zum realen Roboter.

Zum einen wird das dynamische Verhalten des Roboters dadurch beeinflusst, dass die Trägheitsmomente der Bauteile mittels einer Einheitslösung beschrieben werden, zum anderen durch Unterschiede in der Abbildung der Robotergeometrie. Das simulierte Modell entspricht in seinem Aufbau ungefähr dem des Scitos, die einzelnen Bauteile weichen in ihren Formen jedoch vom Original ab und sind aus den Grundformen Quader, Kugel und Zylinder zusammengesetzt.

[4] ROS Time: Verwendung von Simulationszeit – http://wiki.ros.org/Clock#Using_Simulation_Time_from_the_.2BAC8-clock_Topic

Die Odometrie (Positionsbestimmung) des Scitos wird über einen Positionszähler im Antrieb seiner Plattform bestimmt. Dieser Zähler wird nicht modelliert, da die Position des Roboters direkt aus der Simulation bestimmt werden kann.

4 Realisierung

Dieses Kapitel beschreibt die Umsetzung der im vorherigen Kapitel aufgezeigten Anforderungen und des entwickelten Konzepts zur Erstellung des Robotermodells, der Antriebe und Sensoren sowie die Implementierung der benötigten ROS-Schnittstellen. Anschließend wird ein Überblick über die realisierten Komponenten gegeben und mögliche Abweichungen werden aufgezeigt.

Die Realisierung beachtet die Einschränkungen des Scitos bezüglich der Antriebe der Plattform und des Arms, d.h. deren maximale und minimale Gelenkwinkel, Beschleunigungen und Geschwindigkeiten. Ebenso erfolgt die Abbildung der wichtigsten Sensor-Eigenschaften wie beispielsweise Winkelbereich, Scan-Distanz und Abtastrate der Laserscanner.

Die Realisierung ist in mehrere Schritte unterteilt:

- Erstellung des Robotermodells
- Realisierung der Gelenkansteuerung
- Realisierung der Roboter-Sensoren
- Bereitstellung der ROS Software-Funktionalität zur Steuerung des Roboters
- Überprüfung der Realisierung

4.1 Erstellung des Robotermodells

Um den geplanten modularen Aufbau des Modells zu ermöglichen, erfolgt die Erstellung der Roboterplattform und des Roboterarms als jeweils eigenständiges Modell. Die Modellierung erfolgt mit dem SDF-Format.

Anmerkungen zu SDF

- In der aktuellen Version von SDF (1.4) müssen alle Positionsangaben absolut zum Koordinatenursprung des Modells angegeben werden und nicht relativ zum Koordinatenursprung des jeweiligen vorherigen Links.
- Die Repräsentation der Koordinationssysteme erfolgt nach der „Rechte-Hand“-Regel, wobei x nach vorne, y nach links und z nach oben zeigt. Diese Ausrichtung der Koordinatensysteme bleibt für jeden Link und jeden Joint bestehen. Die Bestimmung der Drehachse wird im Link über den Tag „axis“ festgelegt.

Repräsentativ für die Erstellung des Plattform- und des Armmodells erfolgt die Verwendung von Joints und Links am Beispiel des linken Rades der Roboterplattform. Kommentare erklären die wichtigsten sich wiederholenden Modellierungstags. Zu einigen dieser Tags gibt es weitere Optionen. Diese werden im Zuge dieser Arbeit und in Hinblick auf die geplanten Einschränkungen der Dynamik bisher nicht oder nur teilweise verwendet.

Link des linken Rades der Roboterplattform

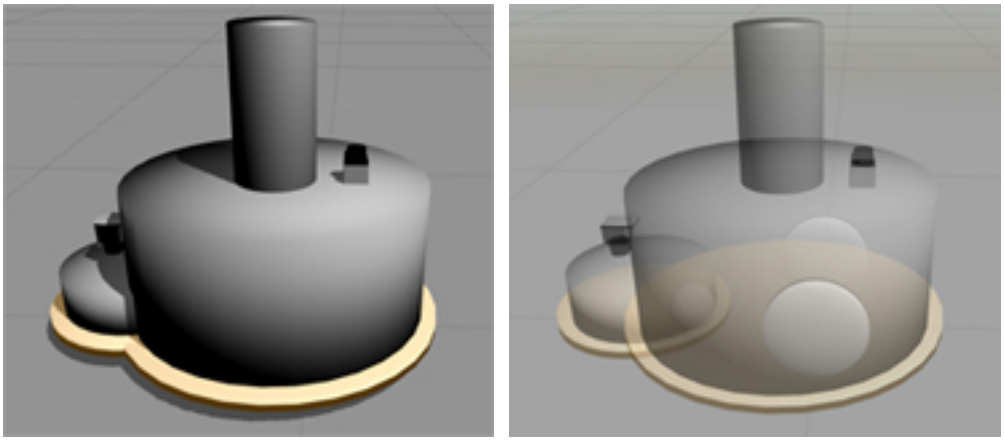
```
<!-- Repräsentativer Name des Links -->
<link name="left_wheel_link">
    <!-- Absolute Position zum Koordinatenursprung des Modells,
       dessen Ursprung liegt auf der Ebene der Welt -->
    <pose>0.075 0.155 0.05 -1.57 0 0</pose>
    <!-- Angaben für die Repräsentation der Trägheitsmomente -->
    <inertial>
        <mass>0.5</mass>
    </inertial>
    <!-- Definition des Kollisionsbereichs für den Link -->
    <collision name="left_wheel_collision">
    <!-- Absolute Position zum Koordinatenursprung des Links -->
        <pose>0 0 0.0 0 0 0</pose>
        <geometry>
            <cylinder>
                <radius>0.05</radius>
                <length>0.025</length>
            </cylinder>
        </geometry>
    </collision>
    <!-- Definition des Links und dessen optischer
       Repräsentation -->
    <visual name="left_wheel_visual">
        <pose>0 0 0.0 0 0 0</pose>
        <geometry>
            <cylinder>
                <radius>0.05</radius>
                <length>0.025</length>
            </cylinder>
        </geometry>
    </visual>
</link>
```

Joint des linken Rades der Roboterplattform

```
<!-- Repräsentativer Name des Joints-->
<joint name="left_wheel_joint" type="revolute">
    <!-- Position zum Koordinatenursprung des "'child"'-Links
       -->
    <!-- Kein Versatz bedeutet, dass der Joint im Ursprung des
       Links (des Rades) liegt -->
    <pose>0 0 0 0 0 0</pose>
    <!-- Namen der zu verbindenden Links -->
    <parent>base_link</parent>
    <child>left_wheel_link</child>
    <!-- Bestimmung der Drehachse -->
    <axis>
        <xyz>0 1 0</xyz>
    </axis>
</joint>
```

In diesem Sinne werden die komplette Basis und der Arm modelliert.

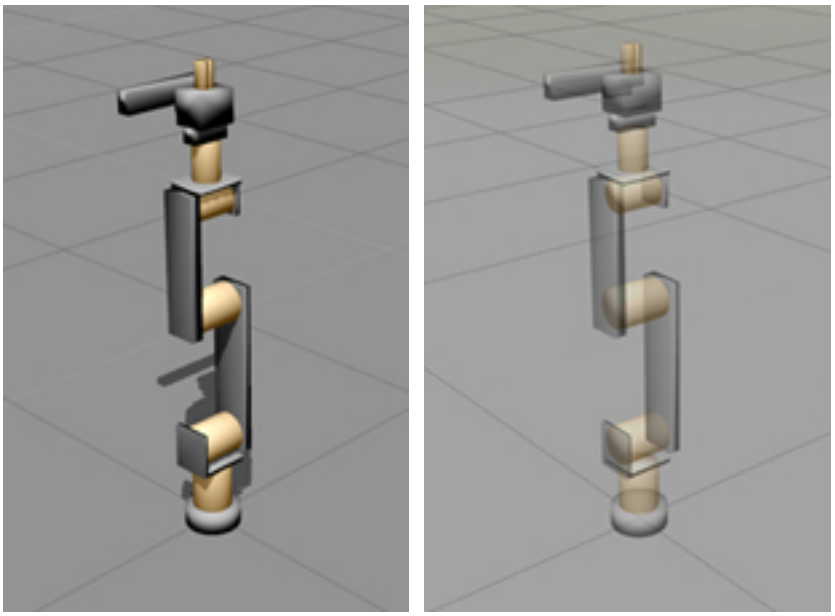
Die dabei entstandenen Modelle sind in Abbildung 4.1 (Roboterplattform) und Abbildung 4.2 (Roboterarm) zu sehen.



(a) Roboterplattform (b) Roboterplattform transparent

Abbildung 4.1: Roboterplattform des Scitos in Gazebo mit den Links für die Laserscanner und den Bumper-Ring (a) und in einer transparenten Darstellung für eine genauere Sicht des Aufbaus (b).

Für die weiteren Entwicklungsschritte wird das komplette Modell des Roboters (vgl. Abbildung 4.3) verwendet. Dieses entsteht durch die Verknüpfung der beiden Module. Um den Roboter bewegen zu können, bedarf es einer Möglichkeit zur Ansteuerung der Gelenke.



(a) Roboterarm (b) Roboterarm transparent

Abbildung 4.2: Roboterarm des Scitos in Gazebo mit dem Link für die Kinect-Kamera und dem Endeffektor (a) und in einer transparenten Darstellung für eine genauere Sicht des Aufbaus (b).

4.2 Realisierung der Gelenkansteuerung

Die Realisierung für die Steuerung der Roboterplattform und des Roboterarmes sowie die Kopplung aller Funktionen für die Abbildung des Roboterverhaltens erfolgt durch zwei Gazebo-Plugins.

Für die Steuerung der Gelenke in Gazebo erfolgt die Implementierung eines PID-Reglers. Die zu regelnden Größen hängen von der jeweils verwendeten ROS-Komponente ab, welche die Steuerungswerte generiert. Der Aufruf des PID-Reglers erfolgt in der OnUpdate-Methode des jeweiligen Plugins. Dies stellt sicher, dass zu jedem Updatezyklus eine Kraft auf die Gelenke wirkt und diese in Position hält. Die Bestimmung der Konfigurationsparameter für den PID erfolgt manuell anhand von praktischen Versuchen. Dieser Ansatz ist im Sinne der Zielsetzung für diese Simulation ausreichend. Der Aufbau der PID-Regelung ist nachfolgend anhand des Reglers für das linke Rad der Roboterbasis dargestellt, welches die Geschwindigkeitsangaben in m/s regelt. Die Implementierung für das rechte Rad ist entsprechend.

Die Steuerung der Armgelenke erfolgt über Winkelangaben in Rad. Die PID-Regelung wird entsprechend der Eingabewerte abgeändert, verhält sich aber ansonsten wie die abgebildete Regelung.

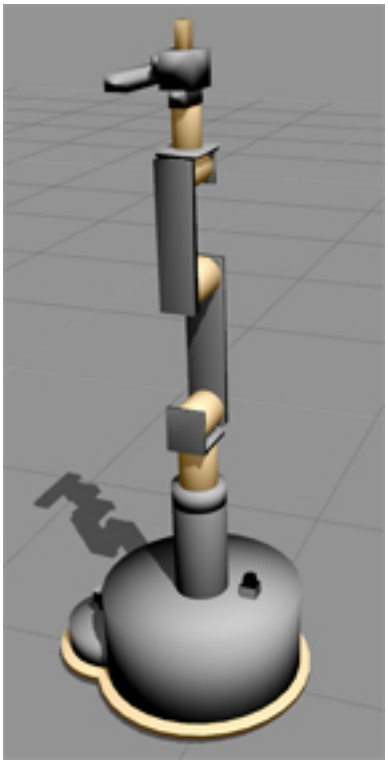


Abbildung 4.3: Simuliertes Modell des Roboters Scitos in Gazebo.

```
// Auszug des PID-Reglers für die Ansteuerung der Radgelenke

{
    // Compute the steptime for the PID
    common::Time curr_time_left =
       this->model->GetWorld()->GetSimTime();
    common::Time step_time_left = curr_time_left -
       this->prev_update_time_left;
    this->prev_update_time_left = curr_time_left;

    // Set the current position of the joint, and the target
       position, and the maximum effort limit
    // The wheels are controled by velocities, so "`target
       position"' = target velocity
    double vel_target_left = this->velL;
    double vel_curr_left =
       this->left_wheel_joint_ptr->GetVelocity(0);
    double max_cmd_left = this->joint_max_effort_left;

    // Calculate the error between the current position and the
       target one
    double vel_err_left = vel_curr_left - vel_target_left;
```

```
17
18      // Compute the effort via the Gazebo-PID library, which you will
           apply on the joint
19      double effort_cmd_left =
           this->left_wheel_pid->Update(vel_err_left, step_time_left);
20
21      // Check if the effort is larger than the maximum permitted one
22      effort_cmd_left = effort_cmd_left > max_cmd_left ? max_cmd_left :
23                  (effort_cmd_left < -max_cmd_left ? -max_cmd_left :
                       effort_cmd_left);
24
25      // Setting the new calculated velocity to the wheels (virtual
           engine)
26      this->left_wheel_joint_ptr->SetForce(0,effort_cmd_left);
27  }
```

Mit der Umsetzung der PID-geregelten Steuerung für die Gelenke können diese nun bewegt werden.

4.2.1 Mutual Exclusion

Der Scitos verfügt über verschiedenen Steuerungsmöglichkeiten. Das heißt, Bewegungskommandos an die Gelenksteuerungen können unabhängig voneinander von unterschiedlichen Quellen oder Datenkanälen eintreffen. Um sicherzustellen, dass diese Bewegungskommandos für jeweils einen Kommandosatz nicht durcheinander geraten und vollständig umgesetzt werden, wird der Zugriff auf die Variablen für die Bewegungskommandos per Mutex geregelt.

4.3 Realisierung der Roboter-Sensoren

Die verschiedenen Sensoren des Roboters werden ebenso wie die Steuerung der Roboterplattform und des Roboterarmes durch Plugins realisiert. Gazebo bietet bereits eine Auswahl an fertigen Sensor-Plugins, so dass diese nicht selber geschrieben werden müssen. Die Sensoren werden an den Sensor-repräsentierenden Link in der Modellbeschreibung gebunden. Die Konfiguration für den jeweiligen Sensortyp, beispielsweise die Reichweite, der Takt und der abgetastete Winkelbereich beim Laserscanner, erfolgt in der Modelldatei. Ein Beispiel für die Einbindung eines Sensors in SDF bietet die folgende Beschreibung für den vorderen Laserscanner.

```
14  <!-- Front laserscanner incl. Gazebo laserscanner-plugin -->
15  <link name="laser_front">
16    <!-- Definition of link-properties e.g. pose, visual
         appearence and collision zone as usual-->
17    <!-- ... -->
```

```
18
19    <!-- SDF-sensor-description: The sensor-type and
          (configuration) information for the handling in Gazebo -->
20    <sensor name="leuze_laser" type="ray">
21      <pose>0 0 0 0 0 0</pose>
22      <always_on>1</always_on>
23      <update_rate>10</update_rate>
24      <visualize>true</visualize>
25      <topic>scan</topic>
26
27      <!-- definition of the scan -->
28      <ray>
29        <scan>
30          <horizontal>
31            <samples>180</samples>
32            <resolution>1</resolution>
33            <min_angle>1.25</min_angle>
34            <max_angle>-1.25</max_angle>
35          </horizontal>
36        </scan>
37        <range>
38          <min>0.05</min>
39          <max>65</max>
40          <resolution>1</resolution>
41        </range>
42      </ray>
43
44      <!-- desired plugin with information needed for the
            plugin-call -->
45      <plugin name="gazebo_ros_laser"
            filename="/home/christoph/catkin_workspace_groovy/devel/lib/libgazebo_ros_laser.so">
46        <gaussianNoise>0.005</gaussianNoise>
47        <hokuyoMinIntensity>100</hokuyoMinIntensity>
48        <alwaysOn>true</alwaysOn>
49        <updateRate>10</updateRate>
50        <topicName>scan</topicName>
51        <frameName>laser</frameName>
52      </plugin>
53
54    </sensor>
55  </link>
```

4.3.1 Verwendete Gazebo-Plugins für die Realisierung der Roboter-Sensoren

Für die Realisierung der Sensoren im Robotermodell werden die folgenden drei Plugin-Typen verwendet:

- Laserscanner-Plugin (libgazebo_ros_laser)

  Das Plugin simuliert einen Laserscanner und publisht die erfassten Laserscan-Daten via Topic. Die Konfiguration der benötigten Parameter für die Abbildung der Laserfunktionalität wird basierend auf den Konfigurationsdaten der vorhandenen Laserscanner durchgeführt. Das Plugin wird für die Abbildung der beiden Laserscanner des Roboters verwendet.

  Über die Gazebo-eigene Physik- und Sensor-Engine wird die Laser-Funktionalität abgebildet. Sie erkennt hierbei Kollision im Laserstrahl des Roboters und verwendet jeweils die dem Roboter am nächsten liegenden Hindernisse.

- OpenNI-Kinect-Plugin (libgazebo_ros_openni_kinect)

  Das Plugin simuliert eine XBox-Kinect-Kamera und bietet ungefilterte Tiefeninformationen, ein RGB-Bild und ein Infrarot-Bild. Es wird für die Abbildung der Umgebung verwendet. In der aktuellen Konfiguration des Scitos wird das Bild zusätzlich zur Erkennung von Hindernissen verwendet, die im Bereich oberhalb der Scan-Ebene des vorderen Laserscanners liegen.

  Für die Realisierung der Kamerafunktion werden Gazebo- und ROS-Sensor-Nachrichten sowie die Point-Clouds-Bibliothek[1] zur Generierung der Bildinformationen verwendet.

- Bumper-Plugin (libgazebo_ros_bumper)

  Das Bumper-Plugin meldet Kollisionen mit der Umgebung. Informationen über den Kontakt des Bumper-Sensors mit seiner Umgebung werden per Topic versendet. Diese werden im Plugin zur Steuerung der Basis empfangen und bearbeitet.

  Gazebo hat eine Kollisionserkennungs-Engine implementiert, die alle Kollisionen in der Welt erkennt. Das Bumper-Plugin greift darauf zu, um festzustellen, ob Kollisionen mit dem Bumper-Link vorliegen.

Nach der Implementierung der Gelenkansteuerung und der Roboter-Sensoren erfolgt die Anbindung an das ROS-Netzwerk um die benötigten ROS-basierten Software-Funktionalitäten zur Steuerung des Roboters nutzen zu können.

[1] Point-Clouds-Bibliothek (PCL): Bibliothek für die Verarbeitung von 2D- und 3D-Bildern sowie von Punktewolken. – http://pointclouds.org/

4.4 Bereitstellung der ROS Software-Funktionalität

Für die Steuerung des simulierten Roboters werden die gleichen ROS-Komponenten (vgl. Abschnitt 3.3) wie bereits beim Scitos verwendet. Diese Komponenten lassen sich den Basisfunktionen, erweiterten Funktionen und Planungsfunktionen zur Steuerung des Roboters (vgl. Abschnitt 3.2) zuordnen. Für die Implementierung dieser Komponenten muss außerdem beachten werden, welche Datentypen sie anbieten bzw. benötigen und über welche Topics diese ausgetauscht werden. Diese bilden die benötigten ROS-Schnittstellen, die in den entwickelten Gazebo-Plugins implementiert werden müssen.

Einige dieser Nodes müssen zusätzlich nach den Spezifikationen des Roboters, wie beispielsweise die maximale Beschleunigung der Antriebsräder, konfiguriert werden. Die Konfiguration erfolgt über Konfigurationsdateien und wurde bereits für den Scitos vorgenommen. Da sich das Simulationsmodell des Roboter ausreichend detailliert an dem Scitos orientiert, können die vorhandenen Konfigurationen übernommen werden.

4.4.1 Basisfunktionen

Die Basisfunktionen beschreiben die Möglichkeiten zur Steuerung des Roboters mittels Eingaben der Tastatur oder des PlayStation-3-Controllers des Robot Vision Labs über den Versand von Geschwindigkeiten und Positionen.

teleop_twist_keyboard Ermöglicht die Steuerung des Roboters über die Tastatur anhand von Translations- und Rotationsgeschwindigkeiten.

bietet an: Translations- und Rotationsgeschwindigkeiten, Topic: /cmd_vel, Nachrichtentyp: geometry_msgs/Twist

teleop_twist_keyboard_arm_cam Eine erweiterte Form des teleopt_twist_keyboards mit der Möglichkeit zur Ansteuerung der Gelenke des Roboterarms über die Tastatur. Die Steuerung der Armgelenke erfolgt anhand von Zielpositionen. Die Identifizierung der einzelnen Armgelenke erfolgt über eine Gelenk-ID.

bietet an: Translations- und Rotationsgeschwindigkeiten, Topic: /cmd_vel, Nachrichtentyp: geometry_msgs/Twist

bietet an: Zielposition und Gelenk-ID, Topic: /schunk/move_position, Nachrichtentyp: metralabs_msgs/IDAndFloat

teleop_arm_controller Ermöglicht die Steuerung des Roboters und des Arms über den Controller der PlayStation 3. Die Steuerung der Plattform und des Arms erfolgt ähnlich wie bei teleopt_twist_keyboar_arm_cam, jedoch werden Ziel-Geschwindigkeiten statt Ziel-Positionen übermittelt.

bietet an: Geschwindigkeiten für den Plattformantrieb , Topic: /cmd_vel, Nachrichtentyp: geometry_msgs/Twist

bietet an: Geschwindigkeiten für die Steuerung der Armgelenke , Topic: /schunk/move_velocity, Nachrichtentyp: metralabs_msgs/IDAndFloat

4.4.2 Erweiterte Funktionen

Zum aktuellen Zeitpunkt kann sich der Scitos unter Verwendung der ROS-Komponenten `slam_gmapping` und `move_base` teilautonom bewegen. `slam_gmapping` erstellt anhand von Laserscanner-Daten eine Umgebungskarte und bestimmt die Position des Roboters. `move_base` berechnet zu einer gegebenen Zielkoordinate eine Trajektorie auf Basis von zu verwendenden Antriebsgeschwindigkeiten mit denen der Roboter die Zielkoordinate erreicht.

slam_gmapping Erstellt aus Laserscan-Daten des Roboters eine Umgebungskarte und bestimmt die Position des Roboters innerhalb der Karte.

abboniert: Laserscandaten vorderer Laser, Topic: /scan, Nachrichtentyp: sensor_msgs/LaserScan

abboniert: Koordinatensystemdaten, Topic: /tf, Nachrichtentyp: tf/tfMessage

bietet an: Erstellte Karte, Topic: /map, Nachrichtentyp: nav_msgs/OccupancyGrid

bietet an: Koordinatensystemdaten, Topic: /tf, Nachrichtentyp: tf/tfMessage

move_base Generiert einer Trajektorie auf Basis von zu verwendenden Antriebsgeschwindigkeiten für eine automatische Bewegung des Roboters vom aktuellen Punkt zu einem gegebenen Ziel.

abboniert: Erstellte Karte, Topic: /map, Nachrichtentyp: nav_msgs/GetMap

abboniert: Laserscandaten vorderer Laser, Topic: /scan, Nachrichtentyp: sensor_msgs/LaserScan

abboniert: Laserscandaten hinterer Laser, Topic: /scan_rear, Nachrichtentyp: sensor_msgs/LaserScan

abboniert: Koordinatensystemdaten, Topic: /tf, Nachrichtentyp: tf/tfMessage

abboniert: Odometriedaten, Topic: /odom, Nachrichtentyp: nav_msgs/Odometrie

abboniert: Einfaches 2D-Ziel, Topic: /move_base_simple/goal, Nachrichtentyp: geometry_msgs/PoseStamped

bietet an: Translations- und Rotationsgeschwindigkeiten, Topic: /cmd_vel, Nachrichtentyp: geometry_msgs/Twist

WarehouseViewer Ein grafisches Werkzeug das die visuelle Vorgabe einer Armpose für den Roboterarm ermöglicht. Anschließend erfolgt eine automatische Generierung der Trajektorie für die Bewegung. Auf Wunsch können die Steuerungsdaten der Trajektorie an den Arm gesendet werden. Für die Darstellung und die Berechnung der Trajektorie wird das URDF-Modells des Scitos verwendet, da nur dieses Format unterstützt wird.

Der WarehouseViewer basiert auf einem weiteren Steuerungskonzept von ROS, sogenannten „Actions[2]“. Hierbei handelt es sich um einen Ansatz zur Steuerung und Planung komplexer Aufgaben. Nach jeder Ausführung eines gesendeten Steuerungsbefehls erfolgt eine Rückmeldung an den Planer, so dass dieser den nächsten Steuerungsbefehl sendet. Wird die Rückmeldung nicht gesendet, weil der Roboter eine andere Teilaufgabe zu erledigen hat, ist die Bewegungsausführung „pausiert“, bis eine Rückmeldung erfolgt.

abboniert: Grafisch eingestellte Pose des Roboterarms

bietet an: Trajektorie, Action: control_msgs/FollowJointTrajectoryAction, Nachrichtentyp: control_msgs/FollowJointTrajectoryGoal

4.4.3 Planungsfunktionen

Der zielorientierten Aktionsplaner (vgl. 3.2.3) ermöglicht es dem Benutzer Aufgaben über eine GUI auszuwählen, die der Roboter anschließend automatisch durch geschickte interne Auswahl von Aktionen durchführt. Der Planer für die Bereitstellung der möglichen Aufgaben basiert auf den existierenden ROS-Schnittstellen und -Controllern des Scitos. Der interne Aktionsplaner für die Ausführung der Aufgaben benötigt keine weiteren Informationen des Roboters. Somit kann der Aktionsplaner einfach gestartet werden und die angebotenen Aufgaben werden von dem simulierten Roboter empfangen.

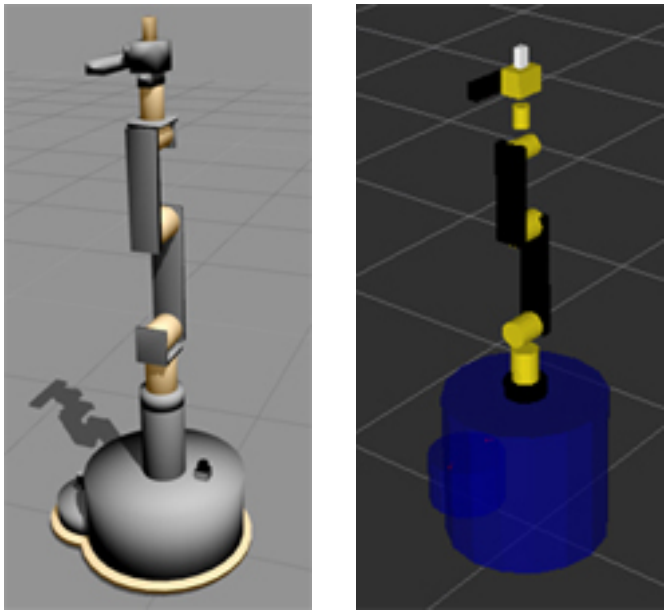
4.4.4 Visualisierung und Steuerung in ROS-RViz

RViz ist ein 3D-Visualisierungstool von ROS, das für die visuelle Steuerung und Darstellung des realen Roboters und seiner Sensoren verwendet wird (vgl. Abschnitt 3.2.4). Da RViz die benötigten Informationen über ROS-Topics austauscht und der Scitos sowie der simulierte Roboter die gleichen ROS-Module nutzen, erfolgt die grafische Steuerung ebenfalls über RViz. Für die Repräsentation des Roboters in RViz wird das URDF-Modell des Scitos verwendet, da in RViz das SDF-Format nicht unterstützt wird (vgl. Abbildung 4.4).

Je nachdem welche Anzeige- und Steuerungs-Plugins in RViz geladen werden, variieren die abonnierten und angebotenen Topics. In Bezug auf die Simulation müssen für die Verwendung

[2] ROS-Actionlib: ROS-Bibliothek für die Ansteuerung unterbrechbarer Aufgaben. – http://wiki.ros.org/actionlib

von RViz keine zusätzlichen Schnittstellen bereitgestellt werden, die über die vorher beschriebenen Funktionen hinausgehen.



(a) Darstellung des simulierten Roboters in Gazebo

(b) Darstellung des Scitos Modells in RViz

Abbildung 4.4: Die Simulation des Scitos erfolgt in Gazebo. Hier werden auch die Sensordaten anhand der simulierten Umgebung erstellt (a). Die reine Darstellung der Roboterposition und der Sensordaten erfolgt in RViz anhand eines vereinfachten Modells (b).

4.4.5 Schnittstellenimplementierung

Für die Verwendung der ROS-Funktionen mit dem simulierten Roboter werden die im vorherigen Abschnitt beschriebenen Schnittstellen der jeweiligen ROS-Nodes in den Plugins des simulierten Roboters implementiert. Die Implementierung erfolgt durch Funktionen, die Daten entsprechend der Nachrichtentypen der Schnittstellen abonnieren, auswerten und abarbeiten bzw. benötigte Daten vorbereiten und publizieren.

Im Anschluss wird beispielhaft eine der implementierten Schnittstellen mit den benötigten Datentypen, den sendenden und empfangenden ROS-Nodes sowie ihre Implementierung in den Roboter-Plugins beschrieben. Eine vollständige detaillierte Liste der implementierten Schnittstellen befindet sich im Anhang in Abschnitt B.

Topic /cmd_vel Datentyp: geometry_msgs/Twist

Sender: teleop_twist_keyboard, teleop_twist_keyboard_arm_cam, teleop_arm_controller, move_base

Empfänger: Roboter-Basis

Implementierung: Base-Plugin

Verarbeitende Funktion: cmdVelCallback()

- Empfang von Translations- und Rotationsgeschwindigkeiten in m/s für die Steuerung der Roboter-Basis.
- Umwandlung der Daten von m/s in rad/s.
- Weiterleiten der Daten an die simulierte Regelung des Plattformantriebs.

Mit der Implementierung der Schnittstellen kann der simulierte Roboter genau wie der Scitos über die verschiedenen ROS-Komponenten gesteuert werden.

Es folgt nun eine Übersicht der realisierten Module und es werden Abweichungen der Simulation zum realen Roboter aufgezeigt.

4.5 Übersicht der realisierten Module

Im Folgenden wird betrachtet, welche Komponenten und Funktionen des Roboters realisiert worden sind.

Realisierung des Robotermodells Die Modellierung des Robotermodells erfolgte modular in Form von zwei Modellbeschreibungen, eine für die Roboter-Basis und eine für den Roboter-Arm. Diese Module wurden zusammengefügt und bilden damit den kompletten Scitos ab. Sie können aber auch separat voneinander in Gazebo eingefügt und verwendet werden.

Realisierung der Gelenkansteuerungen Die Steuerung der Gelenke des simulierten Roboters erfolgt über PID-Regler, die in den geschriebenen Gazebo-Plugins zur Steuerung der Roboter-Basis und des Roboter-Arms implementiert sind. Die Regelung erfolgt im zeitlichen Aktualisierungsintervall der simulierten Welt und stellt sicher, dass die aktuell gültigen Gelenkvorgaben gehalten beziehungsweise umgesetzt werden. Hierfür wurden verschiedene PID-Regler für die jeweiligen Eingabeparameter der Gelenke implementiert.

Realisierung der Sensoren Die Sensoren wurden durch Gazebo-Plugins realisiert. Der simulierte Roboter verfügt wie der Scitos über zwei Laserscanner, eine Kinect-Kamera und einen Bumper-Ring. Die Verwendung der Sensordaten erfolgt durch verschiedene ROS-Nodes.

Schnittstelle	Status	Anmerkung
/cmd_vel	implementiert	—
/schunk/move_position	implementiert	—
/schunk/move_velocity	offen	Die Ansteuerung der Armgelenke über Geschwindigkeiten wird aktuell noch nicht unterstützt. Die Herausforderung liegt hierbei in der Integration des PID-Reglers für die Positionen und des PID-Reglers für die Geschwindigkeiten in einer Kaskadenregelung. Eine ausführlichere Erklärung hierzu findet sich in Abschnitt 4.6.
/schunk/follow_joint_trajectory	implementiert	—
/scan	implementiert	—
/scan_rear	implementiert	—
/joint_states	implementiert	—
/tf	implementiert	—
/odom	implementiert	—
/bumper_base	offen	Die benötigte Empfangsfunktion ist implementiert. Das Gazebo Bumper-Plugin erkennt aktuell keine Kollisionen. Eine Implementierung ist ohne großen Aufwand möglich, sobald das Bumper-Plugin die entsprechenden Daten liefert.

Tabelle 4.1: Schnittstellenimplementierung der verwendeten ROS-Komponenten. Die Plattform kann über Angaben von Geschwindigkeiten gesteuert werden und der Arm über die Vorgabe von Zielpositionen entsprechend den Anforderungen an die Simulation. Die (optionale) Steuerung des Arms über Geschwindigkeiten über den PlayStation3-Controller ist noch nicht implementiert.

Realisierung der ROS-Funktionalitäten Die Anbindung für die Kommunikation mit dem ROS-Netzwerk wurde über die Gazebo-Plugins des simulierten Roboters realisiert. Die Verwendbarkeit der verschiedenen ROS-Funktionalitäten wird durch die implementierten Schnittstellen und Funktionen sichergestellt. Für die Einbindung weiterer ROS-Funktionalitäten können den Plugins die entsprechende Funktionen hinzugefügt werden. Eine Übersicht der implementierten Schnittstellen bietet die Tabelle 4.1.

Vollständige Abbildung aller Komponenten und Schnittstellen Das Komponentenmodell in Abbildung 4.5 zeigt eine Übersicht aller erstellten Plugins und Module in Gazebo und der verwendeten ROS-Komponenten mit den implementierten Schnittstellen und Nachrichtenströmen zwischen der Simulation und dem ROS-Netzwerk. Ebenso sind die Komponenten

abgebildet, die bei dem Scitos den Empfang und Versand der Nachrichten regeln und den Roboter steuern.

4.6 Abweichungen der Simulation zum realen Roboter

Dieser Abschnitt beschreibt Abweichungen des erstellten Modells gegenüber dem realen Roboter Scitos.

Aufbau des Robotermodells Das Modell des simulierten Roboters besteht aus einfachen Formen wie Quadern, Zylindern und Kugeln und bildet damit viele Teile des Roboters nur in so weit ab, wie dieses für die Optik und das Verhalten in der Simulation benötigt wird. Eine komplette Nachbildung des Roboters war für die Aufgabenstellung dieser Masterarbeit nicht erforderlich.

Die Verbinder zwischen den Gelenken sowie der hintere Ansatz des Roboters sind nur als visuelle Teile des Roboters modelliert. Dies hat Einfluss auf die Abbildung des Trägheitsverhaltens der Roboterbasis.

Abbildung der Physik Die Abbildung der Trägheitsmomente erfolgt mittels Verwendung einer Einheitsmatrix. Diese bildet das reale Trägheitsverhalten der einzelnen Komponenten des Roboters nur ansatzweise ab.

Auf Grund der Verwendung von visuellen Modulen für die Modellierung bestimmter Teile des Roboters wird der Schwerpunkt einzelner Bauteile des Roboters nicht exakt abgebildet.

Für die Räder wird aktuell keine Bodenreibung beachtet. Einmal beschleunigt stoppt der Roboter nicht mehr von alleine.

Die Umsetzung der Roboterbeschleunigung erfolgt ungeregelt. Der Roboter erreicht aus dem Stand heraus die vorgegebene Geschwindigkeit ohne Verzögerung auf Grund einer Beschleunigungsphase. Umgekehrt steht der Roboter augenblicklich, wenn die Geschwindigkeit der Antriebe auf null gesetzt wird. Es gibt keine Bremsphase.

Koordinatensysteme der verwendeten Robotermodelle Die Koordinatensysteme des simulierten Roboters wurden entsprechend den Vorgaben der SDF-Beschreibungssprache nach der rechten Hand-Regel modelliert in Anlehnung an das Gelenkverhalten des Scitos. Hierbei wurde die Drehrichtung der Gelenkachsen vertauscht. Dies hat zur Folge, dass alle eingehenden Gelenkpositionen für die Steuerung des Arms, bei dem die Daten von ROS-Nodes stammen die für ihre Funktionen das URDF-Modell verwenden, negiert werden müssen. Dies ist bereits für die aktuell verfügbaren Funktionen implementiert.

Umsetzung der Sensoren Die Sensoren in Gazebo sind ideale Sensoren und sind keinen äußeren Einflüssen ausgesetzt. Diesbezüglich wurden keine weiteren Tests durchgeführt.

Abweichungen bei der Gelenkansteuerung Der Scitos wird über Motoren gesteuert. Diese werden im simulierten Modell durch PID-Regler abgebildet. Die Parameter der Regler wurde experimentell bestimmt weshalb es Unterschiede zum Steuerungsverhalten des Scitos geben kann. Bisher ist dies nur durch leichte Bewegungen im Arm beim Anfahren und Bremsen mit hohen Beschleunigungen beziehungsweise Geschwindigkeiten aufgefallen.

Zusätzlich verfügt der Scitos über Magnetbremsen, die die Gelenke bei einem Stromausfall in Position halten. Diese Bremsen sind nicht abgebildet. Die Gelenke werden zu jedem Zeitpunkt über die PID-Regler in Position gehalten.

Zeitverhalten der Steuerung Das exakte Zeitverhalten der Roboterbewegung wurde im Zuge dieser Arbeit nicht validiert. Die Bewegungen des simulierten Roboters kommen denen des Scitos aber sehr nahe, da die ROS-Komponenten, die für die Steuerung der Plattform und des Arms zuständig sind, die gleichen Konfigurationsdateien verwenden wie beim Scitos. Dies stellt sicher, dass nur gültige Steuerungsparameter an den Roboter übermittelt werden.

Abweichungen der ROS-Software-Funktionalitäten PlayStation-3-Controller: Zum aktuellen Zeitpunkt ist die Steuerung des Arms über den PlayStation-3-Controller noch nicht implementiert. Bisher erfolgt die Steuerung der Arm-Gelenke über die Angabe von Zielpositionen. Der dazugehörige PID-Regler ist für die Regelung von Zielpositionen ausgelegt. Die Steuerung über den PlayStation-3-Controller verwendet hierfür jedoch Zielgeschwindigkeiten. Eine Integration der beiden PID-Regler in einer Kaskadenregelung steht noch aus.

Funktionale Abweichungen zwischen dem simulierten Roboter und dem Scitos Die Bumper-Funktionalität ist zum gegenwärtigen Zeitpunkt noch nicht umgesetzt. Der Bumper-Ring und die Empfangsfunktion für den zukünftigen Empfang von Bumper-Nachrichten sind bereits implementiert. Aktuell sendet das Bumper-Plugin von Gazebo keine Information bei Kollisionen des Bumpers mit seiner Umgebung.

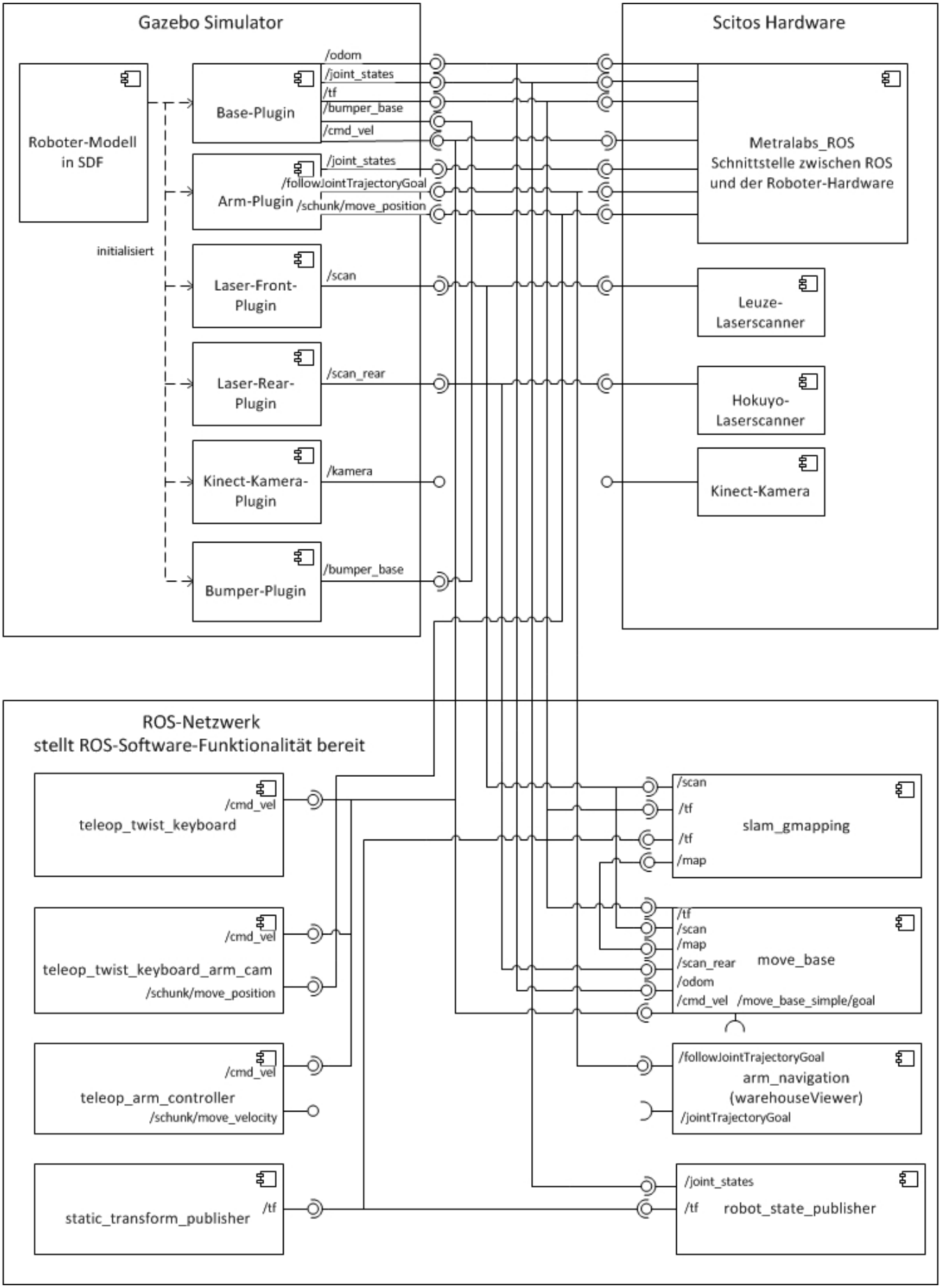


Abbildung 4.5: Komponentendiagramm der realisierten Komponenten und Schnittstellen.

5 Validierung

Im vorangegangen Kapitel wurden das Simulationsmodell des Scitos und die für die Steuerung nötigen Softwarekomponenten entwickelt sowie vorhandene ROS-Komponenten angepasst und deren Schnittstellen implementiert. Um eine Bewertung über die Qualität und Verwendbarkeit der Simulation für eine weitere Entwicklung des Scitos vornehmen zu können, wird nachfolgend eine Validierung des simulierten Roboters und seiner Sensoren gegen den realen Roboter Scitos anhand verschiedener Szenarien durchgeführt.

5.1 Validierungskriterien

Für die Validierung des simulierten Roboters ergeben sich zwei interessante Testfelder. Zum einen die Ansteuerung der simulierten Antriebe der Roboter-Gelenke, speziell die Umsetzung von Steuerungsbefehlen in Bewegungen und die Steuerung des Roboters unter Verwendung der ROS-Komponenten zur Bewertung des Roboterverhaltens, zum anderen die Überprüfung der verschiedenen Sensoren und den von ihnen gelieferten Messwerten. Um die Validierung gegen den Scitos nachvollziehbar zu machen, wird ein 3D-Modell des Flurs und des Robot Vision Labs erstellt, in dem die Tests des echten Roboters durchgeführt werden (vgl. Abbildung 5.1). Das zeitliche Verhalten spielt bei der durchgeführten Validierung keine Rolle, da das Senden und Empfangen von Daten und dessen Auswertung in den ROS-Komponenten unterschiedlich lange dauern kann und für die Durchführung der Aufgaben nicht entscheidend ist.

Die Auswahl der Szenarien wird begrenzt durch die aktuellen Möglichkeiten zur Steuerung des Scitos und ist in zwei Gruppen unterteilt.

Verhaltens-basierte Validierung

- Bewegung des Roboters zu einer Zielposition auf freier Strecke
- Bewegung des Roboters zu einer Zielposition unter Beachtung eines Hindernisses
- Bewegung der Roboter-Basis und des Roboter-Arms über den Aktions-Planer

Sensor-basierte Validierung

- Vergleich der Laserscanner-Sensoren anhand einer erstellten Umgebungskarte
- Vergleich des Kinect-Kamera Sensors anhand verschiedener Bilddaten

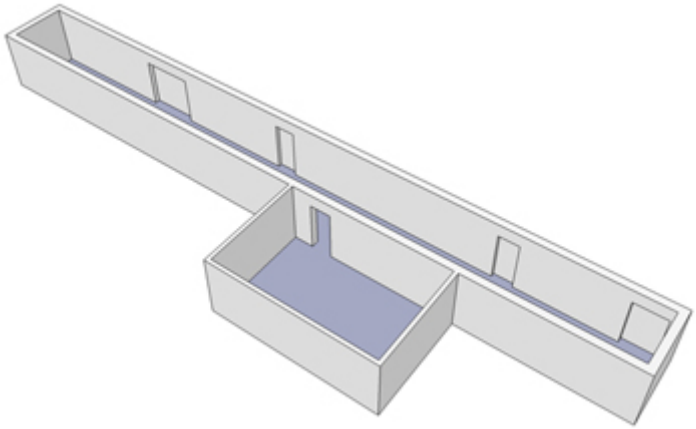


Abbildung 5.1: 3D-Modell des Flurs vor dem Robot Vision Lab in dem die Messungen mit dem Scitos durchgeführt werden.

5.2 Verhaltens-basierte Validierung

Dieser Abschnitt befasst sich mit der Ansteuerung der Roboter-Gelenke und der Auswertung des daraus resultierenden Roboterverhaltens in verschiedenen Szenarien. Die Eingabe der Ziele für die Bewegung der Roboterbasis erfolgt über ROS-RViz. Die Erstellung der Umgebungskarte als Basis für die Generierung der Trajektorie findet durch den ROS-Node `slam_gmapping` anhand der gelieferten Daten des vorderen Laserscanners statt. Die anschließende Generierung der Trajektorie mit den Steuerungsparametern für den Roboter erfolgt durch den ROS-Node `move_base`. Für die Generierung der Daten zur Bewegung des Arms wird wahlweise der ROS-WarehouseViewer oder der Aktions-Planer verwendet. Bei den verwendeten Modulen handelt es sich um sensitive Systeme. Das heißt, die generierten Trajektorien können auch bei einer exakten Abbildung der Testumgebung aufgrund von kleinsten Unterschieden der Start- und Zielposition voneinander abweichen. Eine Bewertung der ausgeführten Bewegungen erfolgt daher auf Basis von Erfahrungswerten mit dem Scitos.

5.2.1 Bewegung des Roboters zu einer Zielposition auf freier Strecke

In diesem Versuch führt der Roboter eine Zielfahrt auf freier Strecke durch. Der Roboter erhält eine Zielvorgabe zu der er sich hinbewegen soll. Die Abbildung des Versuchs ist in der Abbildung 5.2 zu sehen.

Das Ziel dieses Versuchs ist die Überprüfung des Plattformantriebs in Bezug auf seine Funktionsfähigkeit und die Umsetzung der Steuerungsbefehle.

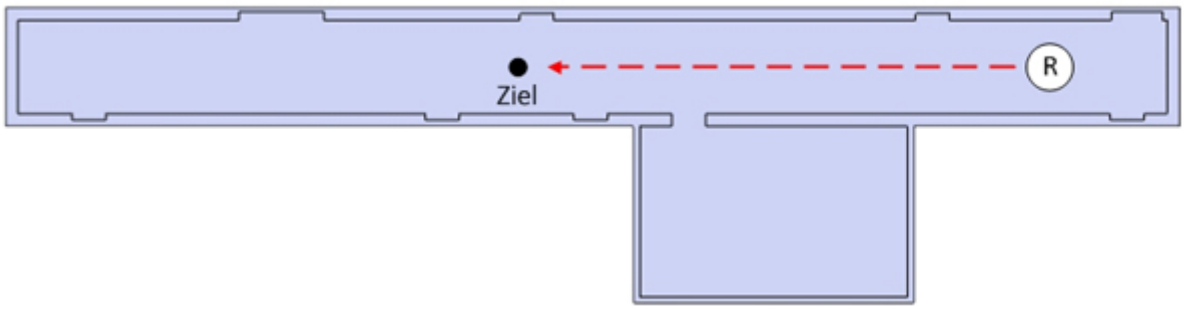


Abbildung 5.2: Versuchsaufbau des ersten Szenarios: Zielanfahrt einer gegebenen Zielposition ohne Hindernis.

Auswertung

Der Roboter hat die Steuerungsbefehle für die Antriebssteuerung vom zuständigen ROS-Node erhalten und sich zur Zielposition bewegt. Die Umsetzung der Antriebsbefehle und das daraus folgende Fahrverhalten entsprach hierbei in akzeptabler Weise dem Verhalten wie es bei der Fahrt des Scitos zu beobachten ist.

5.2.2 Bewegung des Roboters zu einer Zielposition unter Beachtung eines Hindernisses

Bei diesem Versuch handelt es sich um eine erweiterte Version des vorherigen Versuchs bei dem der Flur durch ein Hindernis verschmälert wird. Der Roboter soll sich an dem Hindernis vorbei bewegen, sich drehen und hinter dem Hindernis positionieren (vgl. Abbildung 5.3).

Das Ziel dieses Versuchs ist die Überprüfung des Roboterverhaltens in einer Umgebung mit Hindernissen. Der Raum zwischen dem Hindernis und der Flurwand ist eingeschränkt und bietet dem Roboter daher weniger Spielraum für Bewegungen. Dies ist vor allem eine Herausforderung für die Trajektorienplanung, die bei der Planung eines Weges immer einen konfigurierten Sicherheitsabstand beachten muss.

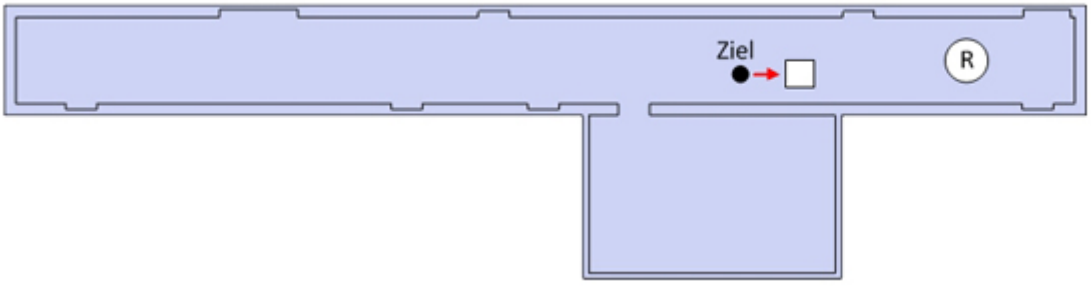


Abbildung 5.3: Versuchsaufbau des zweiten Szenarios: Zielanfahrt mit Hindernis.

Auswertung

Der Roboter hat einen Weg zum Ziel gefunden. Hierbei ist anzumerken, dass die Bewegungen des Roboters in den schmalen Bereichen nur sehr langsam ausgeführt werden, da der Trajektorienplaner darauf achten muss, genügend Abstand zu Hindernissen zu halten. Weicht der Roboter von der vorgegeben Trajektorie ab und kommt in einen kritischen Bereich, wo er einem Hindernis zu nahe kommt, so erkennt dies der Trajektorienplaner. Je nach Situation hält er den Roboter an, bis eine neue sichere Route aus dem kritischen Bereich heraus ermittelt wurde und die Fahrt zum Ziel wieder aufgenommen werden kann.

5.2.3 Bewegung der Roboter-Basis und des Roboter-Arms über den Aktions-Planer

Der Aktions-Planer wird beim Scitos für die Auswahl von durchzuführenden Aufgaben verwendet, die Bewegungen der Plattform und des Arms beinhalten. In diesem Szenario wird der Aktions-Planer verwendet um den Arm des Roboters zu bewegen und um eine Anfahrt von zufälligen Punkten im Raum durchzuführen. Der Aktions-Planer greift dabei auf verschiedene ROS-Module zu.

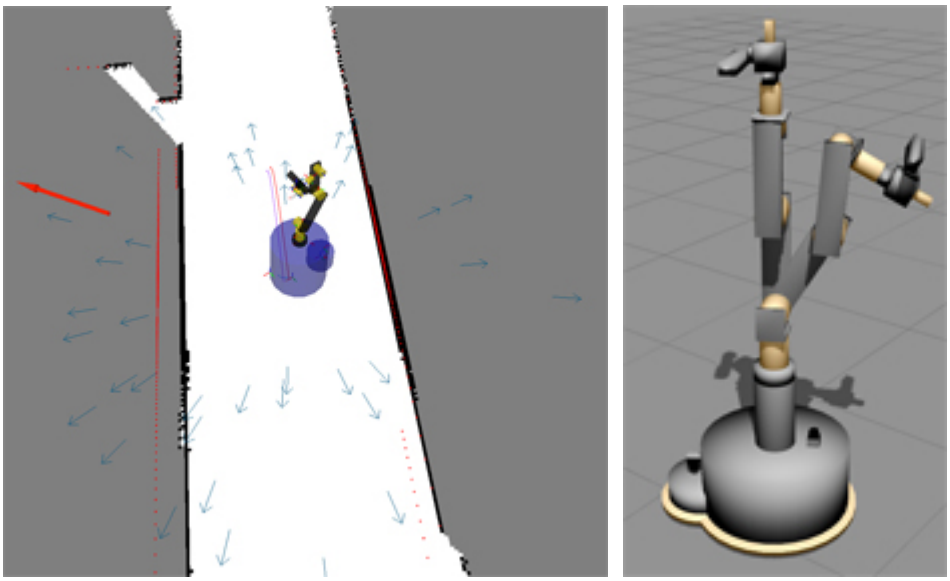
Das Ziel dieses Versuchs ist die Steuerung des Roboters über den Planer anhand von ausgewählten Aufgaben. Im speziellen der Bewegung des Arms und der zufälligen Bewegung im Raum. Durch diesen Test lässt sich überprüfen, ob die Steuerung des Arms sowie die entsprechenden ROS-Schnittstellen vollständig und korrekt implementiert worden sind. Im Zuge dieses Test werden keine Hindernisse verwendet, da ihr Vorhandensein für die Steuerung über den Planer unerheblich ist. Der Planer verwendet die gleichen ROS-Module zur Erstellung der Umgebungskarte und zur Berechnung der Trajektorie, deren Funktion bereits im vorherigen Versuch untersucht wurde.

Auswertung

Der Test war erfolgreich. Während des Tests hat der Roboter verschiedene Zielpositionen im Flur angefahren und der Arm wurde in verschiedene Positionen bewegt (vgl. Abbildung 5.4). Zielpositionen wurden zufällig gewählt, die Auswahl der Armpositionen erfolgte manuell.

5.3 Sensor-basierte Validierung

In diesem Abschnitt werden die simulierten Sensoren des Roboters auf ihre Funktionsfähigkeit untersucht. Die gelieferten Daten werden mit den Sensordaten des Scitos verglichen. Die Ausgabe der Sensordaten erfolgt über ROS-RViz. Als Versuchsobjekt für den Abgleich der



(a) Roboter mit Zufallszielen in RViz

(b) Roboter-Arm nach der Bewegungsausführung

Abbildung 5.4: Ergebnisse des dritten Szenarios: Verhaltens-basierte Validierung. Roboter mit zufälligen durch den Aktions-Planer generierten Bewegungszielen. Der Planer hat eines dieser Ziele ausgewählt (roter Pfeil) und überprüft im nächsten Schritt, ob eine Trajektorie zu diesem Punkt generiert werden kann. Ist dies nicht der Fall, wird ein neues Ziel bestimmt (a). Neue Armstellung nach Umsetzung der Zielvorgaben für die Armgelenke aus dem Aktions-Planer (b).

Laserscanner-Daten wird der Flur vor dem Robot Vision Lab verwendet. Die Analyse der Kinect-Kamera erfolgt in einer Ecke des Flurs unter Zuhilfenahme der Scitos-Transportbox. Die Aufnahmen der simulierten und der realen Bilder unterscheiden sich in Blickwinkel und Objektentfernung, da eine identische Positionierung des Roboter in der Realität und der Testumgebung nicht durchgeführt werden kann. Dies ist jedoch für eine Bewertung der gelieferten Werte nicht ausschlaggebend.

5.3.1 Vergleich der Laserscanner-Sensoren anhand einer erstellten Umgebungskarte

Zur Bewertung des simulierten Laserscanners wird der Flur vor dem Robot Vision Lab sowie das Modell des Flurs in der Simulation vom Roboter abgefahren und mit dem Laserscanner erfasst und ausgemessen (vgl. Abbildung 5.5). Aus den Daten wird anschließend durch ROS `gmapping` eine Umgebungskarte erstellt. Die Steuerung des Roboters erfolgt hierbei manuell.

Das Ziel dieses Versuchs ist die Bewertung der simulierten Laserscanner und ihrer Messdaten in Bezug auf Zuverlässigkeit und Genauigkeit anhand der erstellten Umgebungskarte.

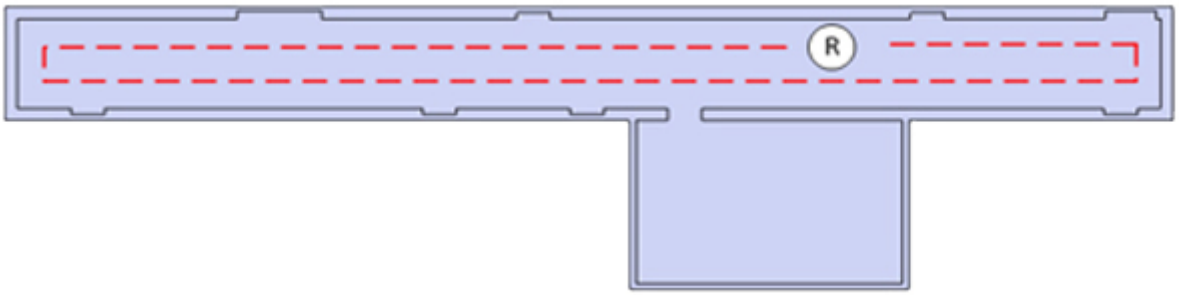


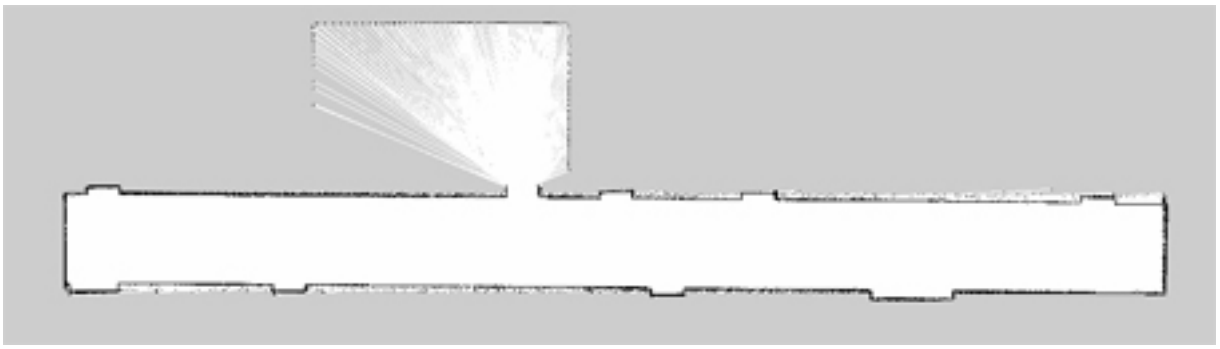
Abbildung 5.5: Versuchsaufbau des Szenarios: Die Erstellung einer Umgebungskarte.

Auswertung

Die Ergebnisse zeigen, dass die simulierten Laserscanner in Kombination mit `gmapping` sehr gute Ergebnisse erzielt haben (vgl. Abbildung 5.6). Die beiden Umgebungskarten sind sehr ähnlich in Bezug auf die Realität und weisen die gleichen Charakteristiken einer Laserscankarte auf.

Es müsste Untersucht werden, inwieweit die Daten durch verschieden Lichtverhältnisse beeinflusst werden oder wie diese in Gazebo behandelt werden.

(a) Erstellte Umgebungskarte anhand realer Laserscanner-Daten.



(b) Erstellte Umgebungskarte anhand simulierter Laserscanner-Daten.

Abbildung 5.6: Ergebnisse des Szenarios zur Validierung des simulierten Laserscanners. In Abbildung (a): Die Karte erstellt anhand der Laserscanner-Daten des Scitos. In Abbildung (b): Die Karte erstellt anhand der simulierten Laserscanner-Daten aus Gazebo.

5.3.2 Vergleich des Kinect-Kamera-Sensors anhand verschiedener Bilddaten

Dieser Versuch prüft die gelieferten Bilder der simulierten Kamera. Hierfür wird der Roboter vor einem Ende des Flures aufgestellt von wo aus er Fotos in Richtung Flurende aufnimmt. Durch diesen Aufbau wird sichergestellt, dass der taugliche Bereich der Kinect ausgenutzt wird. Um einen Darstellung der räumlicher Tiefe zu erreichen, wird außerdem die Transportbox des Scitos mit etwas Abstand vor der Wand platziert (vgl. Abbildung 5.7).

Das Ziel dieses Versuchs ist zum einen die Überprüfung, ob die beim Scitos verwendeten Daten für die Erstellung eines Farbbilds, der Punktewolke mit Tiefeninformationen und die Darstellung des Tiefenbildes als Octomap sowie der entsprechenden Schnittstellen in der Simulation bereitgestellt werden, und zum anderen ob der Abgleich gegen die Aufnahmen der Kinect-Kamera des Scitos Sinn ergibt.

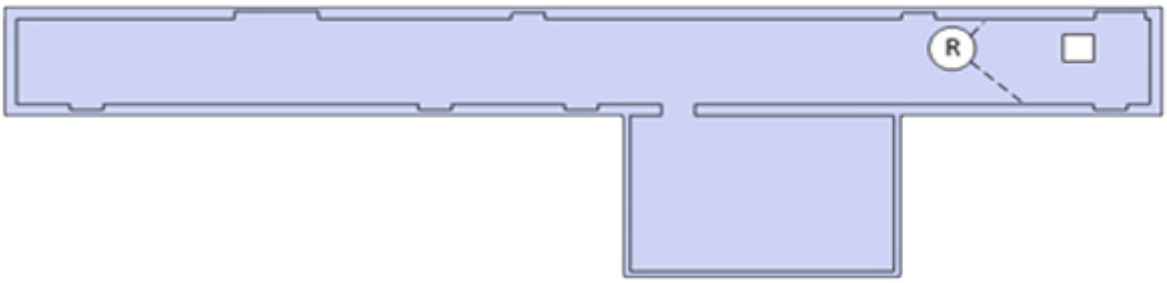
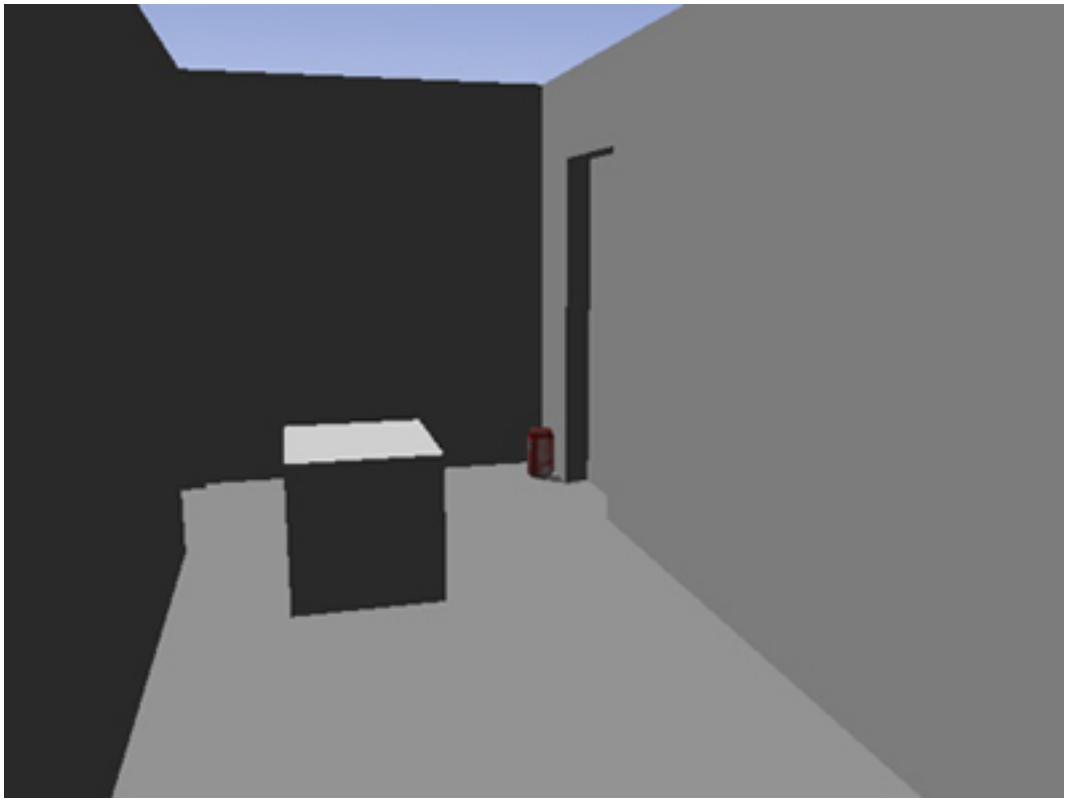


Abbildung 5.7: Versuchsaufbau des Szenarios: Roboter und Kamera-Ausrichtung zum Flurende.

Auswertung

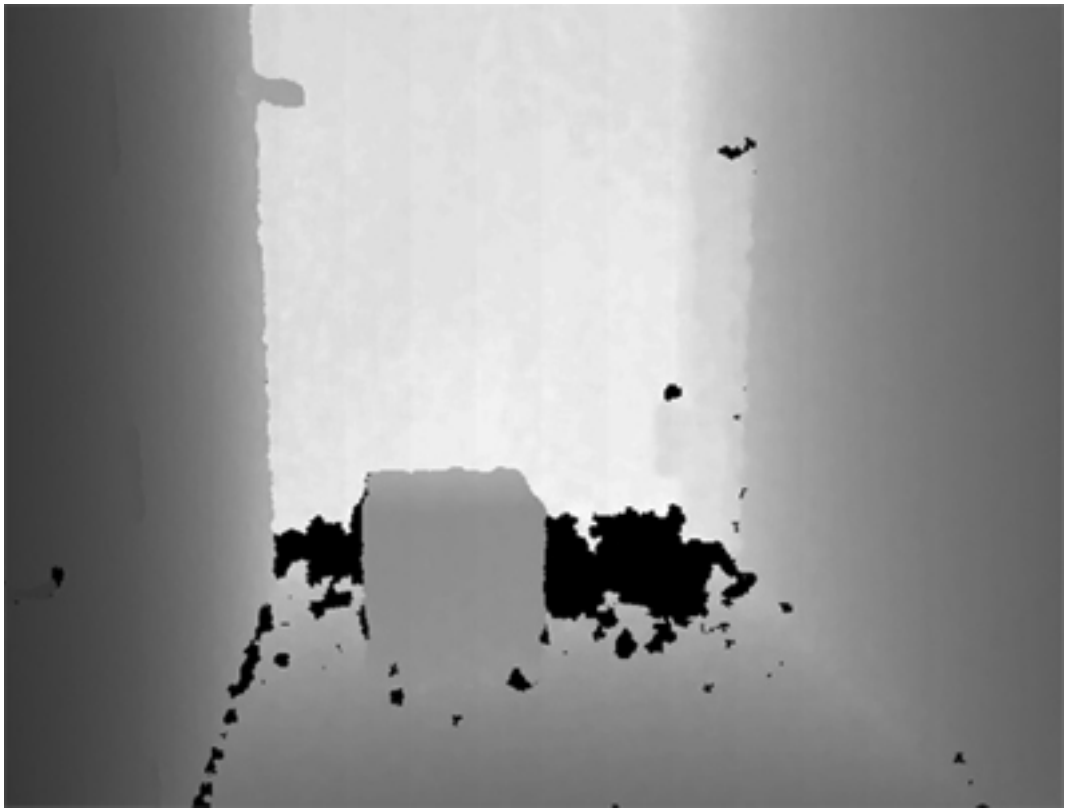
Die Ergebnisse in den Abbildungen 5.8, 5.8 und 5.9 zeigen, dass die gelieferten Sensorwerte mit denen des Scitos vergleichbar sind. Hierbei ist zu beachten, dass Ungenauigkeiten der realen Kamera und äußere Einflüsse wie die Lichtverhältnisse Auswirkungen auf die Aufnahmen haben. Das Farbbild unterscheidet sich vom simulierten Bild in sofern, als das der aufgenommene Flur in der Simulation deutlich weniger farbige Flächen aufweist (vgl. Abbildung 5.8). Beim Vergleich der Tiefenbilder des Scitos mit der Simulation ist zu erkennen, dass die Abbildung in der Simulation exakter und ohne große Ungenauigkeiten abgebildet wird (vgl. Abbildung 5.9). Die Unterscheidung in der Octomap ist u.a. auf die etwas unterschiedlichen Aufnahmewinkel zurückzuführen. Außerdem ist der Boden des realen Flurs nicht ganz plan (vgl. Abbildung 5.10).

(a) Farbbild der realen Kamera

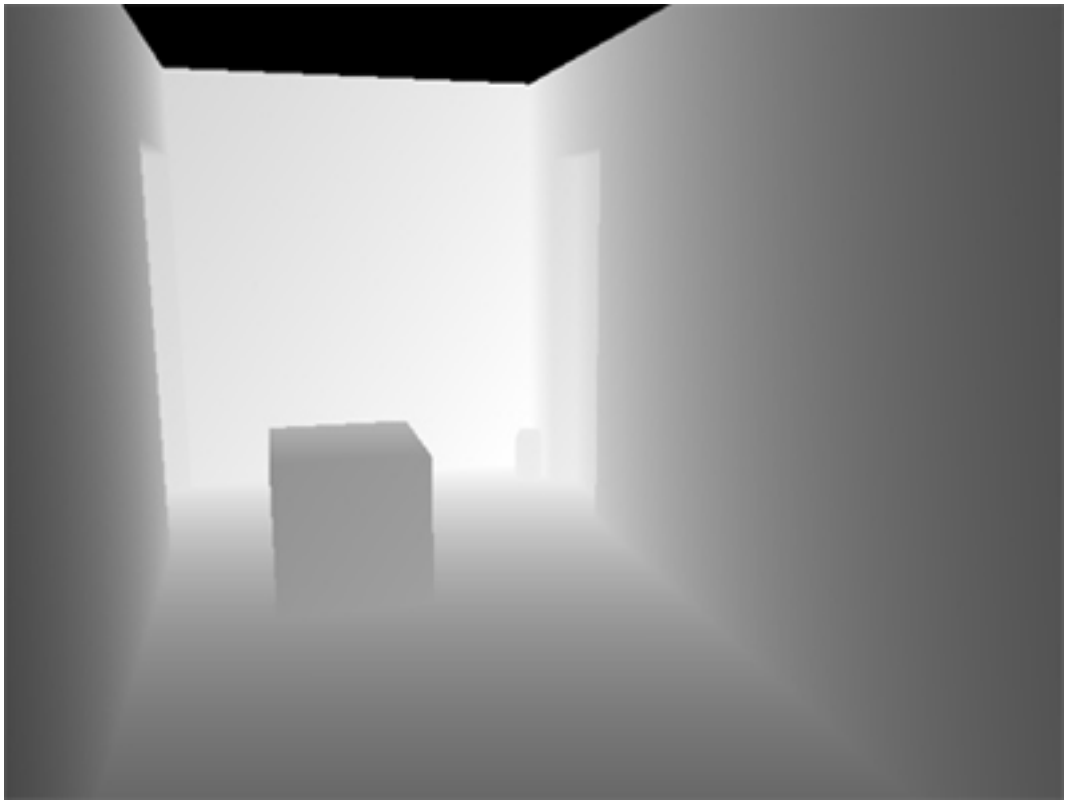


(b) Farbbild der simulierten Kamera

Abbildung 5.8: Ergebnisse der Sensor-basierten Validierung: Vergleich der Farbbilder.

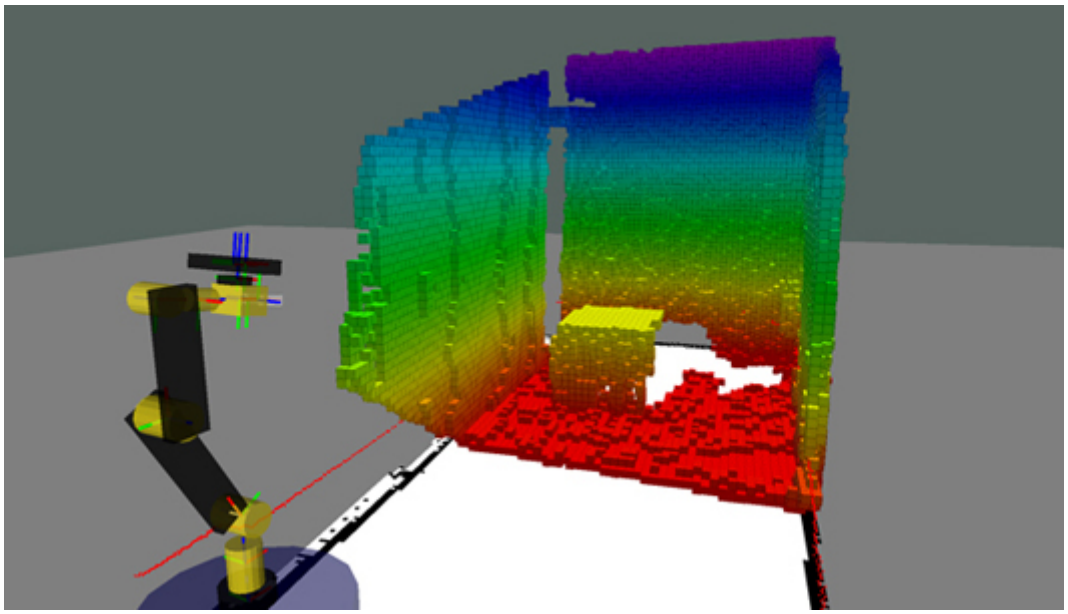


(a) Tiefenbild der realen Kamera

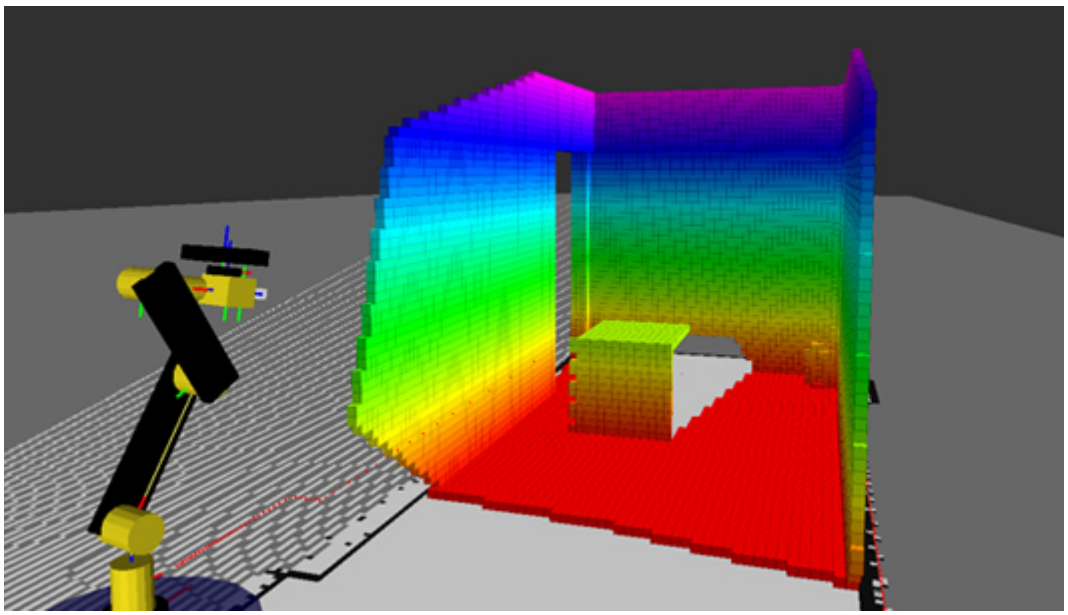


(b) Tiefenbild der simulierten Kamera

Abbildung 5.9: Ergebnisse der Sensor-basierten Validierung: Vergleich der Tiefenbilder.



(a) Oktomap der realen Kamera



(b) Oktomap der simulierten Kamera

Abbildung 5.10: Ergebnisse der Sensor-basierten Validierung: Vergleich der Octomap-Darstellung.

5.4 Szenario-übergreifende Zusammenfassung

Die Validierung hat ergeben, dass die benötigten Funktionen und Schnittstellen zur Steuerung des Roboters und der Generierung und Erfassung der Sensordaten erfolgreich implementiert worden sind. Die Ergebnisse der Verhaltens-basierten Validierung haben gezeigt, dass sich der simulierte Roboter ähnlich wie der reale Roboter Scitos verhält. Er kann sich erfolgreich in seiner Umgebung bewegen und der Arm kann in gewünschte Positionen gebracht werden. Die Auswertung der Sensor-basierten Validierung zeigt, dass die simulierten Sensoren sehr realitätsnahe Messdaten und Bilder generieren.

6 Zusammenfassung und Ausblick

Im Zuge dieser Arbeit wurde ein Simulationsmodell des Roboters Scitos G5 aus dem Robot Vision Lab der HAW Hamburg mit der Simulationssoftware Gazebo entwickelt, welches den Roboter mit seinen Aktoren und Sensoren sowie den bereits in ROS vorhandenen Bewegungs- und Steuerungs-Funktionen abbildet.

Die Realisierung der Robotersteuerung erfolgt über zwei zentrale Komponenten, von denen jeweils eine für die Steuerung der Plattform und eine für die Steuerung des Roboterarms zuständig ist. Die Implementierung der benötigten Schnittstellen für die Kommunikation mit dem ROS-Netzwerk und die Verwendung von ROS-Funktionen wird ebenfalls über diese Komponenten bereitgestellt. Ziel der Arbeit war hierbei nicht die Abbildung des Roboters in allen Einzelheiten, sondern vielmehr eines Modells, welches eine optische und funktionale Handhabung im Robot Vision Lab ermöglicht.

Die Abbildung der Aktoren, also die Gelenke der Roboterplattform und des Roboterarms, erfolgt durch den Einsatz von PID-Reglern, welche die Gelenke auf eine vorgegebene Geschwindigkeit beschleunigen bzw. Position bewegen und diese dort halten, bis weitere Steuerungsbefehle eintreffen.

Die Umsetzung der Sensoren des Roboters, wie Laserscanner oder Kinect-Kamera, wird über Gazebo-eigene Plugins realisiert, die diese Funktionen anhand von Informationen aus der verwendeten Grafik- und Physik-Engine abbilden. Die von den Plugins generierten Daten werden durch verschiedenen ROS-Komponenten verarbeitet und beispielsweise für die Erstellung von Umgebungskarten des Roboters oder bei der Erkennung von Hindernissen eingesetzt.

Um eine Bewertung der entwickelten Simulation vornehmen zu können und um zu entscheiden, ob diese für die weitere Entwicklung des Scitos verwendet werden kann, erfolgte eine Validierung der Simulation gegen den Scitos anhand verschiedener Szenarien und unter Einbeziehung der bereits verwendeten ROS-Komponenten des Scitos.

Zum einen wurde das Verhalten des Roboters während der Bewegungen der Plattform und des Arms untersucht und zum anderen die gelieferten Sensordaten der beiden Systeme verglichen. Das Verhalten des simulierten Roboters konnte gut nachverfolgt werden und entsprach hierbei weitestgehend dem des Scitos. Die simulierten Sensoren lieferten sehr gute Ergebnisse, die mitunter genauer waren als die des Scitos. Dies ist darauf zurückzuführen, dass die simulierten Sensoren keinen Ungenauigkeiten und äußeren Einflüssen unterliegen.

Manuelle Tests in der Phase der Validierung haben aufgezeigt, dass das erstellte Modell und die Steuerung des simulierten Roboters noch verbessert werden kann.

Das Setzen von Geschwindigkeiten und Gelenkwinkeln über die PID-Regler erfolgt verzögerungsfrei. So lange die Werteänderungen in kleinen Bereichen liegen, wie es bei der Steuerung über Trajektorien der Fall ist, macht sich dieses nicht bemerkbar. Erfolgt aber beispielsweise eine manuelle Vorgabe von Zielwerten die eine große Werteänderung bedeuten, kann sich dieses durch unkontrollierte Bewegungen des Roboters zeigen.

Ebenso wurden die Konfigurationsparameter für die PID-Regler so durch Versuche bestimmt, dass die Gelenke des Roboters bei der Fahrt und abruptem Anhalten ihre Position halten. Möglicherweise verfügen diese daher über zu viel Kraft. Eine gesteuerte Umsetzung solcher Eingaben könnte das Verhalten des Roboters weiter verbessern.

Wie bereits beschrieben wurden einige Bauteile des Scitos als rein visuelle Elemente modelliert. Deswegen haben diese kein Gewicht und folglich keinen Einfluss auf die Trägheit und das Verhalten des Roboterarms. Die Trägheitsmomente der übrigen modellierten Bauteile des Roboters sind mit Hilfe einer Einheitsmatrix bestimmt worden. Diese Lösung funktioniert für die erstellte Simulation, ist aber nicht optimal. Für eine genauere Abbildung Dynamik sollten alle Bauteile des Roboters als Links modelliert und die Trägheitsmomente individuell berechnet werden.

Die Entwicklung eines CAD-Modells des Roboters mit einer umfassenden Beschreibung der physischen Eigenschaften würde eine vollständigere Abbildung des Scitos in der Simulation ermöglichen. Dieses ist aber möglicherweise erst für fortgeschrittenere Aufgabenstellungen wie das Greifen von Objekten nötig.

Die Roadmap von Gazebo sieht für 2014 die Einführung von einfachen zerbrechbaren Gegenständen vor, womit sich möglicherweise die sich in Entwicklung befindlichen Drucksensoren für die Finger des Robotergreifers testen und überprüfen ließen.

Abschließend ist festzuhalten, dass die entstandene Simulation und die Möglichkeiten von Gazebo bei der Simulation von Sensoren und der Anbindung von ROS eine gute Grundlage für die Entwicklung weiterer Komponenten und Verhaltensansätze des Scitos im Robot Vision Lab bildet. In der Simulation entwickelte und getestete ROS-Komponenten können nach der Implementierung der entsprechenden Schnittstellen direkt auf dem realen Roboter verwendet werden.

Literaturverzeichnis

[N-238784 2013] WALZ, Jörg-Dieter (Hrsg.)Fraunhofer-Institut für Produktionstechnik und Automatisierung -IPA-, Stuttgart (Veranst.): *Fraunhofer-Institut für Produktionstechnik und Automatisierung. Jahresbericht 2012*. 2013. – URL http://publica.fraunhofer.de/documents/N-238784.html. – Abruf: 2013-11-04

[Colombo u. a. 2000] COLOMBO, Gery ; JOERG, Matthias ; SCHREIER, Reinhard ; DIETZ, Volker: Treadmill training of paraplegic patients using a robotic orthosis. In: *Journal of Rehabilitation Research and Development* 37 (2000), November, Nr. 6, S. 693–700

[G5 Manual 2010] METRALABS (Hrsg.): *SCITOS G5 Manual*. 2010

[Hägele 1994] HÄGELE, M. (Hrsg.): *Serviceroboter - ein Beitrag zur Innovation im Dienstleistungswesen : Band 1: Ergebnisse im Überblick. Band 2: Detailuntersuchungen*. 1994. – V, 63, T1.1–T5.5 S

[Hägele u. a. 2011] HÄGELE, Martin (Hrsg.) ; BENGEL, Matthias (Hrsg.) ; BLÜMLEIN, Nikolaus (Hrsg.) ; CONNETTE, Christian P. (Hrsg.) ; FISCHER, Jan (Hrsg.) ; GRAF, Birgit (Hrsg.) ; HÖPF, Michael (Hrsg.) ; JACOBS, Theo (Hrsg.) ; KLEINE, Oliver (Hrsg.) ; PFEIFFER, Kai (Hrsg.) ; WÖLTJE, Kay (Hrsg.) ; ROST, Arne (Hrsg.): *Wirtschaftlichkeitsanalysen neuartiger Servicerobotik - Anwendungen und ihre Bedeutung für die Robotik-Entwicklung : EFFIROB-Studie. Eine Analyse der Fraunhofer-Institute IPA und ISI im Auftrag des BMBF (Kennzeichen 01M09001) zwischen dem 1. Dezember 2009 und dem 30. November 2010*. URL http://publica.fraunhofer.de/documents/N-183304.html, 2011. – 371 S

[Hokuyo Spezifikationen 2013] HOKUYO SPEZIFIKATIONEN: *Hokuyo URG-04LX-UG01 Spezifikationen*. 2013. – URL http://www.roboparts.de/Hokuyo-URG-04LX-UG01/de. – Abruf: 2013-12-05

[International Federation of Robotics 2013a] INTERNATIONAL FEDERATION OF ROBOTICS: *World Robotics 2013 Industrial Robots*. 2013. – URL http://www.ifr.org/industrial-robots/statistics. – Abruf: 2013-11-04

[International Federation of Robotics 2013b] INTERNATIONAL FEDERATION OF ROBOTICS: *World Robotics 2013 Service Robots*. 2013. – URL http://www.ifr.org/service-robots/statistics/. – Abruf: 2013-11-04

[Kinect Spezifikationen 2013] KINECT SPEZIFIKATIONEN: *Kinect Spezifikationen*. 2013. – URL http://msdn.microsoft.com/en-us/library/jj131033.aspx. – Abruf: 2013-12-05

[Koenig und Howard 2004] KOENIG, N. ; HOWARD, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on* Bd. 3, 2004, S. 2149–2154 vol.3

[Kolbe 2013] KOLBE, Felix: *Goal Oriented Task Planning for Autonomous Service Robots*, Hamburg: Hochschule für Angewandte Wissenschaften, Dep. Informatik, Masterarbeit, November 2013

[Leuze Spezifikationen 2013] LEUZE SPEZIFIKATIONEN: *Leuze RS4-2E Spezifikationen*. 2013. – URL http://www.leuze.com/de/deutschland/produkte/produkte_fuer_die_arbeitssicherheit/optoelektronische_sicherheits_sensoren/sicherheits_laserscanner/sicherheits_laserscanner_funktionspaket__erweitert/selector.php?supplier_aid=520082&grp_id=A1-3-1-4-2&lang=deu. – Abruf: 2013-12-05

[Pires 2007] PIRES, J.N.: *Industrial Robots Programming: Building Applications for the Factories of the Future*. Springer, 2007. – URL http://books.google.de/books?id=oYTg9Kn-OU4C. – ISBN 9780387233260

[Pohlmann 2013] POHLMANN, Bernd: *Aktor-Sensor Simulation für Robotikanwendungen*, Hamburg: Hochschule für Angewandte Wissenschaften, Dep. Informatik, Masterarbeit, März 2013

[Reiser u. a. 2013] REISER, Ulrich ; JACOBS, Theo ; ARBEITER, Georg ; PARLITZ, Christopher ; DAUTENHAHN, Kerstin: *Care-O-bot® 3 - vision of a robot butler*. S. 97–116. In: *Your Virtual Butler. The Making-of*, URL http://dx.doi.org/10.1007/978-3-642-37346-6_9, 2013

[Schunk-Arm Manual 2010] SCHUNK GMBH & CO. KG (Hrsg.): *Schunk Roboterarm LWA 5DOF Montage- und Betriebsanleitung*. 2010

[Siciliano und Khatib 2008] SICILIANO, B. ; KHATIB, O.: *Springer Handbook of Robotics*. Springer, 2008 (Gale virtual reference library). – URL http://books.google.de/books?id=Xpgi5gSuBxsC. – ISBN 9783540239574

[Siddhartha u. a. 2010] SIDDHARTHA, Srinivasa ; DAVID, Ferguson ; CASEY, Helfrich ; DMITRY, Berenson ; ALVARO, Collet R. ; ROSEN, Diankov ; GARRATT, Gallagher ; GEOFFREY, Hollinger ; JAMES, Kuffner ; J MICHAEL, Vandeweghe: HERB: a home exploring robotic butler. In: *Autonomous Robots* 28 (2010), January, Nr. 1, S. 5–20

[Struss 2013] STRUSS, Ben: *3D-Kartierung von halbstatischen Indoor Umgebungen*, Hamburg: Hochschule für Angewandte Wissenschaften, Dep. Informatik, Ausarbeitung, Mai 2013

[Sun u. a. 2007] SUN, Loi-Wah ; VAN MEER, F. ; BAILLY, Y. ; YEUNG, Chung K.: Design and Development of a Da Vinci Surgical System Simulator. In: *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, 2007, S. 1050–1055

Tabellenverzeichnis

Abbildungsverzeichnis

A Antriebs- und Gelenkparameter des Roboters

Übersicht der Antriebs- und Gelenkparameter, die für die Erstellung der Simulation und die Konfiguration der verwendeten ROS-Komponenten benötigt werden.

A.1 Antriebsparameter der Roboterbasis

Antriebsparameter der Roboterbasis die bei der Erstellung des Simulationsmodells beachtet werden.

max. Geschwindigkeit translatorisch (m/s)	0,2
min. Geschwindigkeit translatorisch (m/s)	0,05
max. Geschwindigkeit rotatorisch (rad/s)	0,7
min. Geschwindigkeit rotatorisch (rad/s)	0,4
max. Beschleunigung in x-Richtung (m/s^2)	2,5
max. Beschleunigung in y-Richtung (m/s^2)	2,5
max. Beschleunigung rotatorisch (rad/s^2)	3,2

Tabelle A.1: Antriebsparameter der Roboterbasis

A.2 Gelenkparameter des Roboterarms

Gelenkparameter des Roboterarms die bei der Erstellung des Simulationsmodells beachtet werden.

Gelenkname	maximale Geschwindigkeit (rad/s)	maximale Beschleunigung (rad/s)	obere positive Gelenkauslenkung (rad)	untere negative Gelenkauslenkung (rad)
arm_joint_1	0.33	0.42	3.141593	-3.141593
arm_joint_2	0.33	0.42	2.024580	-2.024580
arm_joint_3	0.33	0.42	2.740166	-2.740166
arm_joint_4	0.33	0.42	2.286380	-2.373644
arm_joint_5	0.33	0.42	3.143321	-3.143321

Tabelle A.2: Gelenkparameter des Roboterarms

B ROS-Schnittstellenimplementierung

Vollständige Übersicht der implementierten ROS-Schnittstellen mit den benötigten Datentypen, den sendenden und empfangenden ROS-Nodes sowie ihre Implementierung in den Roboter-Plugins.

Topic /cmd_vel Datentyp: geometry_msgs/Twist

Sender: teleop_twist_keyboard, teleop_twist_keyboard_arm_cam, teleop_arm_controller, move_base

Empfänger: Roboter-Basis

Implementierung: Base-Plugin

Verarbeitende Funktion: cmdVelCallback()

- Empfang von Translations- und Rotationsgeschwindigkeiten in m/s für die Steuerung der Roboter-Basis.
- Umwandlung der Daten von m/s in rad/s
- Weiterleiten der Daten an die simulierte Regelung des Plattformantriebs

Topic /schunk/move_position Datentyp: metralabs_msgs/IDAndFloat

Sender: teleop_twist_keyboard_arm_cam

Empfänger: Roboter-Arm

Implementierung: Arm-Plugin

Verarbeitende Funktion: idAndFloatCallback()

- Empfang von Ziel-Positionen für die Armgelenke und einer Gelenk-ID
- Umwandlung der ID in einen Gelenknamen
- Weiterleiten der Daten an die simulierte Regelung des Armantriebs

Topic /schunk/move_velocity Datentyp: metralabs_msgs/IDAndFloat

Sender: teleop_arm_controller

Empfänger: Roboter-Arm

Implementierung: Arm-Plugin

Verarbeitende Funktion: Not implemented

- Empfang von Ziel-Geschwindigkeiten für die Armgelenke und einer einer Gelenk-ID
- Umwandlung der Gelenk-ID in einen Gelenknamen
- Weiterleiten der Daten an die simulierte Regelung des Armantriebs

Action /schunk/follow_joint_trajectory Datentyp: control_msgs/FollowJointTrajectoryGoal

Sender: WarehouseViewer

Empfänger: Roboter-Arm

Implementierung: Arm-Plugin

Verarbeitende Funktion: trajectoryActionCallback()

- Empfang einer Trajektorie mit Positions-, Geschwindigkeits- und Beschleunigungswerten für die Steuerung des Roboter-Arms
- Auslesen der Positionswerte eines Iterationsschritts der Trajektorie
- Weiterleiten der Daten an die simulierte Regelung des Armantriebs

Action /schunk/follow_joint_trajectory Datentyp: control_msgs::FollowJointTrajectoryFeedback

Sender: Roboter-Arm

Empfänger: Action-Client

Implementierung: Arm-Plugin

Verarbeitende Funktion: trajectoryActionCallback()

- Erstellen einer Feedback-Nachricht nach Durchführung eines Iterationsschritts der Trajektorie
- Senden des Feedbacks an den Action-Client

topic /scan Datentyp: sensor_msgs/LaserScan

Sender: Laserscanner vorne

Empfänger: slam_gmapping, move_base, RViz

Implementierung: Laserscanner-Plugin

Verarbeitende Funktion: Das Gazebo-Plugin des Laserscanners publiziert die Daten automatisch auf das konfigurierte Topic.

topic /scan_rear Datentyp: sensor_msgs/LaserScan

Sender: Laserscanner hinten

Empfänger: RViz

Implementierung: Laserscanner-Plugin

Verarbeitende Funktion: Das Gazebo-Plugin des Laserscanners publiziert die Daten automatisch auf das konfigurierte Topic.

topic /joint_states Datentyp: sensor_msgs/JointState

Sender: Roboter-Basis

Empfänger: state_publisher

Implementierung: Base-Plugin

Verarbeitende Funktion: publishJointStates()

- Erstellen einer JointState-Nachricht über die aktuellen Positionen und Geschwindigkeiten der Antriebsgelenke.
- Senden der Nachricht an den state_publisher.

topic /joint_states Datentyp: sensor_msgs/JointState

Sender: Roboter-Arm

Empfänger: state_publisher

Implementierung: Arm-Plugin

Verarbeitende Funktion: publishJointStates()

- Erstellen einer JointState-Nachricht über die aktuellen Positionen und Geschwindigkeiten der Armgelenke.
- Senden der Nachricht an den state_publisher.

topic /tf Datentyp: tf/tfMessage

Sender: state_publisher

Empfänger: tf

Implementierung: state_publisher Node

Verarbeitende Funktion: Der state_publisher empfängt joint_state-Nachrichten und erstellt daraus tf-Nachrichten, die er anschließend and /tf weiterleitet.

topic /tf Datentyp: tf/tfMessage

Sender: Laserscanner vorne

Empfänger: tf

Implementierung: Implementierung des static_transform_publisher Nodes in einem .launch-File.

Verarbeitende Funktion: Der static_transform_publisher sendet die absolute Position des vorderen Laserscanners zur Roboterbasis an /tf.

topic /tf Datentyp: tf/tfMessage

Sender: Laserscanner hinten

Empfänger: tf

Implementierung: Implementierung des static_transform_publisher Nodes in einem .launch-File.

Verarbeitende Funktion: Der static_transform_publisher sendet die absolute Position des hinteren Laserscanners zur Roboterbasis an /tf.

topic /odom Datentyp: nav_msgs/Odometry, geometry_msgs/TransformStamped

Sender: Roboter-Basis

Empfänger: odom, tf

Implementierung: Implementierung des static_transform_publisher Nodes in einem .launch-File.

Verarbeitende Funktion: publishOdometry()

- Erstellung von zwei Nachrichten mit den aktuellen Odometriedaten des Roboters und den dazugehörigen Transformdaten.
- Senden der Nachrichten an /odom und /tf.

topic /bumper_base Datentyp: gazebo_msgs/ContactsState

Sender: gazebo_ros_bumper plugin

Empfänger: Roboter-Basis

Implementierung: Base-Plugin

Verarbeitende Funktion: BumperCallback()

- Empfange von Bumper-Nachrichten
- Überprüfung auf vorliegende Kollisionen
- Stoppen der Antriebsgelenke

topic /bumper_base Datentyp: gazebo_msgs/ContactsState

Sender: gazebo_ros_bumper plugin

Empfänger: Roboter-Basis

Implementierung: Base-Plugin

Verarbeitende Funktion: BumperCallback()

- Empfange von Bumper-Nachrichten
- Überprüfung auf vorliegende Kollisionen
- Stoppen der Antriebsgelenke

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 06.12.2013
Ort, Datum

Unterschrift