Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Cagri Erdogan

Steuerung eines mobilen acht Achsen Roboters unter ROS

Fakultät Technik und Informatik *Faculty of Engineering and Computer Science*

Cagri Erdogan

Steuerung eines mobilen acht Achsen Roboters unter ROS

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Mechatronik
an der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Frischgesell
Zweitgutachterin: Dipl.Ing. Carolina Bohnert

Abgegeben am 28. Juni 2018

Cagri Erdogan

Thema der Bachelorarbeitarbeit

Steuerung eines mobilen acht Achsen Roboters unter ROS

Stichworte

ROS, KUKA, youBot, Matlab, Simulink, Robotik, Steuerung, Raspberry Pi, Odroid, Ubuntu, direkte Kinematik, inverse Kinematik, Jacobimatrix

Kurzzusammenfassung

Ziel dieser Arbeit ist es, den youBot Roboter des Herstellers KUKA so umzubauen, dass er statt durch den werksseitig installierten eingebetteten Computer von einem Raspberry Pi Einplatinencomputer gesteuert wird. Zum Steuern des Roboters in der **realen Umgebung** wird das Robot Operating System (ROS) verwendet und auf dem Raspberry Pi (oder alternativ dem Odroid) installiert. ROS bietet die Möglichkeit, Sensordaten auszuwerten, Aktoren anzusteuern, sowie eine Regelung zu implementieren. Das Matrixberechnungsprogramm MATLAB berechnet die Gelenkwinkel des Roboters. Die berechneten Daten werden an ROS geschickt. Für die Entwicklung der Roboterregelung wird ein Modell des Roboters in der **Simulationsumgebung** Simulink erstellt.

Cagri Erdogan

Title of the paper

The Controlling of eight axis mobile robot under ROS

Keywords

ROS, KUKA, youBot, Matlab, Simulink, Robotik, Steuerung, Raspberry Pi, Odroid, Ubuntu, Inverse Kinematic, Direct Kinematic, Jacobian Matrix

Abstract

The goal of this paper is to describe a modification to the youBot robot manufactured by KUKA - that is, to replace the factoy installed embedded computer with a Raspberry Pi single board computer. The Robot Operating System (ROS) which will be installed on the Raspberry Pi (alternatively Odroid) will be used to control the robot in the real-world enviroment. ROS provides capabilities of control and regulation of actuators and as well as evaluation of sensor data. The matrix calculation software MATLAB will be used to calculate the joint angles of the robot. The calculated data will be send to ROS. For the development of the robot controller software a simulation model of the robot will be developed in Simulink.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ADC	Analog-Digital-Umsetzer
API	Application Programming Interface
CNC	Computer Numerical Control
DIY	Do It Yourself
DK	direkte Kinematik
DOF	Degrees of Freedom
E/A	Eingänge und Ausgänge
FuE	Forschung und Entwicklung
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IK	inverse Kinematik
IRS	Infrarotsensor
KOS	Koordinatensystem
LAN	Local Area Network
PWM	Puls-width modulation
Raspi	Raspberry Pi
RCHH	RobOtter Club Hamburg
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping ; Simultane Lokalisierung und Kartenerstellung
SoC	System-on-a-Chip

SPI	Serial Peripheral Interface
SSH	Secure Shell
SW	SolidWorks
TCP	Tool Center Point ; Endeffektor
UART	Universal Asynchronous Receiver Transmitter
uC	Mikrocontroller
URDF	Unified Robot Description Format
US	Ultraschallsensor
USB	Universal Serial Bus
v-rep	virtual robot experimentation platform
XML	Extensible Markup Language

1. Einleitung

Mit dem Wort Roboter wird oft ein technisches Gerät assoziiert, das dazu dient, dem Menschen mechanische Arbeit abzunehmen. Der Ursprung des Wortes "Roboter" kommt vom tschechischen Wort "robota" und kann als "Zwangsarbeit" übersetzt werden. Roboter können sowohl stationäre als auch mobile Maschinen sein und werden durch Software gesteuert.
Im Jahr 1954 wurde in den USA ein Patent für einen programmierbaren Manipulator [43] angemeldet. Dieses Datum gilt als Beginn für die Entwicklung von Manipulatoren bzw. Roboterarmen. Stammt der erste mobile Roboter noch aus dem Jahr 1968, hat die Robotik seitdem und besonders in den letzten Jahren bedeutende Fortschritte gemacht. Diese Entwicklungsschübe entstanden vor allem dadurch, dass Menschen nach schnelleren und autonomeren Lösungen für tägliche Probleme suchten. Ein mobiler Manipulator ist ein Robotersystem, das aus einer selbstfahrenden Basis und einem Roboterarm besteht. Die autonome mobile Plattform kann sich in ihrer Umgebung selbständig bewegen und interagieren. Der Manipulator ist ein Roboterarm mit mehreren Gelenken und wird oft mit einem Industrieroboter assoziiert. Ein Roboterarm übernimmt oft Präzisionsarbeit (z.B. Schweißroboter), Fließbandarbeit (z.B. Teile sortieren) oder Montagearbeiten (z.B. Automobilbranche). Das System des mobilen Manipulators hat den Vorteil, dass die Komponenten des Systems entweder separat oder gemeinsam angesteuert werden können. Dadurch lassen sich in der Entwicklung und Instandhaltung die Teilprobleme auf einzelne Komponenten herunter brechen und vereinfachen so das Verständnis des Systems. Die Innovationen im Bereich der Robotik haben die Bereiche Hardware (Elektronik, Mechanik) und Software beeinflusst. Dadurch sind bessere und leistungsstärkere mechanische und elektronische Komponenten entstanden, die heute in der Robotik verwendet werden. Genauso wurden spezielle Softwarelösungen entwickelt, die das Ansteuern von Robotern erleichtern und den Entwicklungsaufwand verkürzen.

1.1. Motivation

Die Motivation dieser Bachelorarbeit ist es, den mobilen Manipulator youBot der Firma KUKA umzubauen. Die Idee hinter dem Umbau ist es, die vorhandene Steuerplatine des Roboters durch einen Einplatinenrechners [39] wie den Raspberry Pi [25] oder Odroid [11] zu ersetzen, was mehrere Vorteile gegenüber herkömmlichen Computern bietet: Geringerer

Stromverbrauch, günstigere Komponenten, kleinere Bauweise und besonders die vielfältigen Möglichkeiten eines open-source Mikro-Computers selbst, insbesondere

- zahlreiche standardisierte Schnittstellen; z.B. für weitere Sensoren wie Taster, Ultraschall, Infrarot, Laser Scanner, USB Camera, Bewegungssensoren wie die Microsoft Kinect oder Asus Xtion
- Anwendung in der Robotikprojekte
- Kompatibilität mit anderen populären open-source Mikrocontrollern (z.B. Arduino) und
- eine große online community, die zahlreiche Softwarebibliotheken frei zugänglich macht und als Forum für Problemlösungen dient.

Diese reiche Infrastrukturlandschaft um open-source Mikrocontroller eröffnet die Möglichkeit für zahlreiche weitere Modifikationen, die das Anwendungsgebiet des Roboters erweitern können. Dazu soll der Raspberry Pi [25] und/oder Odroid [11] mit einem Steuerungsfirmware Robot Operating System (ROS) verwendet werden. Der KUKA youBot ist ein mobiler Roboter, der für Forschungszwecke eingesetzt wird. Die Plattform bietet Möglichkeit, zwei Manipulatoren parallel zu betreiben. Um Kollisionen zwei zeitgleich betriebener Manipulatoren zu vermeiden, kann eine Softwarelösung verwendet werden. Mit eingebauten Sensoren können die Objekte in der Umgebung zu erkennen und aufzunehmen, Kollisionen bei der Navigation zu vermeiden oder eine virtuelle Karte der Umgebung erstellen.

Eine weitere Motivation dieser Arbeit ist es, die erworbenen ROS-Kenntnisse im **RobOtter Club Hamburg (RCHH)** anzuwenden. Der RCHH ist ein Robotik-Team an der HAW Hamburg, das aus Studenten besteht, in dem "Eurobot" Roboter Wettbewerb mitmacht. Im RCHH kann die erlernte Theorie aus verschiedenen Disziplinen, wie der Elektrotechnik, Informatik und Maschinenbau in die Tat umgesetzt und praktisch angewendet werden. Mit dieser Wettkampfidee kann der youBot in einige Wettkämpfen wie z.B. dem RoboCupIndustrial [30] oder RoboCup@Work [29] teilnehmen.

Der Einsatz innovativer mobiler Roboter, die mit fortschrittlichen Manipulatoren für aktuelle und zukünftige industrielle Anwendungen ausgestattet sind, ermöglicht, an Orten an denen die Roboter mit Menschen zusammenarbeiten, komplexe Aufgaben auf Maschinen zu übertragen.Dieses gilt für die Fertigung, Automatisierung und der Teilehandhabung bis hin zur allgemeinen Logistik.

2. Hardware

In der Abbildung 2.1 ist ein KUKA youBot mit seiner mobilen Plattform und einem Roboterarm abgebildet. Der KUKA youBot besteht aus einem omnidirektional verfahrbahren Chassis und einem oder zwei Roboterarmen, die auf dem Chassis montiert werden. Der Manipulator

Abbildung 2.1.: KUKA youBot [14]

hat fünf Gelenke. Jedes Gelenk hat einen Motor mit eingebautem Getriebe. Alle Gelenke haben einen mechanischen Endschalter. Jede Gelenkposition wird von einem Encoder bzw. Drehgeber gemessen. Über ein Steuerungsprogramm kann die Positionen von den Gelenken abgefragt werden. Der Roboterarm besitzt einen Greifer mit zwei Fingern. Der Greifer lässt sich öffnen und schließen. Die Fingerposition wird nicht von einem Encoder gemessen. Während der Initialisierung werden die Greiferfinger vollständig geöffnet und geschlossen.

2.1. youBot Base

Die youBot Basis ist eine omnidirektionale mobile Plattform mit vier Mecanum Rädern. Das sind spezielle Räder, deren Lauffläche aus im 45°-Winkel über den Radumfang verteilten Rollen besteht, wodurch der youBot entlang X und Y Achse translatorische und um die z-Achse rotatorische Bewegung beliebig überlagern kann. So werden omnidirektionale Bewegungen einschließlich Seitwärts- und Diagonalfahrten möglich. Die Abbildung 2.2 veranschaulicht das virtuelle Koordinatensystem, das während der Arbeit mit dem Roboter verwendet wird. Das Koordinatensystem befindet sich in der Mitte der Plattform.

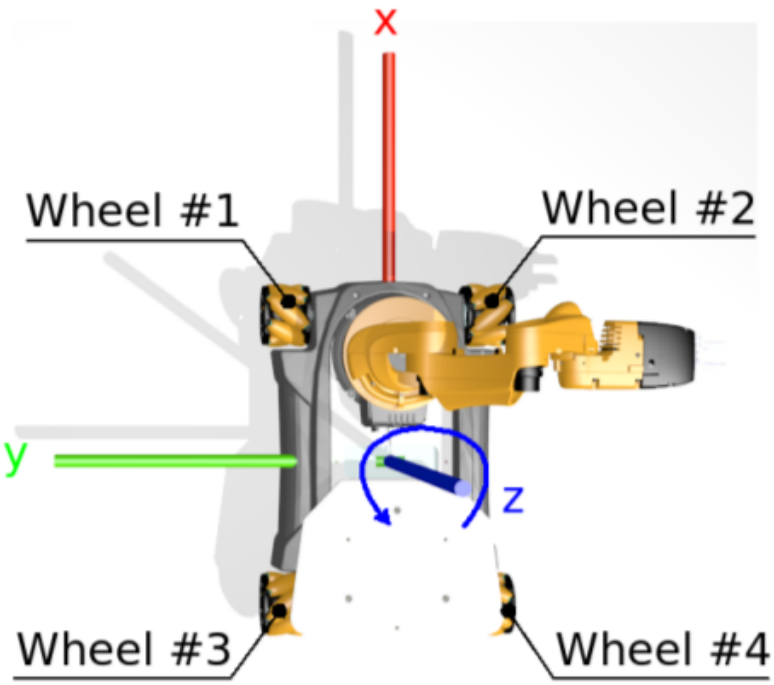

Abbildung 2.2.: KUKA youBot Base Koordinatensystem [16]

Positive Werte für die Drehung um die z-Achse führen zu einer Bewegung gegen den Uhrzeigersinn, wie durch den blauen Pfeil angezeigt. Jedes Rad kann separat angesteuert werden. Die Ansteuerung der Motoren erfolgt über Ethernet und/oder EtherCAT. Die Steuerung der Plattform kann über einem Ethernet-Kabel entweder an den Bordrechner oder an einen externen Computer angeschlossen werden.

2.2. youBot Arm

Die Abbildung 2.3 veranschaulicht die Dimensionen des youBot Roboterarms. Der youBot Arm ist eine serielle Kette von Drehgelenken. Der Endeffektor ist ein Zwei-Finger-Greifer, der tauschbar ist. Jedes Gelenk kann separat über Ethernet und/oder EtherCAT angesteuert werden. Die Abbildung 2.4 zeigt ebenfalls die Abmessungen der einzelnen Gelenke des

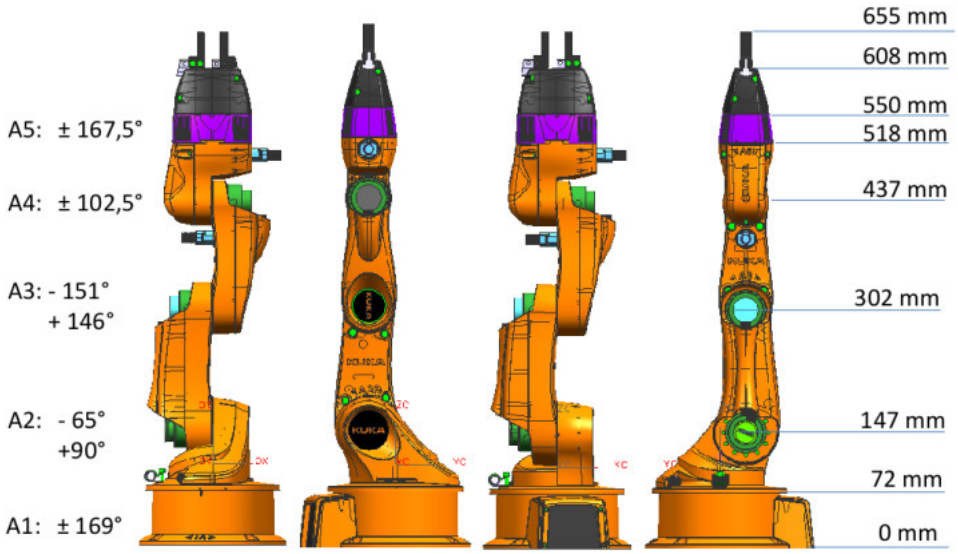

Abbildung 2.3.: KUKA youBot Armabmessungen [16]

Arms und gibt Auskunft darüber, welche Achse wie weit ausgelenkt werden kann. Zudem wird der Arbeitsraum zweidimensional in der Seitenansicht dargestellt.

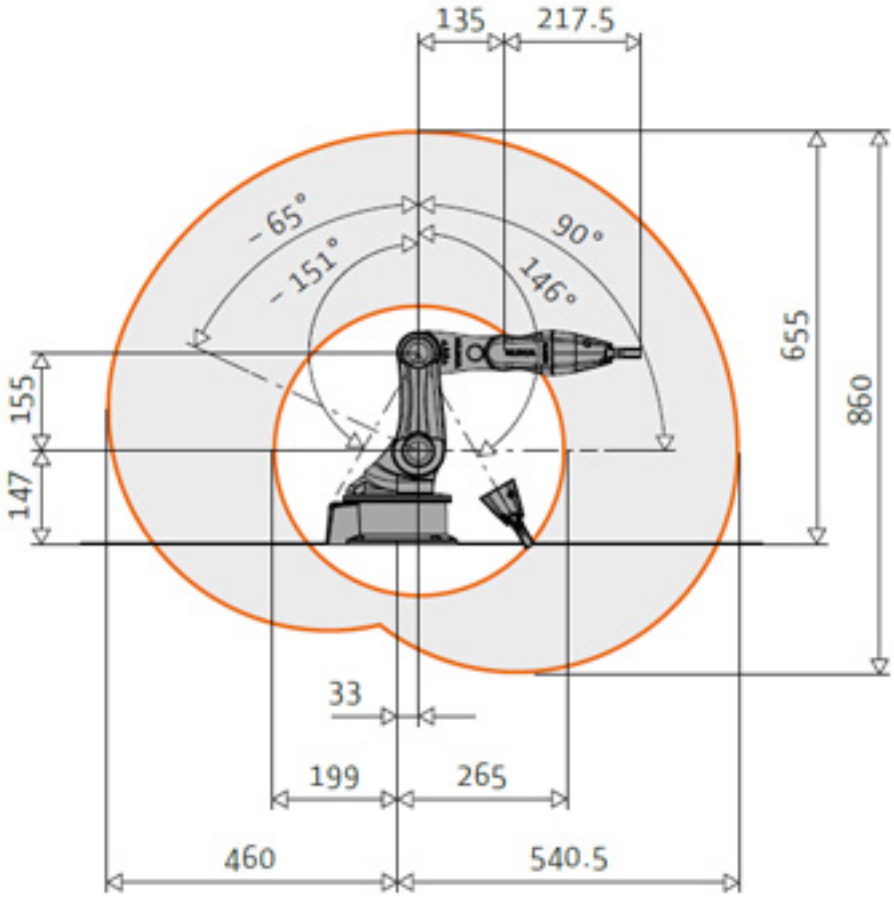

Abbildung 2.4.: KUKA youBot Arm Arbeitsbereich [6]

2.3. Optionen für Mikrocontroller

Die Modifikation im Rahmen dieser Arbeit sieht es vor, den werksseitig verbauten Embedded-PC durch einen Einplatinenrechner zu ersetzen. Im folgenden sollen dafür zwei Alternativen vorgestellt werden, die in die engere Wahl fielen. An dieser Stelle soll darauf hingewiesen werden, dass für Versuchsaufbau die Entscheidung zu Gunsten des Raspi gefällt wurde. Obwohl der Odroid zum Zeitpunkt des Projekt leistungsfähiger ist, bietet der Raspi mit seiner überlegenen online-community breitere Entwicklungsressourcen.

2.3.1. Raspberry Pi

Der Raspberry Pi ist ein beliebter und auf dem Markt weit verbreiteter Einplatinenrechner. Sein Anwendungsgebiet ist weitläufig und umfasst die Robotik, Hausautomation, Industrie, Computer Numerical Control (CNC) Steuerungen und insbesondere Do It Yourself (DIY) / Heimwerker Projekte. Der Raspberry Pi hat ein System-on-a-Chip ARM-Prozessor, Arbeitsspeicher und GPU. Der Raspi bietet zahlreiche Anschlüsse für externe Hardware: Ethernet, Sound, mehrere USB, HDMI, Camera, LCD und GPIO-Pins.

Abbildung 2.5.: Raspberry Pi 3 Model B [26]

Die Vorteile zu einem dem werksseitig installierten Embedded-PC sind:

- Kleine Bauform
- Kostengünstig
- 5V Stromversorgung
- Mikro-SD-Karte
- Linux und Win 10 als Betriebssystem
- Konfigurierbar
- Open-Source
- Portable

Mit Erweiterungsplatinen kann der Funktionsumfang und damit das Einsatzgebiet des Raspi erweitert werden. Das Hauptmerkmal der Raspi-Boards sind die dedizierten 40 GPIO-Pins. Diese Pins haben folgende Funktionen:

- Inter-Integrated Circuit (I2C)
- Universal Asynchronous Receiver Transmitter (UART)
- Serial Peripheral Interface (SPI)
- Puls-width modulation (PWM)

2.3.2. Odroid

Odroid ist ein Einplatinenrechner auf dem Markt. Er ist Konkurrenz gegenüber Raspberry Pi. Vom Aufbau, Eigenschaften und Pinbelegung ist dieser dem Raspberry Pi sehr ähnlich.

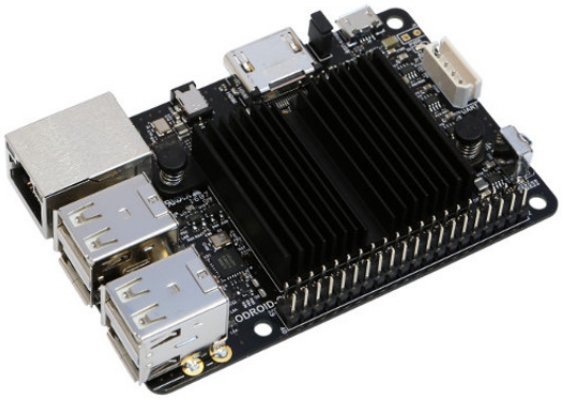

Abbildung 2.6.: Odroid C2 [10]

Beide Einplatinenrechner, der Odroid und der Raspberry Pi, haben ungefähr die gleichen Eigenschaften. Im Gegensatz zu dem Raspi ist der Odroid jedoch leistungsstärker. Dieser bietet einen schnelleren Prozessor, einen größeren Arbeitsspeicher, einige GPIOs sind unterschiedlich und er besitzt einen eMMC Kartenanschluss. Preislich liegen beide im gleichen Umfeld. Ein weiteres Merkmal des Odroid ist, dass dieser die Möglichkeiten bietet, diesen mit verschiedenen Aufsteckplatine zu erweitern. Im Gegensatz dazu besitzt der Raspberry Pi eine größere Gemeinschaft, wodurch eine große Anzahl an öffentlichen Projekten existiert, die den Einstieg für Anfänger erleichtern.

2.4. Sensoren

Die Sensorik spielt in der mobilen Robotik eine große Rolle. Sensoren helfen Maschinen dabei, mit ihrer Umgebung zu interagieren. Sie können einem Roboter ermöglichen, seine Position ermitteln oder Umgebungskarten zu erstellen (Simultaneous Localization and Mapping (SLAM)), Objekte zu erkennen, ihnen zu folgen oder ihnen auszuweichen. In dieser Arbeit werden Sensoren, deren Daten und ihre Auswertung nur eine kleine Rolle spielen. Das soll einerseits den Projektumfang auf ein zu bewältigendes Maß beschränken. Außerdem werden leider bei weitem nicht alle Sensoren von ROS unterstützt (vergleiche diese Liste von unterstützten Sensoren: http://wiki.ros.org/Sensors) und so ist es auch mit den im youBot von KUKA verbauten.

3. Software

In diesem Abschnitt wird die in dieser Arbeit verwendete Software näher erläutert.

3.1. Robot Operating System (ROS)

Die Entwicklung eines neuen Roboters bedeutet oft einen hohen Aufwand. Das betrifft besonders die Integration der elektronischen Hardwarekomponenten und der Software. Für den neu entwickelten Roboter werden entweder standardisierte Hardwarekomponenten verwendet, sofern diese auf dem Markt erhältlich sind, oder wenn notwendig, auch neu entwickelte Komponenten. Um Wartungs- und Ersatzteilkosten gering zu halten und von verfügbaren Upgrades einzelner Komponenten zu profitieren, ist es sinnvoll, auf eine Software zu setzen, die die Entwicklung unter Wiederverwendung bereitstehender Hardware- und Softwarekomponenten zulässt. Ein solches System verspricht einen schnelleren Fortschritt bei der Entwicklung von neuen Robotern und seiner Anwendungen.

Diesen Ansatz verfolgt das Robot Operating System **ROS** [33] , eine Open-Source Software-Framework für die Roboterentwicklung unter Ubuntu Linux. Es bietet viele Möglichkeiten, verschiedene Roboter und deren Komponenten zu steuern. ROS ist ein Open-Source-Meta-Betriebssystem für Roboter, welches die Entwicklung von Robotikanwendungen erleichtert. Es bietet verschiedene Dienste eines Betriebssystems an, einschließlich der Hardware-Abstraktion [40], Low-Level-Gerätesteuerung, Implementierung von häufig genutzten Funktionen, Gerätetreiber, Visualisierer, Nachrichtenübergabe zwischen Prozessen und Paketverwaltung und viele weitere Funktionen.

Das Hauptziel von ROS ist es, eine die Wiederverwendung des Quellcodes in der Robotikforschung und -entwicklung zu ermöglichen. Um die Kompatibilität mit verschiedener Hardware zu gewährleisten, bietet die ROS Bibliothek Systemfunktionen auf einer von der Hardware durch Abstraktion getrennten höheren Ebene an, während die Ansteuerung der Hardwarekomponenten wie der Sensoren und Aktoren durch Gerätetreiber realisiert ist. Durch diese Abstraktion kann ist es möglich, ROS auf weit verbreiteter Open-Source-Hardware wie dem Arduino oder dem Raspberry Pi einzusetzen - unter der besagten Bedingung, dass sie Ubuntu Linux unterstützt.

Die folgenden ROS Algorithmen können für allgemeine Aufgaben verwendet werden.

- Navigation
- Bahnplanung (Motion planning)
- Kartierung
- Hinderniserkennung
- Erstellung von 3D Modellen und die Verwendung in Simulatoren
- Objekt- und Gesichtserkennung mittels Kameras
- 3D-Objekterkennung
- Autonomes Navigieren
- Simultane Lokalisierung und Kartenerstellung (Simultaneous Localization and Mapping SLAM)
- Aktionsplanung
- Motion Control für mehr gelenkige Arme
- Maschinelles Lernen (Maschine Learning)
- Spracherkennung
- Steuerung von Aktoren und Auslesen von Sensoren

Das ROS System kann low-level Hardwaresteuerung und high-level Verarbeitung voneinander trennen und sie an separate Programme delegieren. Durch diese Separation ist es möglich, ein low-level Programm (und die entsprechende Hardware) durch einen Simulator zu ersetzen, um das Verhalten des Systems zu testen. ROS bietet komfortable Mechanismen für das Speichern und Auslesen von Sensordaten und anderen Arten von Nachrichten. Dadurch lässt sich in der Entwicklung viel Zeit sparen, da bei einem Versuch mit einem physikalischen Roboter aufgenommene Sensordaten wiederverwendet werden können, um verschiedene Verarbeitungswege zu testen - etwas, dass sonst nur durch eine Wiederholung des Versuchs möglich ist. Die gespeicherten Daten werden im ROS in Datenstrukturen gespeichert, die als **bags** ("Taschen") oder auch als **rosbags** bezeichnet werden. Sie sind sowohl schreib- als auch lesbar.

Der entscheidende Punkt für die oben genannten Eigenschaften ist flüssige Verarbeitung der Daten. Das heißt, der reelle Roboter, Simulator und rosbag (Abspielmechanismus) verwenden eine sehr ähnliche Schnittstelle und sind somit kompatibel mit einander. ROS ist nicht nur eine Plattform. Unter anderem ist ROS ein verbreiteter Support in der gesamten Robotergemeinschaft. In der Zukunft wird ROS kontinuierlich entwickelt, erweitert und verbessert.

3.2. ROS Installation

Im folgenden wird die Installation von ROS gemäß des Versuchsaufbaus auf Ubuntu 16.04 LTS für Raspberry Pi (download unter https://wiki.ubuntu.com/ARM/RaspberryPi) beschreiben. Hier wie mit jedem anderen vergleichbaren Einplatinenrechner muss darauf geachtet werden, ein sowohl mit der Hardware als auch mit der ROS Version kompatibles Betriebssystem zu verwenden.

Aktuell ist ROS nur unter Unix-basierten Plattformen kompatibel. Das empfohlene Betriebssystem ist hierbei Ubuntu. In dieser Arbeit wird die ROS Distribution **ROS-Kinetic** [34] eingesetzt. Diese ROS Version ist nur mit Ubuntu 16.04 kompatibel. Das Installationspaket trägt den Namen **ros-kinetic-desktop-full** [32]. Der Installationsvorgang läuft wie folgt ab und wird über die Kommandozeile ausgeführt.

1. **Für Ubuntu 16.04 Xenial Xerus**
   sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"> /etc/apt/sources.list.d/ros-latest.list'
2. **Schlüssel einrichten**
   sudo apt-key adv –keyserver hkp://ha.pool.sks-keyservers.net:80 –recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
3. **Ubuntu Aktualisieren**
   sudo apt-get update
4. **ROS Indigo Installation**
   sudo apt-get install ros-kinetic-desktop-full
5. **rosdep initialisieren**
   sudo rosdep init
   rosdep update
6. **ROS Umgebungsvariablen einrichten**
   echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
   source /.bashrc
7. **rosinstall**
   sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential
8. **catkin workspace erstellen und initialisieren**
   mkdir -p ~/catkin_ws/src
   cd catkin_ws/src
   catkin_init_workspace

```
cd /catkin_ws/
catkin_make
```

9. **catkin_ws in der ROS-Umgebung hinzufügen**
```
source /catkin_ws/devel/setup.bash
echo "source /catkin_ws/devel/setup.bash" >> ~/.bashrc
```

10. **ROS Umgebungsvariablen überprüfen**
```
export | grep ROS
```

3.3. MATLAB

MATLAB ist eine Softwarepaket zur Lösung numerischer Probleme und der Visualisierung von berechneten Daten und wird in diesem Projekt dazu dienen, die kinematischen Matrizen für die korrekte Steuerung des Roboterarms zu berechnen. Durch sog. Toolboxen kann der Funktionsumfang erweitert werden. Der Name MATLAB ist von **MAT**rix **LAB**oratory abgeleitet. Das Anwendungsgebiet von MATLAB umfasst unter anderem Datenanalyse, Bildverarbeitung, Auswertung von Sensoren und Ansteuerung von Aktoren, Flugsimulationen, Mikrocontroller Programmierung sowie Robotik. Seit 2015 gehört die ROS [20] Toolbox dazu und ist unentgeltlich verfügbar.

3.4. ROS und MATLAB

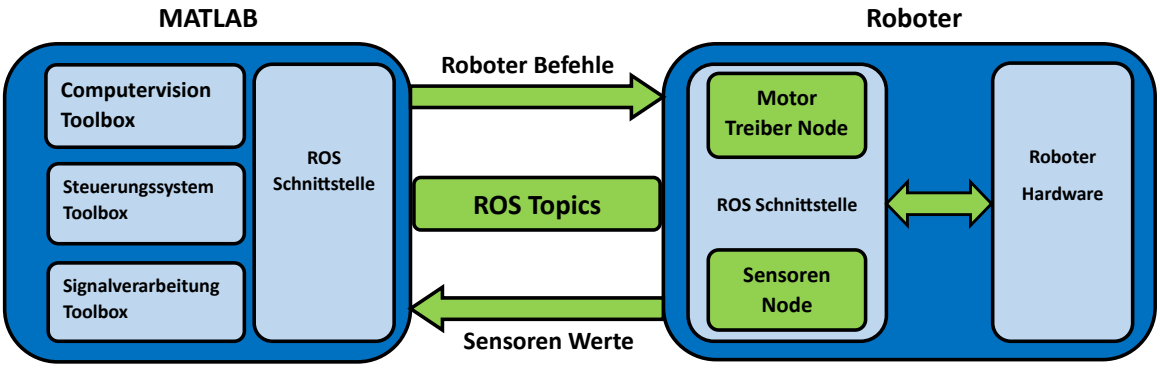

Abbildung 3.1.: Kommunikation zwischen MATLAB und Roboter über eine ROS Schnittstelle

Die Robotic System Toolbox bietet eine Schnittstelle von MATLAB und Simulink zu dem Robot Operating System. Dies ermöglicht es mit einem ROS-Netzwerk zu kommunizieren, Roboterfunktionen interaktiv zu erkunden und Sensordaten zu visualisieren. Außerdem können Robotik-Algorithmen und Anwendungen auf ROS-fähigen physischen Robotern und in Robotersimulatoren wie Gazebo und V-REP entwickelt und getestet werden. Zusätzlich besteht die Möglichkeit ein unabhängiges ROS-Netzwerk direkt in MATLAB und/oder Simulink zu erstellen. In dieses Netzwerk können ROS-Protokolldateien (**rosbag**) importieren werden um diese zu visualisieren und auszuwerten. Mit dieser Funktion können Robotik-Algorithmen in MATLAB und Simulink entwickeln und gleichzeitig Nachrichten mit anderen Knoten im ROS-Netzwerk ausgetauscht werden. Mit dem Embedded Coder kann C++ - Code in einem Simulink-Modell für eine eigenständige ROS-Anwendung generiert werden, die auf jeder Linux-Plattform ausgeführt werden kann, auf der ROS installiert ist.
Hauptmerkmale der Robotics System Toolbox sind:

- Kommunikation zwischen Matlab und einem ROS-Netzwerk, Roboterfunktionen testen und Sensordaten visualisieren
- ROS-Nodes(**rosnode**), Publisher und Subscriber direkt aus MATLAB und Simulink erstellen
- ROS Nachrichten (**rosmsg**) erstellen und senden
- ROS-Dienste (**rosservice**) aufrufen und bereitstellen
- ROS-Funktionalitäten werden auf jeder Plattform (Windows, Linux, Mac) verwendet
- MATLAB als ROS-Master verwenden
- Simulink-Modelle erstellen, die mit einem ROS-Netzwerk arbeitet

3.4.1. Kommunikation zwischen ROS und MATLAB

Ein ROS-Netzwerk besteht aus einem einzelnen ROS-Master und mehreren ROS-Nodes. Der ROS-Master erleichtert die Kommunikation im ROS-Netzwerk, indem er alle aktiven ROS-Einheiten überwacht. Jede Node muss sich beim ROS-Master mit ihrer Netzwerkadresse anmelden, um mit dem Rest des ROS-Netzwerks kommunizieren zu können. Ein Master kann sowohl innerhalb von MATLAB als auch außerhalb (z.B. auf einem anderen Rechner / Roboter) gestartet werden. Im Rahmen dieses Projekts sind vor allem folgende Aspekte der Kommunikation zwischen MATLAB und ROS von Bedeutung:

- Um eine Verbindung zu einem ROS-Netzwerk herzustellen, kann der ROS-Master entweder direkt in Matlab erstellt oder mit einem vorhandenem Master verbunden werden. In beiden Fällen erstellt MATLAB seinen eigenen ROS-Node (den sogenannten **"globalen node"** und meldet diesen beim Master an. Die rosinit Funktion verwaltet diesen Prozess.
- Der Datenaustausch startet, sobald eine Verbindung hergestellt ist. Dann tauscht MATLAB Daten mit anderen ROS-Nodes über Publisher, Subscriber und Services aus.
- Durch einen Aufruf der Funktion "rosshutdown" wird MATLAB vom ROS-Netzwerk getrennt.

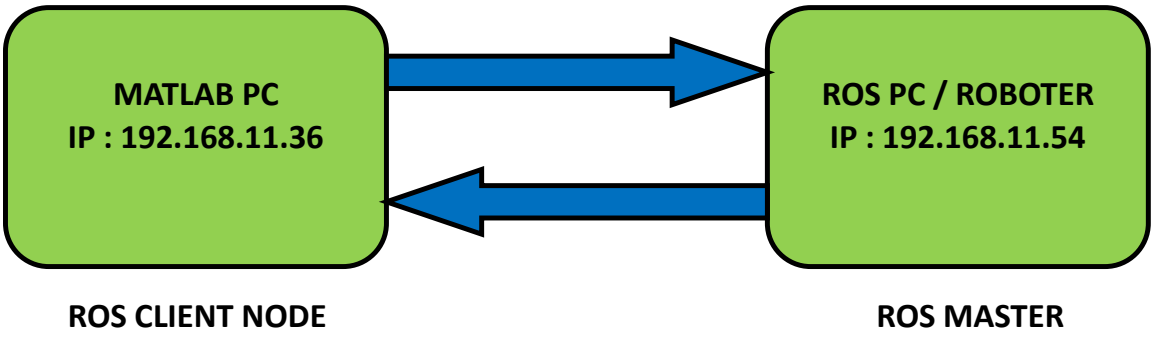

Abbildung 3.2.: Kommunikation zwischen MATLAB und ROS

Um die Kommunikation zwischen MATLAB und ROS zu gewährleisten, müssen beide über LAN oder Wifi mit dem gleichen Netzwerk verbunden sein und sich im gleichen Subnetz (d. h. gleicher Netzwerkteil in der IP-Adresse) befinden. Um die IP-Adresse eines Geräts zu ermitteln, kann unter Windows in der Konsolenzeile der Befehl **"ipconfig"** verwendet werden, während unter Ubuntu im Linux-Terminal **"ifconfig"** das Äquivalent darstellt. In der 3.2 ist zu sehen, dass sich beide Knoten ein Subnetz teilen.

3.4.2. Grundlagen ROS-Funktionen in MATLAB

In diesem Kapitel wird eine kleine Übersicht der verwendeten ROS Befehle in MATLAB vorgestellt. In der Abbildung 3.3 ist die Kommunikation der ROS-Nodes in MATLAB dargestellt. Diese wird im nächst Abschnitt näher erklärt.

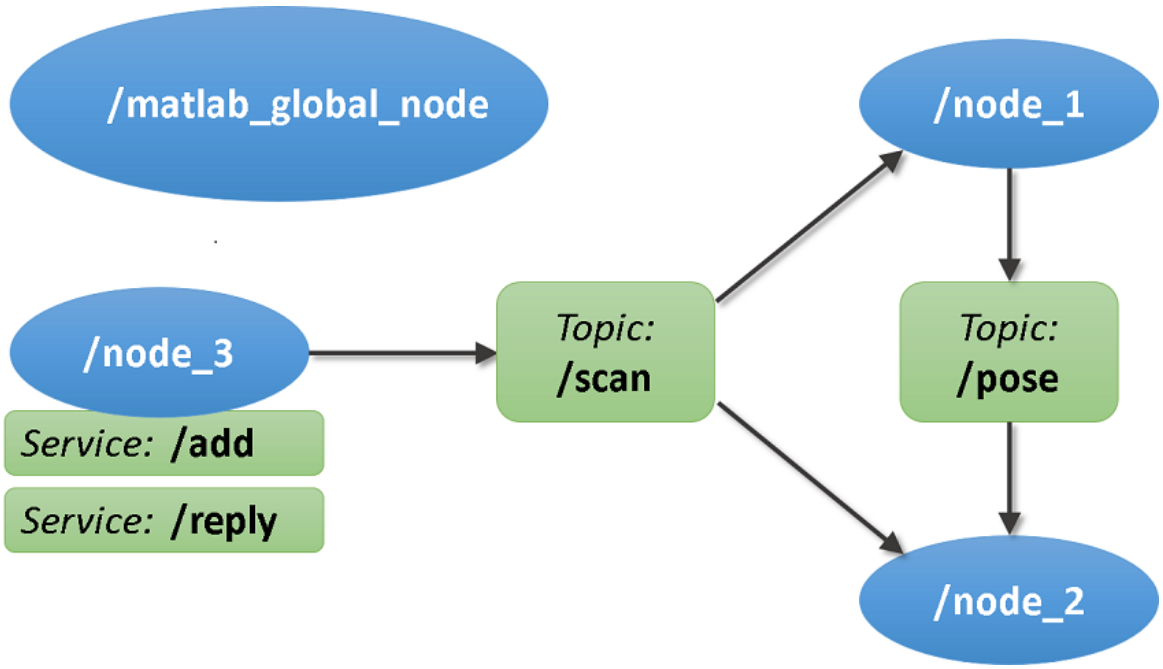

Abbildung 3.3.: Ein Beispiel für MATLAB und ROS Netzwerk [21]

3.4.3. ROS-Netzwerk zu initialisieren in MATLAB

In der Abbildung 3.4 wird der **rosinit** Befehl verwendet um ROS zu initialisieren. Der Befehl erstellt einen ROS Master in MATLAB und startet einen **global_node**, welcher mit dem Master verbunden ist. Der global_node wird automatisch von anderen ROS-Funktionen verwendet.

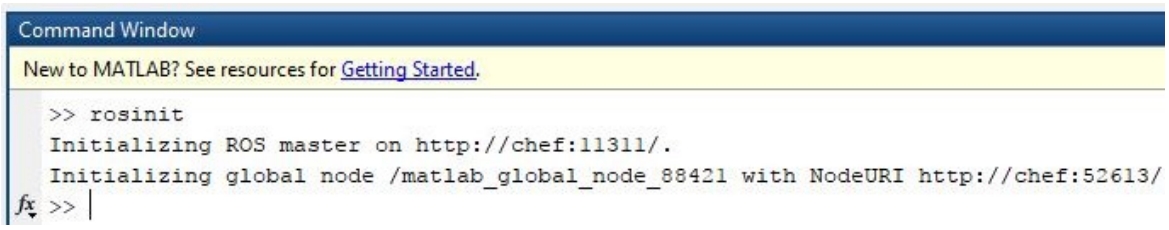

Abbildung 3.4.: rosinit Befehl in MATLAB

rosnode list: Nachdem das Netzwerk initialisiert wird, können mit dem Befehl "rosnode list" die ROS-Nodes in dem ROS-Netzwerk überprüft werden. Hier werden nur verfügbare Nodes angezeigt, die mit global_node und rosinit erstellt wurden.
/**matlab_global_node_88421**: Zeigt an, dass der ROS-Master in MATLAB erfolgreich initialisiert wurde.
Mit dem beispiel Programm "exampleHelperROSCreateSampleNetwork" fügt man dem ROS-Netzwerk drei zusätzliche Nodes, einen Sample Publisher und einen Subscriber, hinzu. Der Publisher sendet Nachricht, ein Subscriber empfängt die Nachricht.

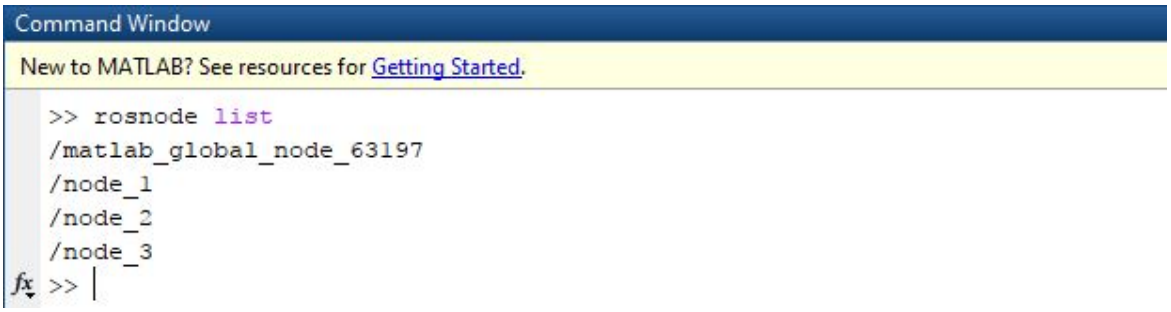

Abbildung 3.5.: rosnode list in MATLAB

rostopic list: Nachdem das Beispielprogramm ausgeführt wird, werden noch drei weitere Nodes im ROS-Netzwerk hinzugefügt. Mit dem Befehl rostopic list werden die verfügbaren Topics im ROS-Netzwerk gelistet.

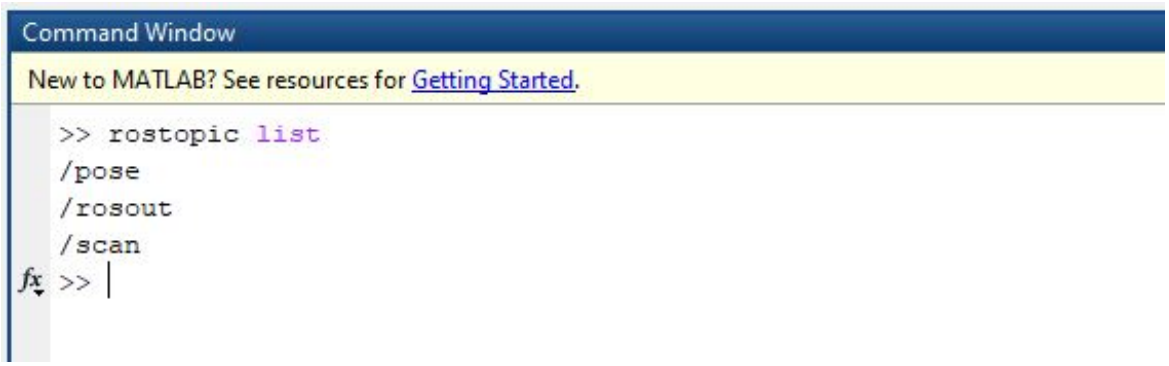

Abbildung 3.6.: rostopic list in MATLAB

rostopic info /pose: Mit dem Befehl "rostopic info /pose" können spezifische Informationen zu einem bestimmten Topic abgefragt werden. Der nachfolgende Befehl zeigt, dass /node_1 Nachrichten an /pose" Topic und /node_2" sendet und diese wiederum die Nachrichten empfangen.

Abbildung 3.7.: rostopic info /pose in MATLAB

rosservice: Der ROS-Service stellt einen Mechanismus für "Prozeduraufrufe" über das ROS-Netzwerk bereit. Ein Service-Client sendet eine Anforderungsnachricht an einen

Service-Server, der die Informationen in der Anfrage verarbeitet und mit einer Antwortnachricht zurückkehrt.
Mit dem Befehl **"rosservice list"** wird verfügbaren Services gelistet.
Der **"rosservice info /xxx"** gibt die Informationen über bestimmte Services.
rosmsg: Publisher, Subscriber und Service verwenden ROS-Nachrichten, um Informationen auszutauschen. Jede ROS-Nachricht hat einen zugeordneten Nachrichtentyp, der die Datentypen und das Layout der Informationen in dieser Nachricht definiert.
rosmsg show geometry_msgs/Twist
rosmsg list
Die weiteren ROS-Befehle in MATLAB können unter Nutzung folgendem Befehl aufgerufen werden.
⇒ **help robotics.ros**

3.4.4. MATLAB mit dem ROS-Netzwerk verbinden

Die IP-Adresse von ROS muss durch die Umgebungsvariable **ROS_MASTER_URI** im Betriebssystem definiert werden. Hierbei ist darauf zu achten, auch den Port, über den die Kommunikation laufen soll, mit anzugeben. Der Standard-Port ist 11311. Vor dem Kommunikationsaufbau muss der Befehl **roscore** im Netzwerk ausgeführt werden. Die Abbildung 3.8 zeigt die Einrichtung in MATLAB.

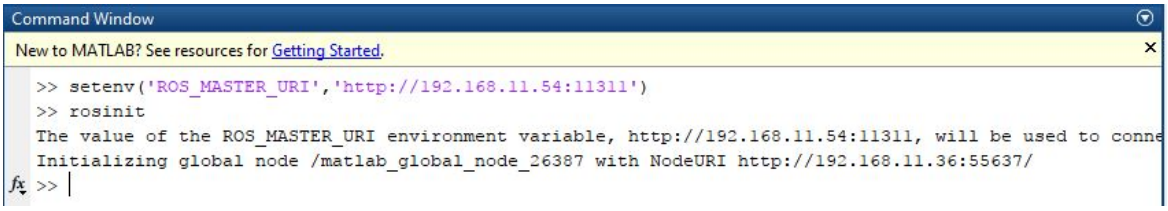

Abbildung 3.8.: MATLAB mit dem ROS-Master verbinden

4. youBot Simulation mit MATLAB

In diesem Abschnitt wird auf eine youBot Simulation in Simulink eingegangen. Als Vorlage dient die Studienarbeit [28]. In dieser Arbeit wird der youBot in MATLAB programmiert und ein Robotermodell in Simulink realisiert. In seiner Studienarbeit hat er den youBot Roboter in MATLAB programmiert und das Robotermodell in Simulink dargestellt. Bevor die Ansteuerung des youBot durch MATLAB untersucht wird, sollen einige Grundlagen erläutert werden. Im Bereich der Robotik wird oft die Lokalisierung von Objekten in dreidimensionalem Raum angewendet. Diese Objekte sind Gelenke des Manipulators, Teile und Werkzeuge mit denen er sich befasst und andere Objekte in der Umgebung. Die Objekte werden durch ihre **Position** und **Orientierung** beschrieben.

Um die Position und die Orientierung eines Objekts zu beschreiben, wird eine Koordinatensystem oder ein Rahmen an das Objekt angehängt. Dann wird die Position und die Orientierung dieses Rahmens in Bezug auf ein Referenzkoordinatensystem beschrieben [4]. Die KOS sind in Abbildung 4.1 dargestellt.

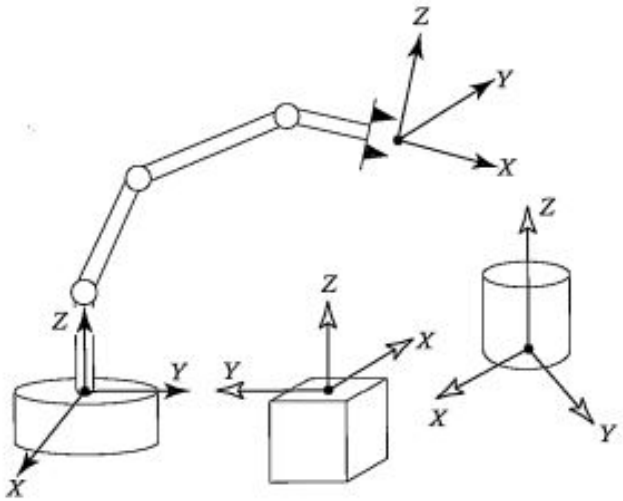

Abbildung 4.1.: Koordinatensystem an dem Manipulator [4]

4.1. Direkte Kinematik (Vorwärtskinematik)

Die Kinematik ist die Wissenschaft der Bewegung, der Position, der Geschwindigkeit und der Beschleunigung. Daher bezieht sich das Studium der Kinematik von Manipulatoren auf alle geometrischen und zeitbasierten Eigenschaften der Bewegung. Manipulatoren bestehen fast nur aus starren Gliedern, welche durch Gelenke verbunden sind. Die Gelenke sind mit Positionssensoren ausgestattet, die genauestens die Position des Gelenks messen können. Einige Manipulatoren haben sog. prismatische Gelenke, die eine relative Verschiebung zwischen einzelnen Gliedern zulassen. Diese translatorische Bewegung wird als "Gelenkoffset" bezeichnet. Bei typischen Industrierobotern wird in der Regel ein Manipulator als eine offene kinematische Kette und jede Gelenkposition durch eine Variable definiert. Die Anzahl der Gelenke gibt den Freiheitsgrad (DOF) des Manipulators an. Das letzte Element einer kinematischen Kette wird als Endeffektor (TCP) bezeichnet. Hier wird in der Regel das Werkzeug des Arms montiert, z.B. eine Schweißeinheit oder ein Greifer.

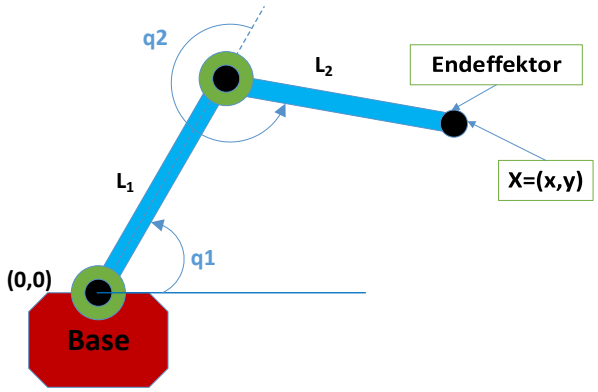

Abbildung 4.2.: Direkte Kinematik vom Roboter mit zwei Gelenken

Bei Methodik der direkten oder Vorwärtskinematik wird aus den Gelenkwinkeln des Manipulators die Position und Orientierung des Endeffektors bestimmt. Bei der Mehrzahl der Manipulatoren werden die Gelenkwinkel kontinuierlich durch Sensoren überwacht - sie sind also jederzeit bekannt. Befindet sich der Endeffektor also in der falschen Pose, um eine Aufgabe zu verrichten, kann aus die Differenz zwischen der aktuellen und gewünschten Pose errechnet und an die die Positionsteuerung ([12] S.39) weitergegeben werden. Insbesondere bei gegebenen Gelenkwinkeln besteht das Problem der Vorwärtskinematik darin, die Position und Ausrichtung des Werkzeugrahmens relativ zum Referenz-KOS oder Grundrahmen zu berechnen ([4] S.6).
Das KOS des Endeffektors $\vec{x}$ enthält die kartesischen Koordinaten x, y, z und die Eulerwinkel α, β, γ, die die Position und Orientierung des KOS im Raum beschreiben. Die Gelenkkoordinaten $\vec{q}$ gehen in die Funktion $f(\vec{q})$ ein, diese kennzeichnet die Abhängigkeit der Pose

von den Gelenkwinkeln ([28] S.11).

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \\ \alpha \\ \beta \\ \gamma \end{pmatrix} = f \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{pmatrix}, \text{ n ist die Anzahl der Gelenkwinkel} \tag{4.1}$$

Die Berechnung der direkten Kinematik des Endeffektors zu der Basis.

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \\ \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} L_1 cos(q_1) + L_2 cos(q_1 + q_2) \\ L_1 sin(q_1) + L_2 sin(q_1 + q_2) \\ 0 \\ 0 \\ 0 \\ q_1 + q_2 \end{pmatrix} \tag{4.2}$$

4.2. Inverse Kinematik (Rückwärtskinematik)

Die inverse Kinematik (IK) bildet das methodische Gegenstück zur direkten Kinematik. Mit ihrer Hilfe werden einzelne Gelenkwinkel des Manipulators aus der Position und der Orientierung des Endeffektors berechnet. Die kartesischen Koordinaten werden auf die Gelenkkoordinaten der Roboterachsen transformiert.

Hieraus ergibt sich die Gleichung 4.3 für IK.

$$\vec{q} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{pmatrix} = f^{-1} \begin{pmatrix} x \\ y \\ z \\ \alpha \\ \beta \\ \gamma \end{pmatrix} \text{ (nach [28] S.12)} \tag{4.3}$$

Mit dieser Formel lässt sich bspw. die inverse Kinematik eines zwei-gelenkigen Roboterarms berechnen.

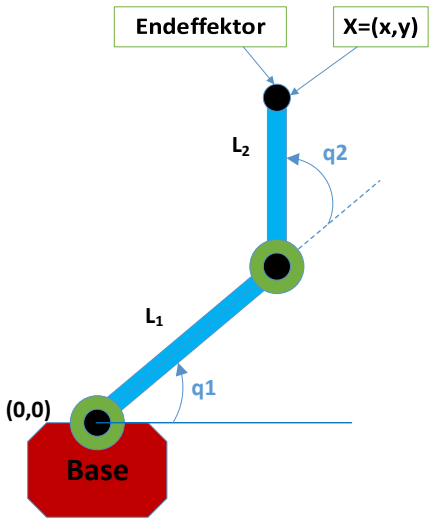

Abbildung 4.3.: Inverse Kinematik vom Roboter mit zwei Gelenken

$$\vec{q} = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = f^{-1} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \arcsin\left(\frac{y}{\sqrt{x^2+y^2}}\right) \pm \arcsin\left(\frac{\sqrt{(2L_1L_2)^2-(x^2+y^2-L_1^2-L_2^2)^2}}{2L_1\sqrt{x^2+y^2}}\right) \\ \pm \arccos\left(\frac{x^2+y^2-L_1^2-L_2^2}{2L_1^2L_2^2}\right) \end{pmatrix} \tag{4.4}$$

Das IK Problem ist nicht so einfach wie das der DK. Die Gleichungen sind nicht linear. Es können unzulässige Konfigurationen entstehen. Diese sind mathematisch zwar vielleicht korrekt, können von den Gelenken aber nicht erfüllt bzw. eingenommen werden oder führen zu nicht erreichbaren Ziellagen ([4] S.7).

4.3. DH Notation des youBot Roboters

Die Vorwärtskinematik befasst sich mit der Frage, wie aus den Gelenkwinkeln der Armelemente eines Roboters die Pose (Position und Orientierung) des Endeffektors in Bezug auf das Basiskoordinatensystem bestimmt werden kann. Anders als bei der inversen Kinematik werden die Gelenkwinkel bis zur Endeffektorposition im kartesischen Raum transformiert. Um diese Transformation zu berechnen, wird sehr oft die Denavit-Hartenberg (DH) Konvention angewendet. Ein offener Kettenmanipulator besteht aus $n + 1$ Gelenken, die durch n Gelenke verbunden sind.

Da eine Verbindung zwei aufeinanderfolgende Gelenke verbindet, kann eine homogene

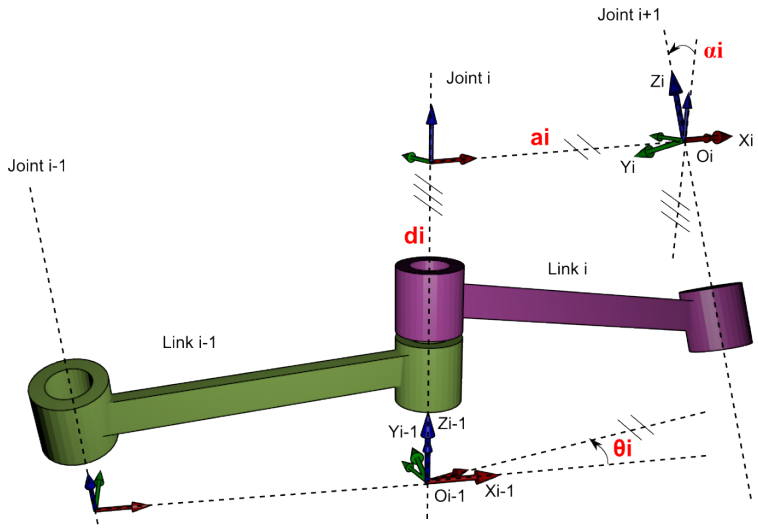

Abbildung 4.4.: Klassische DH Konvention [44]

Transformation die Position und Orientierung eines Koordinatensystems auf einer Verbindung in Bezug auf das vorherige / nächste beschreiben. Der Zweck der D-H-Konvention besteht darin, die Ableitung dieser homogenen Transformationen zu erleichtern und die Vorwärtskinematikskarte rekursiv durch Matrixmultiplikation dieser Transformationen zu finden.

Insbesondere verwendet die DH-Konvention vier Parameter, um die Position des Rahmens i in Bezug auf den Rahmen $i-1$ zu transformieren:

- a_i, der Abstand zwischen den Ursprüngen der zwei Koordinatenrahmen O_i und O_{i-1}
- d_i, die Koordinate von O_i entlang z_{i-1}
- α_i, der Winkel zwischen den Achsen z_{i-1} und z_i um die Achse x_i
- θ_i, der Winkel zwischen den Achsen x_{i-1} und x_i um die Achse z_{i-1}

In der Abbildung 4.5 wird der youBot mit seinen Gelenken und dem jeweiligem Drehsinn dargestellt. Bei der Berechnung der DH-Konvention wird der gesamte youBot mit Arm und Plattform als kinematische Kette mit acht seriell angeordneten Gelenken betrachtet.

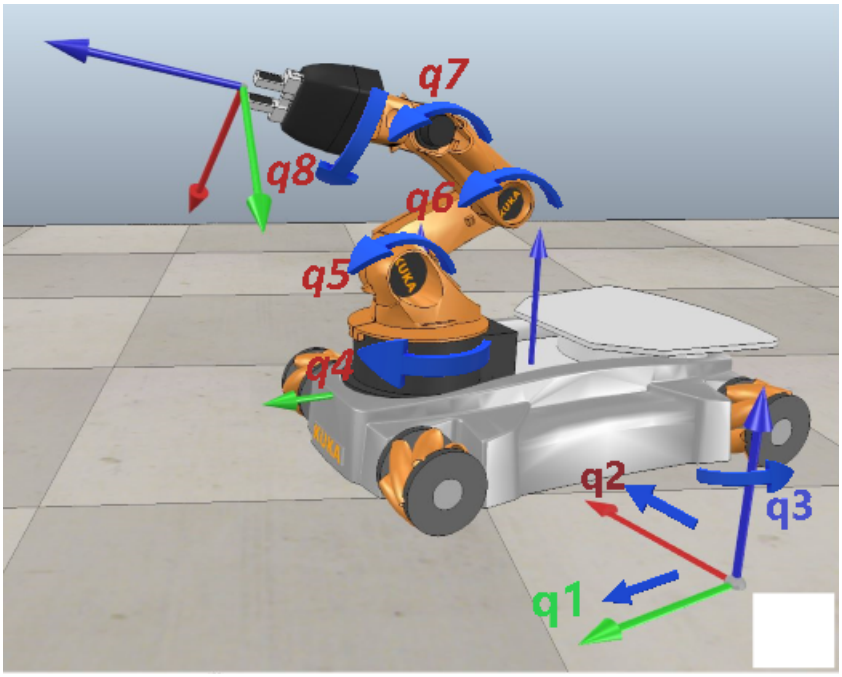

Abbildung 4.5.: youBot Gelenkwinkel

In der Tabelle 4.1 ist die DH-Notation des youBot Roboters tabellarisch dargestellt.

KOS	Theta	d	a	α
$0 \rightarrow 1$	0	q_1	0	$\pi/2$
$1 \rightarrow 2$	$\pi/2$	q_2	0	$\pi/2$
$2 \rightarrow 3$	q_3	0	0.1505	π
$3 \rightarrow 4$	q_4	-0.267	0.033	$\pi/2$
$4 \rightarrow 5$	$q_5 - \pi/2$	0	0.155	0
$5 \rightarrow 6$	q_6	0	0.135	0
$6 \rightarrow 7$	$q_7 + \pi/2$	0	0	$\pi/2$
$7 \rightarrow 8$	$q_8 + \pi$	0.218	0	0

Tabelle 4.1.: DH-Notation für den youBot

4.4. Jacobimatrix

Während die Berechnung der IK eines zwei-gelenkigen Roboterarm noch möglich ist, entfällt dieser geometrische Lösungsansatz, sobald mehr Segmente im Spiel sind. Aus diesem Grund wird für die Berechnung mehr-gelenkiger Roboterarme die Jacobimatrix verwendet. Die Jacobimatrix beschreibt eine Beziehung zwischen Gelenkraten und Geschwindigkeit des Endeffektors und wird oft für Geschwindigkeitsanalysen innerhalb des betrachteten Systems verwendet. Sie wird auch als Funktionsmatrix [41] oder Ableitungsmatrix bezeichnet und wird zur Approximation oder Minimierung mehrdimensionaler Funktionen eingesetzt. Die Ableitung der Funktion eines M-Vektors y von einem n-Vektor x nach dem Vektor x ergibt die (m,n) Jacobimatrix [45]. Im Bereich der Robotik wird im Allgemeinen die Jacobimatrix verwendet, um die Gelenkgeschwindigkeit auf die kartesische Geschwindigkeit des Endeffektors zu beziehen ([4] S.150).

$$\frac{\partial y(x)}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} = \left(\frac{\partial y_i}{\partial x_k} \right) \tag{4.5}$$

Die direkte Kinematik beschreibt also die Position des Endeffektors bei vorgegebenen Gelenkwinkeln und die Jacobimatrix berechnet die Änderung der Position des Endeffektors bei vorgegebenen Winkeln an. Für die Position gilt folglich:

$$\dot{\vec{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \underline{J}_{A,P}(\vec{q})\dot{\vec{q}} = \begin{pmatrix} \frac{\partial x_1}{\partial q_1} & \cdots & \frac{\partial x}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial z}{\partial q_1} & \cdots & \frac{\partial z}{\partial q_n} \end{pmatrix} \begin{pmatrix} dq_1 \\ \vdots \\ dq_n \end{pmatrix} \tag{4.6}$$

In der Gleichung 4.6 drückt die Jacobimatrix die Änderung der Lage in Abhängigkeit zur Änderung der Winkel aus. Diese Jacobimatrix wird als analytische Jacobimatrix bezeichnet ([28] S.20). Der Zusammenhang zwischen den Gelenkkoordinaten und $q \in R^n$ und den Umweltkoordinaten des Endeffektors $x \in R^m$ ist durch die direkte Kinematik x = f(q) gegeben. Die entsprechenden differentiellen Zusammenhänge werden durch die analytische Jacobimatrix J_A(q) beschrieben.

$$\dot{x} = \frac{\partial f}{\partial q}\dot{q} \quad bzw. \quad dx = J_A(q)dq \tag{4.7}$$

Die analytische Jacobimatrix ist dabei die Ableitung der vektoriellen Funktion $f(q)$ bezüglich aller Gelenkwinkel. Sie hat die Dimension m x n. Hier wird n=dim(q) als Zahl der Minimalkoordinaten und m=dim(x) Dimension des Arbeitsraumes bezeichnet [7].

Die andere Jacobimatrix wird als geometrisch bezeichnet. Diese Matrix J_G beschreibt die Abhängigkeit der Geschwindigkeit ν und der Winkelgeschwindigkeit ω des Endeffektors von

den Gelenkgeschwindigkeiten:

$$\dot{x}_G = \left[\nu^T, \omega^T\right]^T = J_G(q)\dot{q} \tag{4.8}$$

Die Spalten von J(q) lassen sich in einen translatorischen Anteil J_{T_i} und einen rotatorischen Anteil J_{R_i} aufteilen. Für m = 6: dim(J_{T_i}) = dim(J_{R_i}) = (3x1).

$$\dot{x} = \begin{bmatrix} \nu \\ \omega \end{bmatrix} = \begin{bmatrix} J_{T1} & J_{T2} & \cdots & J_{Tn} \\ J_{R1} & J_{R2} & \cdots & J_{Rn} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix} \tag{4.9}$$

Geometrische Jacobimatrix für Schub- und Drehgelenken:

$$Schubgelenke : \begin{bmatrix} J_{Ti} \\ J_{Ri} \end{bmatrix} = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix} \tag{4.10}$$

$$Drehgelenke : \begin{bmatrix} J_{Ti} \\ J_{Ri} \end{bmatrix} = \begin{bmatrix} z_{i-1} & x & r_{i-1,E} \\ & z_{i-1} & \end{bmatrix} \tag{4.11}$$

Dabei ist $r_{i-1,E}$ der Vektor vom Ursprung (KS_{i-1}) zum Endeffektor. Wenn n = m ist, dann gilt:

1. Umrechnung von Bahn- in Gelenkkoordinaten für $det(J) \neq 0 : \dot{q} = J^{-1} \cdot \dot{x}$
2. In singulären Punkten gilt $det(J) = 0$
3. Rekursionsformel für die Implementierung:

$$q_{k+1} = q_k + J_k^{-1} \cdot (x_{k+1} - x_k) \tag{4.12}$$

Bei dem youBot Modell wird eine 6x8 Jacobimatrix verwendet.

$$\underline{J}_G = \begin{pmatrix} \underline{J}_{G,T} \\ \underline{J}_{G,R} \end{pmatrix}^{6x8} \tag{4.13}$$

Der youBot Roboter besteht aus zwei Teilen. Ein Teil ist die Plattform und der zweite Teil ist der Manipulator. Das ergibt eine kinematische Kette der Form SSDDDDDD für den youBot, wenn D für ein Drehgelenk und S für ein Schubgelenk steht. Die ersten drei Buchstaben beschreiben die Plattform. Der Roboter kann sich um seine eigene z-Achse drehen. Ebenso kann sich dieser in die Richtung der x-Achse, y-Achse oder xy-Achse bewegen. Die Mecanum Räder ermöglichen diese Bewegungsfreiheit. Da diese Matrix nicht quadratisch ist, lässt sie sich nicht invertieren. Symbolisch ausgedrückt sieht die geometrische Jacobimatrix

des youBot wie folgt aus:

$$\underline{J}_G = \begin{pmatrix} \underline{J}_{G,T} \\ \underline{J}_{G,R} \end{pmatrix}^{6x8} = \begin{pmatrix} \vec{z}_1 & \vec{z}_2 & \vec{z}_3 \times {}^3\vec{r}_{EE} & \vec{z}_4 \times {}^4\vec{r}_{EE} & \vec{z}_5 \times {}^5\vec{r}_{EE} & \vec{z}_6 \times {}^6\vec{r}_{EE} & \vec{z}_7 \times {}^7\vec{r}_{EE} & \vec{z}_8 \times {}^8\vec{r}_{EE} \\ \vec{0} & \vec{0} & \vec{z}_3 & \vec{z}_4 & \vec{z}_5 & \vec{z}_6 & \vec{z}_7 & \vec{z}_8 \end{pmatrix} \tag{4.14}$$

Die Jacobimatrix wird mit MATLAB berechnet. Das Ergebnis lässt sich auf Grund der Größe der Matrix nicht zielführend in dieser Arbeit darstellen.

4.5. Pseudoinverse

Da die geometrische Jacobimatrix des youBot Roboters $J_G \in \mathbb{R}^{6x8}$ nicht invertierbar ist, müssen mit Hilfe von Pseudoinversen durch Annäherung zunächst die Inversen bestimmt werden. Über die Jacobimatrix lassen sich folgende Aussagen machen [1]:

- wenn m < n, dann ist das System unterbestimmt, d.h. mehr Unbekannte als Gleichungen
- wenn m = n, dann existieren gleich viele Gleichungen wie Unbekannte
- wenn m > n, dann ist das System überbestimmt, es gibt mehr Gleichungen als Unbekannten

Beim youBot handelt es sich nach der o.g. Klassifikation um ein überbestimmtes System mit sechs Unbekannten und acht Gelenken, die durch Gleichungen beschrieben sind (acht Freiheitsgrade). Das bedeutet, die gewünschte Position des Endeffektors kann berechnet mit Hilfe der Jacobimatrix berechnet werden, jedoch nicht ohne die Hilfe der Pseudoinversen. Die Pseudoinverse [42] einer Matrix ist eine Verallgemeinerung der inversen Matrix auf singuläre und nicht quadratische Matrizen.

$$AA^+ = A \tag{4.15}$$

Es gibt zwei Arten von Pseudoinversen:

- mit vollem Zeilenrang r = m > n, rechte Pseudoinverse A_R^+

$$A_R^+ = A^T(AA^T)^{-1} \tag{4.16}$$

- mit vollem Spaltenrang r = m < n, linke Pseudoinverse A_L^+

$$A_L^+ = (AA^T)^{-1}A^T \tag{4.17}$$

Der youBot Roboter hat den vollen Zeilenrang r = m, daher wird die rechte Pseudoinverse der Jacobimatrix benutzt.

$$J_R^+ = J^T(JT^T)^{-1} \in \mathbb{R}^{8x6} \tag{4.18}$$

4.6. Die Pose des Endeffektors halten beim youBot

Ein Teilziel dieser Arbeit ist es, das Halten der Pose des Endeffektors zu realisieren. Das heißt, während der Bewegung der Plattform sollte der Endeffektor seine Position und seine Orientierung (zum Referenz-Koordinatensystem (KOS), nicht zur Plattform) beibehalten können. Die Bewegung der Plattform wird durch die Schubgelenkparameter q_1 und q_2 beschrieben. Um die Pose des Endeffektors zu halten, werden die Schubachsen von der Roboterstruktur abgekoppelt. Zu diesem Zweck wird eine neue Berechnungsvorschrift hergeleitet. In der Gleichung 4.19 besteht die Jacobimatrix aus zwei Teilen. Der erste Teil steht für die sechs Rotationsachsen und der zweite für die translatorischen Achsenvorgaben. J ist eine 6×6 Matrix und steht für die normale Jacobimatrix. J_v ist eine 2×1 Matrix und steht für die Jacobimatrix Vorgabe.

$$\dot{x} = J\dot{q} + J_v\dot{q}_v \tag{4.19}$$

q steht hier für die unbekannten Größen der Gelenke und q_v ist für die Gelenkvorgaben. Die inverse Kinematik sieht wie in der Gl. 4.20 aus.

$$\dot{q} = J^{-1}(\dot{x} - J_v\dot{q}_v) \tag{4.20}$$

In der Gl. 4.21 resultierende Rekursionsformel ergibt:

$$\frac{q_{i+1} - q_i}{\triangle t} = J^{-1}\left(\frac{x_{i+1} - x_i}{\triangle t} - J_v q_v\right) \tag{4.21}$$

Umstellen der Gleichung 4.21 nach $q_1 + 1$.

$$q_{i+1} = q_i + J^{-1}\left(\frac{x_{i+1} - x_i}{\triangle t} - J_v q_v\right)\triangle t \tag{4.22}$$

Auf den Achsen q_1 und q_2 wird die Plattform in der horizontalen Ebene linear bewegt. Die vorgegebene Achsenwerte $\dot{q}_v = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}$ fließen direkt in der Rekursionsformel ein.

Die geometrische Jacobimatrix wird in zwei Matrizen J und J_v aufgeteilt. Von der $J_G \in \mathbb{R}^{6x8}$ Jacobimatrix wird die erste und zweite Spalte abgezogen. In der Gleichungen 4.23 und 4.24 wird dieses dargestellt.

$$\begin{aligned} \underline{J} &= \begin{pmatrix} \underline{J}_{G,T} \\ \underline{J}_{G,R} \end{pmatrix}^{6x6} \\ &= \begin{pmatrix} \vec{z}_3 \times {}^3\vec{r}_{EE} & \vec{z}_4 \times {}^4\vec{r}_{EE} & \vec{z}_5 \times {}^5\vec{r}_{EE} & \vec{z}_6 \times {}^6\vec{r}_{EE} & \vec{z}_7 \times {}^7\vec{r}_{EE} & \vec{z}_8 \times {}^8\vec{r}_{EE} \\ \vec{z}_3 & \vec{z}_4 & \vec{z}_5 & \vec{z}_6 & \vec{z}_7 & \vec{z}_8 \end{pmatrix} \end{aligned} \tag{4.23}$$

$$\underline{J}_v = \begin{pmatrix} \underline{J}_{G,T} \\ \underline{J}_{G,R} \end{pmatrix}^{6x2} = \begin{pmatrix} \vec{z}_1 & \vec{z}_2 \\ \vec{0} & \vec{0} \end{pmatrix} \tag{4.24}$$

Die Pose des Endeffektors kann bei Bewegung der Plattform auf beiden Schubachsen q_1 und q_2 mit Hilfe dieser Jacobimatrix in der Steuerungsimplementierung erfolgreich gehalten werden. Allerdings tritt beim youBot im realen Umfeld ein Fehler auf: Die Plattform sollte zum Ende des Testszenarios zur ihrer Startposition zurückkehren, hat aber nach dem Stillstand der Plattform einen Offset dazu. Dieser Fehler wirkt sich natürlich entlang der kinematischen Kette bis zum Endeffektor aus. Die beste Vermutung, die zu diesem Zeitpunkt über das Problem abgegeben werden kann, sind fehlerhafter Reglerparameter für den Roboter.

Solange sich der vom Endeffektor zu haltende Punkt im Arbeitsbereich des youBot befindet, kann dieser in der Regel mit seinem Arm die Pose halten. In manchen Fällen jedoch, beispielsweise wenn der zu haltende Punkt außerhalb des Arbeitsbereichs liegt, "gefriert" der Arm in einer bestimmten Pose, weil es zu einer Fehlermeldung in ROS kommt, die einen Neustart des Roboters erforderlich macht.

4.7. youBot MATLAB und Simulink

Die Kenngrößen des youBot Roboters werden als Parameter in verschiedene MATLAB Dateien eingegeben. Durch diese Eingaben werden einige Funktionen für den youBot berechnet. In der nachfolgenden Liste sind die MATLAB Funktionen aufgeführt ([28] S.40), die dafür benutzt werden.

- parameter.m: Hier werden die Parameter (Länge, Winkel usw.) für den youBot vorgegeben, die für den Start der Simulation verwendet werden.

- j2p: "Joint to Position" hiermit wird die Pose des Endeffektors berechnet.
- JacobiInv.m: Dient zur Berechnung der Inversen Jacobimatrix.

In der Abbildung 4.6 ist das Simulink-Modell des youBot Roboters abgebildet. Dieses Modell besteht aus zwei Teilen. Der erste Teil stellt den Steuerungsteil dar, welcher das youBot-Modell in der Simulink-Simulation ansteuert. Das Modell bietet fünf Szenarien, die der Roboter ausführen kann.

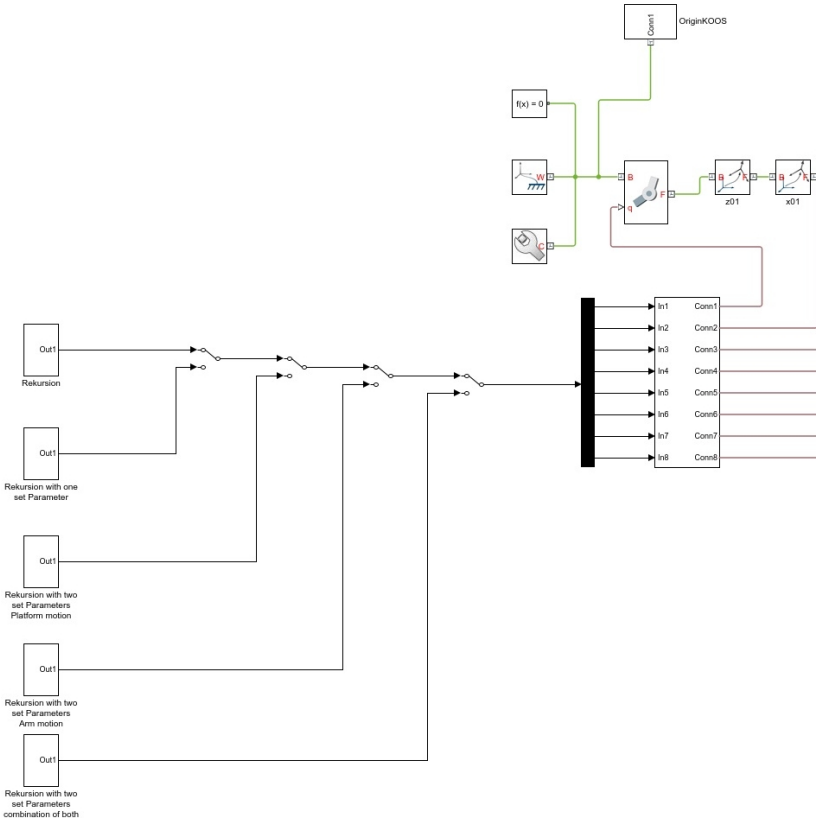

Abbildung 4.6.: youBot Simulink Modell

- Fall 1: Das erste Szenario wird Rekursion genannt. Dieses lässt den Roboter auf den y-Achse hin und her fahren.
- Fall 2: Dieses Senario wird "Recursion with one set parameter"genannt. Dabei wird in der Simulation die Plattform bewegt, während der Endeffektor seine Position hält.
- Fall 3: Das wird als "Recursion with two Parameters Platform Motion"bezeichnet. Dabei wird die youBot Plattform bewegt, während der Endeffektor seine Position hält.

- Fall 4: Das wird als "Recursion with two set Parameters Arm Motion"bezeichnet. Hier wird die youBot Plattform mit einem vorgegebenen Winkel um die z-Achse gedreht. Danach wird der Endeffektor des Manipulators in der x-Richtung bewegt.
- Fall 5: Das letzte Szenario wird "Rekursion with two set Parameters combination of both" ist genannt. Hier bewegt sich die Plattform zuerst in x- und y-Richtung, während sich der Arm bewegt. Danach dreht sich die Plattform um die z-Achse und bewegt ein Armgelenk. Bei der Kombinationen von beiden Punkten, wird der Endeffektor in der gleichen Pose gehalten.

Der zweite Teil des Simulink-Modells beschreibt den Roboter. Dabei sind einzelne Komponenten in Submodule unterteilt und ergeben zusammen das komplette Modell. Hier werden KOS und Gelenkarten (Drehgelenk oder Schubgelenk) des Roboters vorgegeben. Die zuvor beschriebenen fünf Szenarien werden über ein Demux mit dem Modell verbunden.

4.8. Vom SolidWorks Modell zu Simulink

In diesem Abschnitt wird die Konvertierung des CAD-Modells von KUKA youBot SolidWorks zu MATLAB Simscape erläutert. Simscape ermöglicht es, ein physikalische Modell zu erstellen und die physikalischen Verbindungseigenschaften zu definieren, welche direkt in Blockdiagrammen und anderen Modellierungsparameter integriert werden. Simscape Multibody Link ist ein Plugin von MATLAB für CAD-Anwendungen - in diesem Fall in SolidWorks - um die CAD-Baugruppenmodelle zu exportieren. Das Plugin generiert die Dateien, die zum Importieren des Modells in der Simscape Multibody-Umgebung benötigt werden. Das Plugin kann für drei CAD-Anwendungen installiert werden.

- SolidWorks
- Autodesk Inventor
- PTC Creo

4.8.1. Installation der Simscape Multibody-Link

Das Simscape Plugin wird von MATLAB zur Verfügung gestellt und kann auf der Webseite von MathWorks [19] für verschiedene MATLAB Versionen heruntergeladen werden. Im nächsten Schritt der Installation muss MATLAB mit Administrator-Rechten ausgeführt werden. Die gespeicherten Dateien (Simscape Multibody Link 5.2) müssen zum MATLAB-Pfad hinzugefügt werden, indem mit dem Befehl **addpath('foldername')** ihren Speicherort registriert. Danach installiert der Befehl **install_addon('zipname')** das Plugin. Nach erfolgreicher

Installation muss nun durch die Eingabe des Befehls **regmatlabserver** in das MATLAB Befehlsfenster der MATLAB Server registriert. Nun muss der SolidWorks Link in MATLAB durch den Befehl **smlink_linksw** freigeschaltet werden, wonach SolidWorks in MATLAB freigegeben sein sollte. Der letzte Schritt geschieht in SolidWorks: Im Menu Extras, das einige Zusatzanwendungen enthält, muss hier die Option "Simscape Multibody-Link" ausgewählt sein.

4.8.2. Export des youBot Modells von SolidWorks nach Simscape

Zunächst wird das Modell in SolidWorks als Baugruppe erstellt. Dabei wird das youBot Modell in drei kleiner Teil- Baugruppen implementiert: Die Plattform, der Arm und der Greifer. Diese Unterteilung reduziert die Komplexität des Modells auf einem höheren Abstraktionsniveau und erhöht die Übersichtlichkeit in den Teilmodellen. Letztlich werden diese Teile zu nur einer Baugruppe kombiniert, was zu dem in Abbildung 4.7 dargestellten Modell führt. Dieses wird als Simscape Multibody Link aus SolidWorks in einen Ordner exportiert. Die exportierte

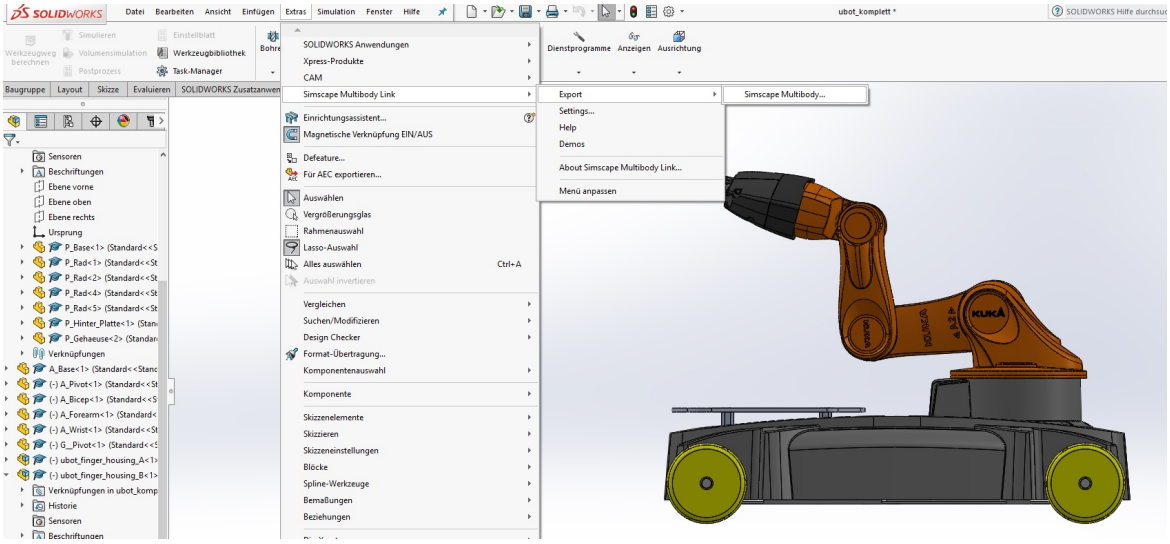

Abbildung 4.7.: Export youBot SolidWorks Modell nach Simscape

XML Datei kann nun mit dem Befehl **smimport('youbot.xml')** in MATLAB importiert werden. Abbildung 4.8 visualisiert die exportierte Simulink-Datei.

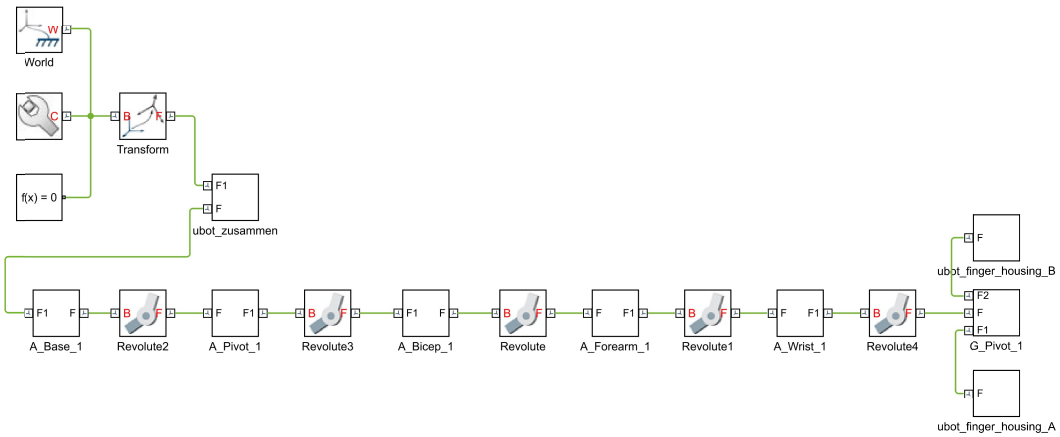

Abbildung 4.8.: youBot Simulink Modell aus SolidWorks

4.9. youBot API und Architektur

Die youBot a Application Programming Interface (API) [24] ist eine Programmierschnittstelle, die unter anderem vollständigen Zugriff auf die youBot Joints (Gelenke) bietet [16].

Die Grundidee der youBot API besteht darin, ein Robotersystem als eine Kombination von entkoppelten funktionalen Subsystemen darzustellen. Das heißt, dass die API den Manipulator und die Basisplattform als eine Kombination von mehreren Gelenken darstellt. Gleichzeitig besteht jedes Gelenk aus einem Motor und einem Getriebe. In der youBot API wird der Roboterarm als 5-DOF kinematische Kette dargestellt und die omnidirektionale Basisplattform wird als eine Sammlung von Drehgelenken behandelt. Es gibt drei Hauptklassen in der youBot API, die wichtig sind.

- **YouBotManipulator Klasse,** der youBot Arm wird als Aggregation mehrerer Gelenke und eines Greifers dargestellt.
- **YouBotBase Klasse,** stellt die Basisplattform dar.
- **YouBotJoint Klasse,** stellt die Gelenke dar, die sowohl den Manipulator als auch die Basis bilden.

MATLAB kommuniziert über ROS Nachrichten mit dem Trajectory Server, der die youBot API nutzt, um Ethercat Pakete mit Befehlen an den youBot zu schicken.

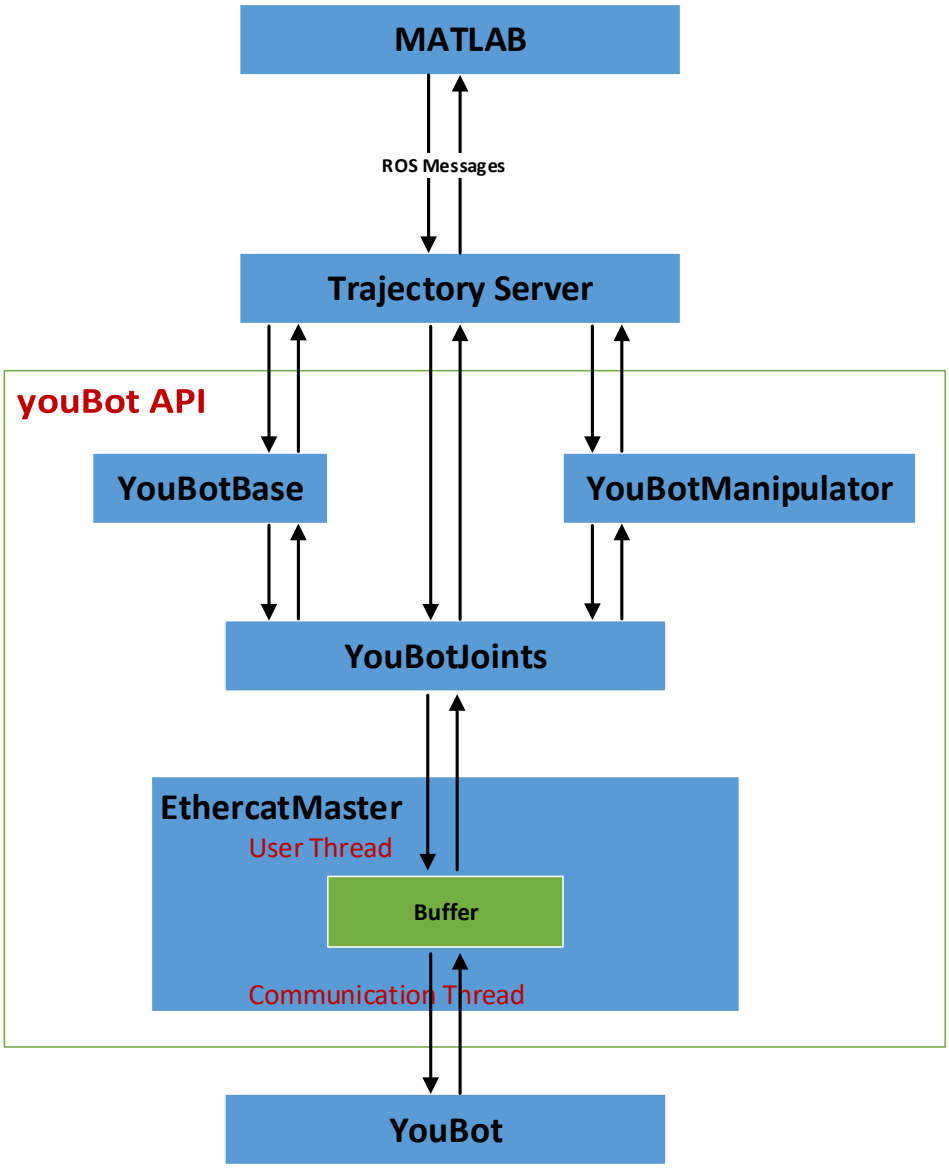

Abbildung 4.9.: youBot Architekturübersicht

5. Implementierung

In diesem Kapitel soll die Implementierung sowohl der physikalischen Modifikation des youBot Roboters also auch dessen Softwaresteuerung mit MATLAB und ROS, sowie der Simulation des Modells in Simulink erläutert werden. Die Implementierung baut auf Reznik's Arbeit [28], in der die Steuerung des youBot mit MATLAB und ROS erläutert wird, auf.

5.1. Datenverbindungen zwischen youBot und Raspberry Pi

Um die Kommunikation zwischen dem youBot Roboter und dem auf dem Raspi (mit ROS) sowie zwischen dem Raspi und einem Laptop (mit MATLAB), der als Steuerungscomputer verwendet wird, zu gewährleisten, müssen Datenverbindungen zwischen diesen Systemen hergestellt werden. Abbildung 5.1 zeigt eine schematische Darstellung für eine Ethernet LAN-basierte Verbindung. In der Abbildung 5.2 sind die physikalischen Komponenten in verbautem Zustand zu sehen.

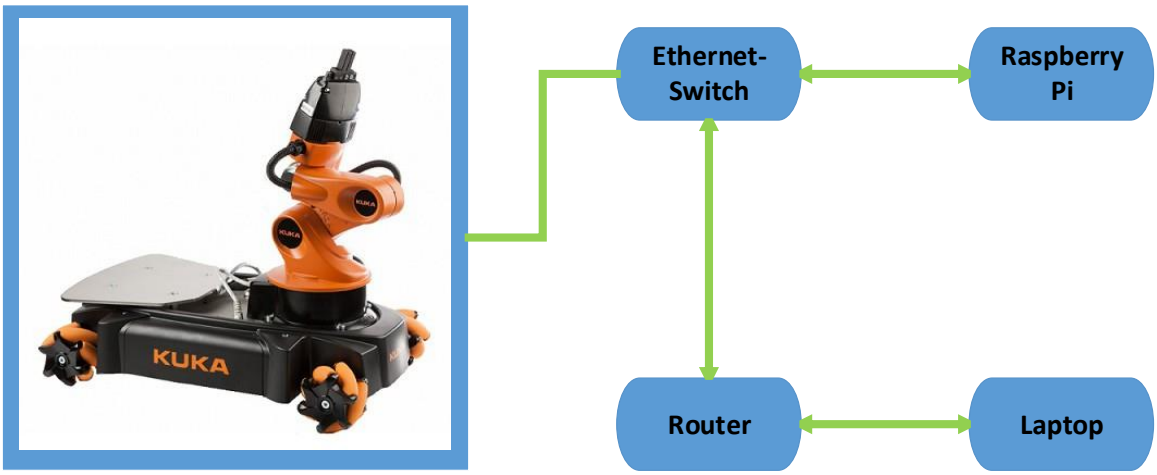

Abbildung 5.1.: schematische Verbindung der Systeme via Ethernet LAN

Physikalische Installation:

- Zuallererst muss der Roboterarm mit der Basisplattform verbunden werden. Danach können beide Komponenten gemeinsam oder einzeln angesteuert werden 5.2.
- Als nächstes sollte der Ethernet-Switch durch ein Ethernet-Kabel mit der Ethercat-Buchse des Roboters verbunden werden. Die Spannungsversorgung für den Switch und den Raspi übernimmt der USB Port des Roboters, jeweils mit einem USB-Kabel.
- Letztlich wird der Raspberry Pi wird per Ethernet-Kabel mit dem Switch verbunden.

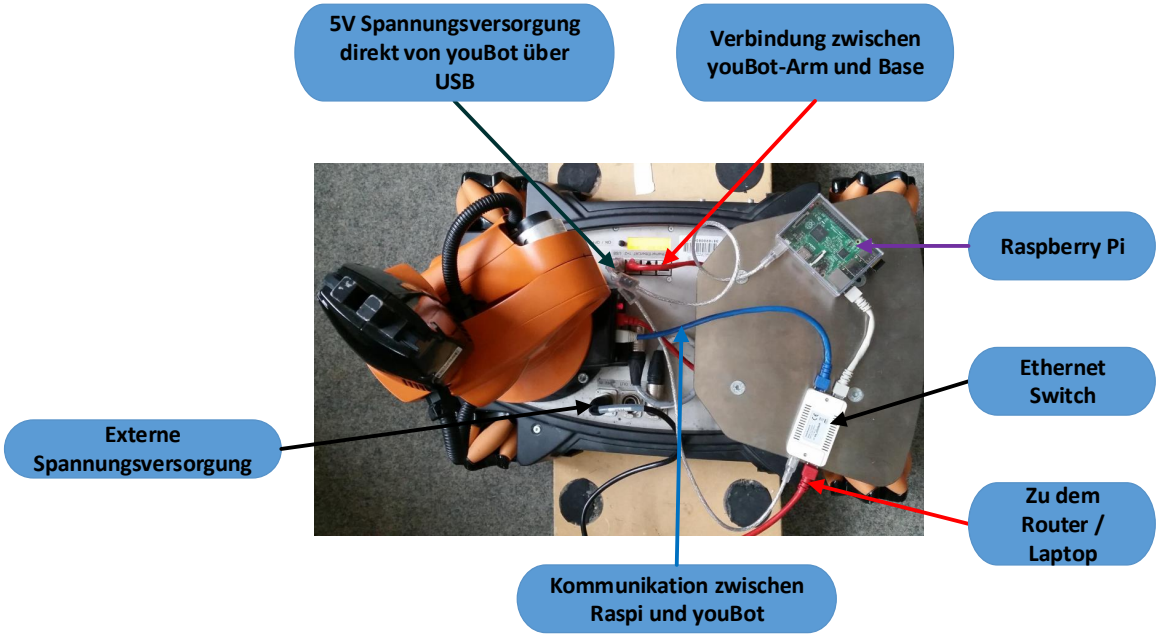

Abbildung 5.2.: Die Verkabelung der Komponenten bei der Verbindung via Ethernet LAN

Hier wird der so gestaltet, dass der Raspberry Pi mit einem externen Rechner kommunizieren kann. In diesem physikalischen Aufbau hat der Raspberry Pi keinen eigenen Bildschirm, was das Ausführen von Testszenarien erschwert. Über die Ethernet Verbindung zum Steuerungslaptop kann nun jedoch die Kommunikation mit dem Ubuntu Linux auf dem Raspi über einen SSH Terminal hergestellt werden, so dass Steuerungsbefehle an das ROS weitergeleitet werden können. Alternativ zum Ethernet-basierten Aufbau mit Switch besteht die Möglichkeit, den Raspberry Pi über WLAN mit einem Router zu verbinden, welcher dann via LAN oder WLAN wiederum mit dem Steuerungslaptop verbunden ist (5.3).

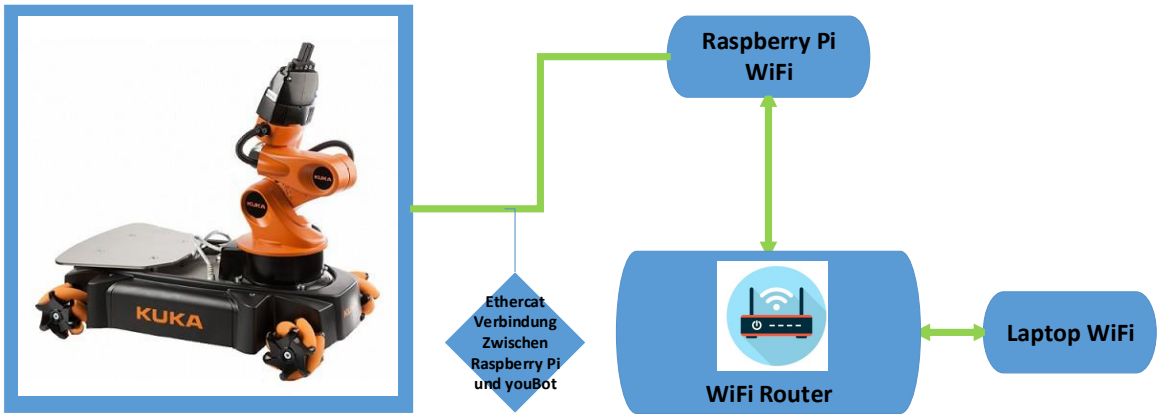

Abbildung 5.3.: Zeichnung für youBot mit Raspberry Pi Wifi

5.2. Umsetzung mit MATLAB und ROS

Wie bereits in Abschnitt 5.1 erklärt wurde, muss eine physikalische Verbindung zwischen Roboter und Raspberry Pi entstehen. Der ROS-Master (Siehe Abschnitt 3.4.1) wird direkt auf dem Raspi laufen. Wenn der Master aktiv ist, kann MATLAB mit dem ROS-Netzwerk kommunizieren (Siehe Abschnitt 3.4.4). In dem Raspberry Pi muss zuerst **catki_ws** kompiliert und **setup.bash** (Siehe Abschnitt 3.2) aufgerufen werden. Dann wird mit dem **roslaunch youbot_driver_ros_interface** Befehl der youBot mit dem ROS-Master verbunden. Sobald alle Verbindungen hergestellt sind, kann der youBot über MATLAB getestet werden. Um den youBot mit ROS zu testen, wird ein Code-Beispiel verwendet. Der Quellcode ist im Anhang gelistet (siehe A.1). Mit der Datei youbot.m werden Arm, Base und Greifer des youBot Roboters gesteuert. Die Steuerung erfolgt über das ROS-Netzwerk. Dieser Quellcode ist nur für Testzwecke benutzt worden [22]. Die youbot.m Datei besteht aus mehreren Funktionen, die für die Steuerung des Roboters zuständig sind. Sie schickt die Daten über das ROS-Netzwerk an den Roboter.

- **classdef Youbot < handle:** Hier beginnt die Youbot Klassendefinition. Der Dateiname und der Klassenname müssen identisch sein.
- **rospublisher:** ROS-Publisher versendet die Nachrichten an die zugehörigen **rostopics** über das ROS-Netzwerk. In der youbot.m Datei sind drei rospublisher Methoden zur Verfügung gestellt. Sie sind unabhängig für den Arm, die Base und den Greifer nutzbar. Der rospublisher erstellt ein Publisher und setzt den Topic-Name. Das Topic muss hierfür bereits in dem ROS-Master vorhanden sein [18].
- **rostopics:** Ruft Informationen über die ROS-Topics ab. Wenn der Roboter am ROS-Netzwerk angemeldet ist, können mit dem Befehl **rostopic list** angemeldete Topics im ROS-Netzwerk gelistet werden.

- **rosmessage:** Es erstellt die ROS-Nachrichten.
- **BaseVelocity:** Die Roboterbase kann in der x, y Richtung fahren oder um die z-Achse drehen. Die Geschwindigkeit wird in m/s in der x und y Richtung angegeben. Für die Umdrehung wird die Eingabe in rad/s vorgenommen.
- **ArmPosition:** Diese Funktion enthält einen Vektor, welcher aus 5 Elementen besteht. Jedes Gelenk des Manipulators kann einzeln gesteuert werden. Alle Einheiten müssen in rad sein.
- **StowArm:** Der Arm wird zusammengeklappt.
- **MoveGripper:** Der Greifer kann in eine bestimmte Position gefahren werden. Hier ist Werteingabe in m.
- **OpenGripper und CloseGripper:** Öffnet und schließt den Greifer.
- **Stop:** Stoppt der Roboter.

5.3. youBot Simulinkblöcke

In der Abbildung 5.4 wird das Simulink-Modell für die Ansteuerung des youBots dargestellt. Diese Steuerung besteht aus vier Teilen. Der erste Teil ist für die Echtzeitsynchronisation (Real Time Synchronisation), welche dem Roboter hilft, in annähernd Echtzeit zu arbeitet. Hierdurch werden Totzeiten der Gelenke so gut wie möglich reduziert. Der zweite Teil, der "Base Controller", ist für die Ansteuerung der youBot-Plattform. Vorgaben in diesem Controller-Block bedingen nur die Bewegungen der Plattform. Die gewünschten Bewegungen werden in x-Richtung über q1 und in y-Richtung über q2 eingegeben, es sind lineare Bewegungsvorgaben in m/s. Die Rotation der Plattform um die Z-Achse wird über q3 in rad/s vorgegeben. Der dritte Teil, der "Arm Controller" ist für die Steuerung des Roboterarms. Jedes der fünf Armgelenke ist rotatorisch und wird in Radiant vorgegeben. Der vierte Teil, der "Greifer Controller" steuert den Greifer an. Hierüber lässt sich der Greifer öffnen und schließen. Die Eingabe für die Greiferbackenposition ist in Metern vorzugeben.

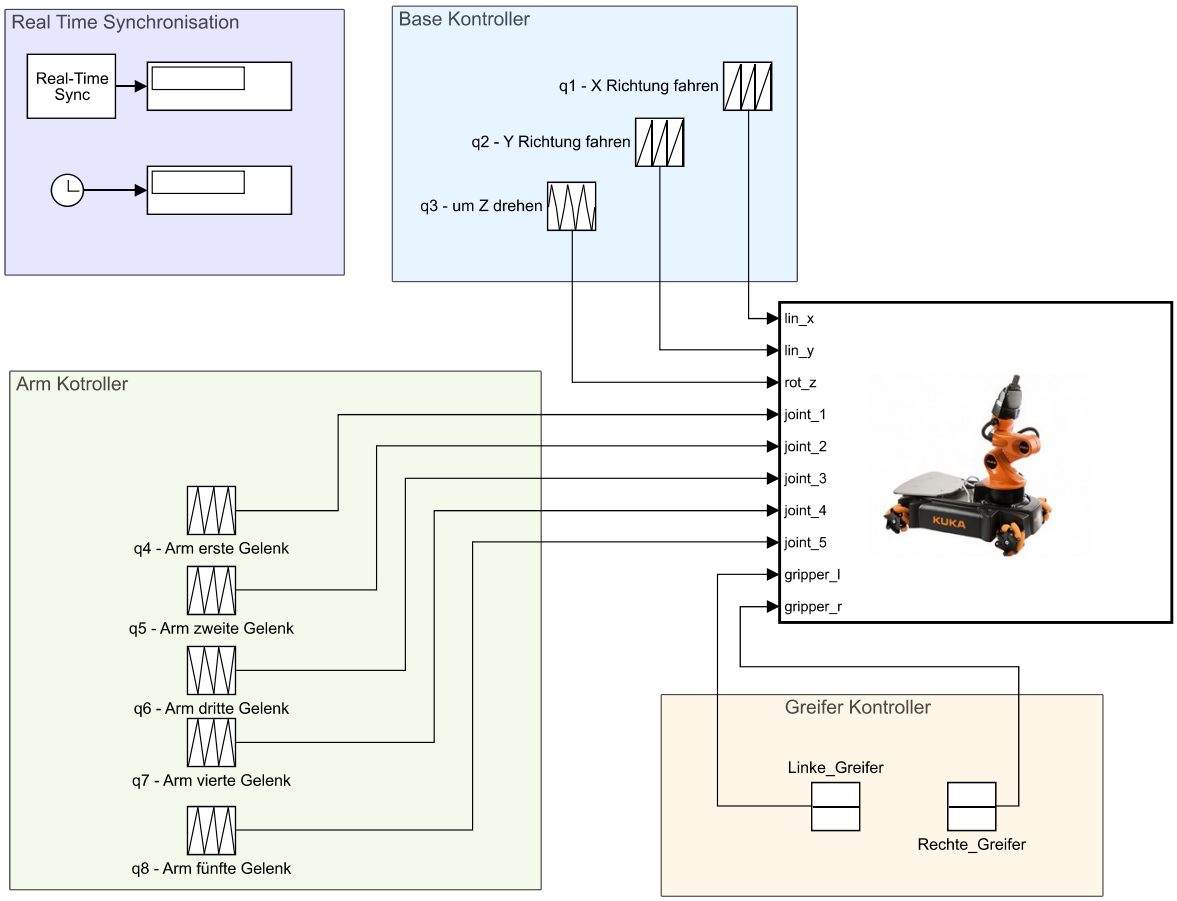

Abbildung 5.4.: youBot Simulink Steuerungsblöcke

5.4. Robotersteuerung mit Simulink

Das Steuerungsmodell für den youBot Roboter wurde in Simulink zusammengestellt. Dieses System besteht aus verschiedenen Blöcken, welche den physikalische Roboter und die Simulation steuert. In der Abbildung 5.5 ist das Simulink-Modell dargestellt.

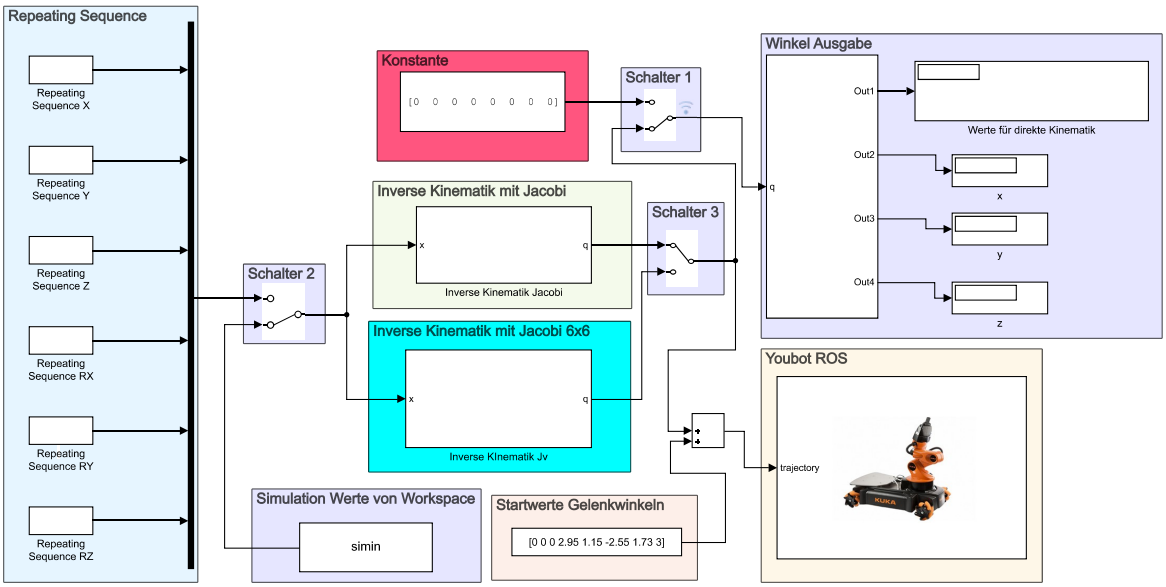

Abbildung 5.5.: youBot Steuerung mit Simulink

Die dargestellte Abbildung 5.5 besteht aus verschiedene Simulinkblöcken. Der **Konstante** Block gibt feste Gelenkwerte an das Simulink-Modell. Damit kann jedes Gelenk des Roboters in der Simulation getestet werden. Durch die **Repeating Sequence** Blöcke werden die kartesischen Werte für die Position und Orientierung des Endeffektors vorgegeben. Die Repeating Sequence hat einen Zeitvektor (t), welcher in der "parameter.m" Datei definiert ist. Der **Simin** Block bezieht die von der "generate_trajectory.m" generierten und im MATLAB-Workspace hinterlegten Trajektorienwerte. Diese Werte werden von hier aufgerufen und an den nächste Block weitergegeben. Schalter 2 ist für die Umschaltung im Simulink-Model zwischen den unterschiedlichen Dateneingängen. Über diesen Schalter kann zwischen den im Repeating-Sequence-Block eingetragenen Daten und denen aus dem Workspace gewählt werden. Hinter dem Schalter 2 gehen die Daten in die für die inverse Kinematik verantwortlichen Funktionen. Mittels der Jacobimatrix werden in rekursiver Methode die kartesischen Koordinaten in direkte Gelenkstellungen umgerechnet. Der Block "inverse Kinematik mit Jacobi" beinhaltet eine 6x8 Matrix. Hier wird über die Jacobimatrix $\underline{J}_G = \begin{pmatrix} \underline{J}_{G,T} \\ \underline{J}_{G,R} \end{pmatrix}^{6x8}$ alle acht

Gelenke eine Stellung berechnet. Der Roboter kann die angegebenen Positionen durch diese Jacobimatrixberechnung im Raum anfahren. In der Abbildung 5.6 ist der Inhalt des Jacobimatrix Blocks angezeigt. Der nächste Block **inverse Kinematik mit Jacobi 6x6**, die ist

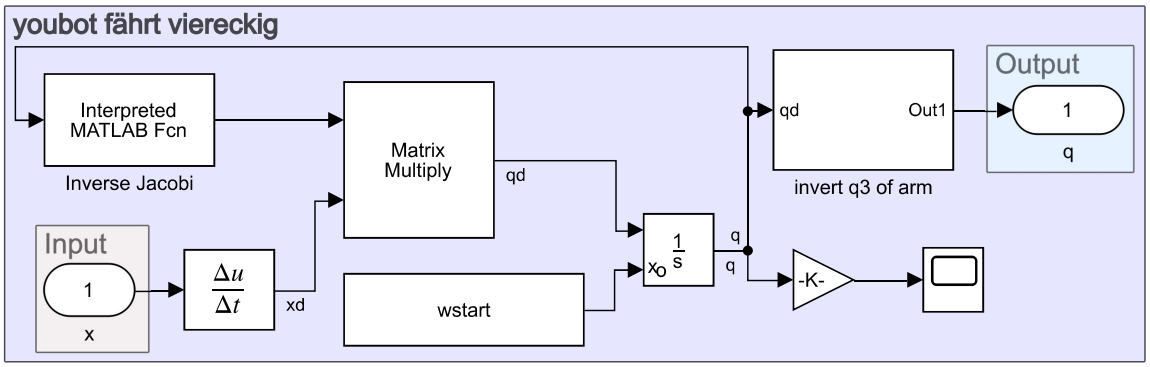

Abbildung 5.6.: Simulink-Modell für inverse Kinematik mit (6x8) Jacobimatrix

zuständig für das Halten die Pose des Endeffektors. In diesem inversen Kinematik-Block sind die zwei Schubachsen der Base vom Rest entkoppelt, in dem sie aus der Jacobimatrix genommen wurden. Hier wird erste zwei Schubachsen von berechneten Jacobimatrix entkoppelt. Die entkoppelten Schubachsen $\dot{q}_v = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}$ werden manuell eingegeben.
Die manuelle Vorgabe für q_1 und q_2 wird in der "parameter.m"Datei A.1.2 definiert und über Variablen in die Repeating-Sequence des Simulinkmodells geführt. In der Abbildung 5.7 sind die repeating-sequence-Bausteine "Winkel Vorgabe"(blau hinterlegt) von q_1 und q_2 zu sehen. Ebenfalls ist der restliche Inhalt des Subsystems der 6x6 Jacobimatrix dargestellt. Zu sehen ist, dass die zwei Daten von q_1 und q_2 anders als q_3 bis q_8 direkt an den Output gehen also nicht in die inverse Berechnung einfließen. Diese Werte sind bereits in der "parameter.m" Datei (A.1.2) eingegeben. Die Abbildung 5.7 zeigt den Inhalt dieses Blocks an. Der Schalter 3 aus Abbildung 5.5 entscheidet welche inverse Kinematik in der Simulation oder an dem echten Roboter werden verwendet soll.
Der **Startwerte Gelenkwinkel** Block kompensiert durch Addition die Aufstartwerte des realen Roboters mit denen des Simulink-Models, damit die Simulation und der youBot die gleiche Pose einnehmen. Der Block "youBot ROS" sendet die kompensierten Daten über ROS an den Roboter. Die Kommunikation zwischen ROS und dem Roboter erfolgt über "kuka_simulink_ros_action.m" (A.1.1) erfolgen. Der **Winkel Ausgabe** Block ist ein Subsystem in dem sich das Simulink-Modell für den youBot befindet. Das in der Abbildung 5.8 dargestellte Simulink-Modell ist, wie im Abschnitt 4.8.2 näher erläutert, direkt aus SolidWorks exportiert. Diese Thema ist bereits 4.8.2 erklärt.

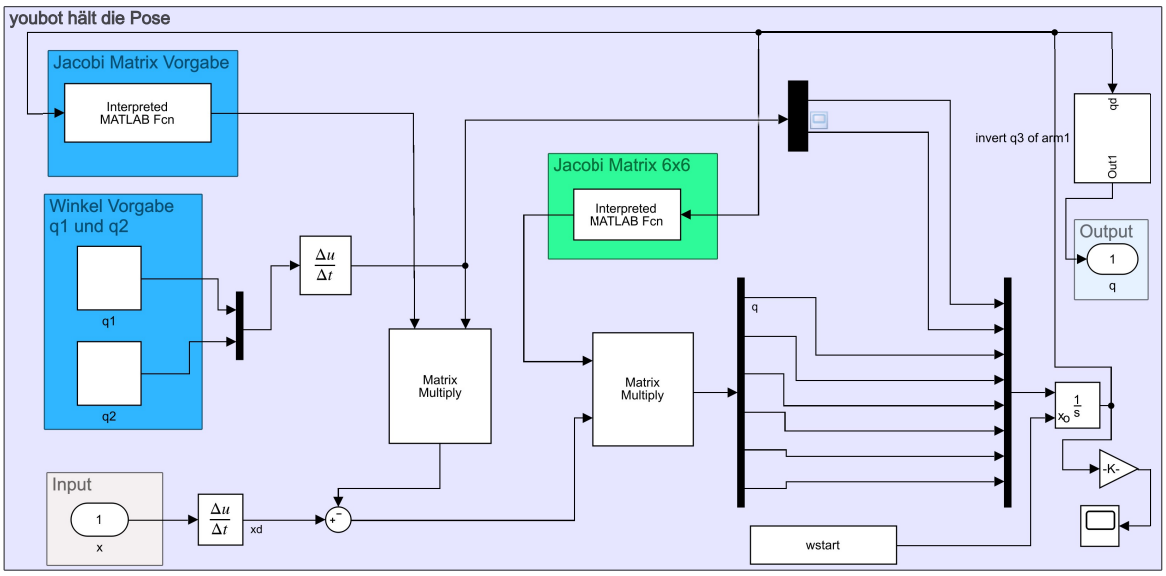

Abbildung 5.7.: Simulink-Modell für inverse Kinematik mit (6x6) Jacobimatrix

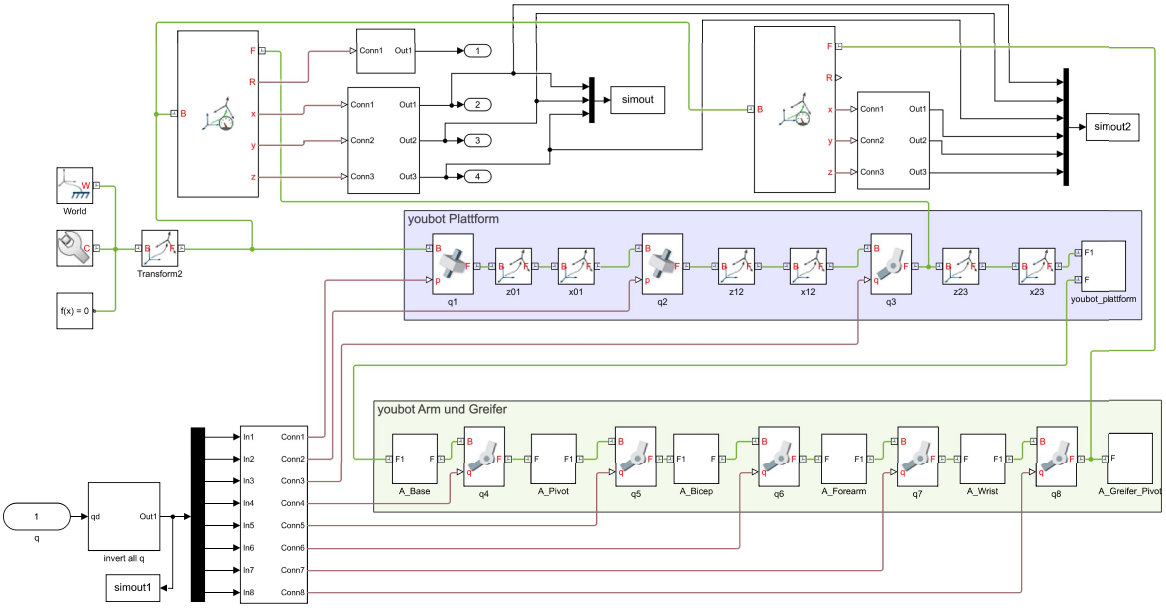

Abbildung 5.8.: Die Erweiterung des youBot Simulink-Modells

5.5. Plots der Viereckfahrt

Die Idee hinter diesem Szenario ist, dass der Roboter eine Viereck zeichnet. Zu diesem Zweck sind die indirekten Parameter in der Datei A.1.2 vorzugeben, welche dann über die Jacobimatrix in Gelenkstellungen umgerechnet werden. Die Simulationslaufzeit beträgt 10s um das Viereck zu beschreiben. Die eingegebene Kantenlänge ist $0.1x0.1m$. In der Simulation fährt der Roboter, mit seiner Plattform und seinem Arm zusammen, das Viereck aus den vorgegebenen Positionen. Die folgende Abbildung 5.9 zeigt, wie ein Viereck in der Simulation aussieht.

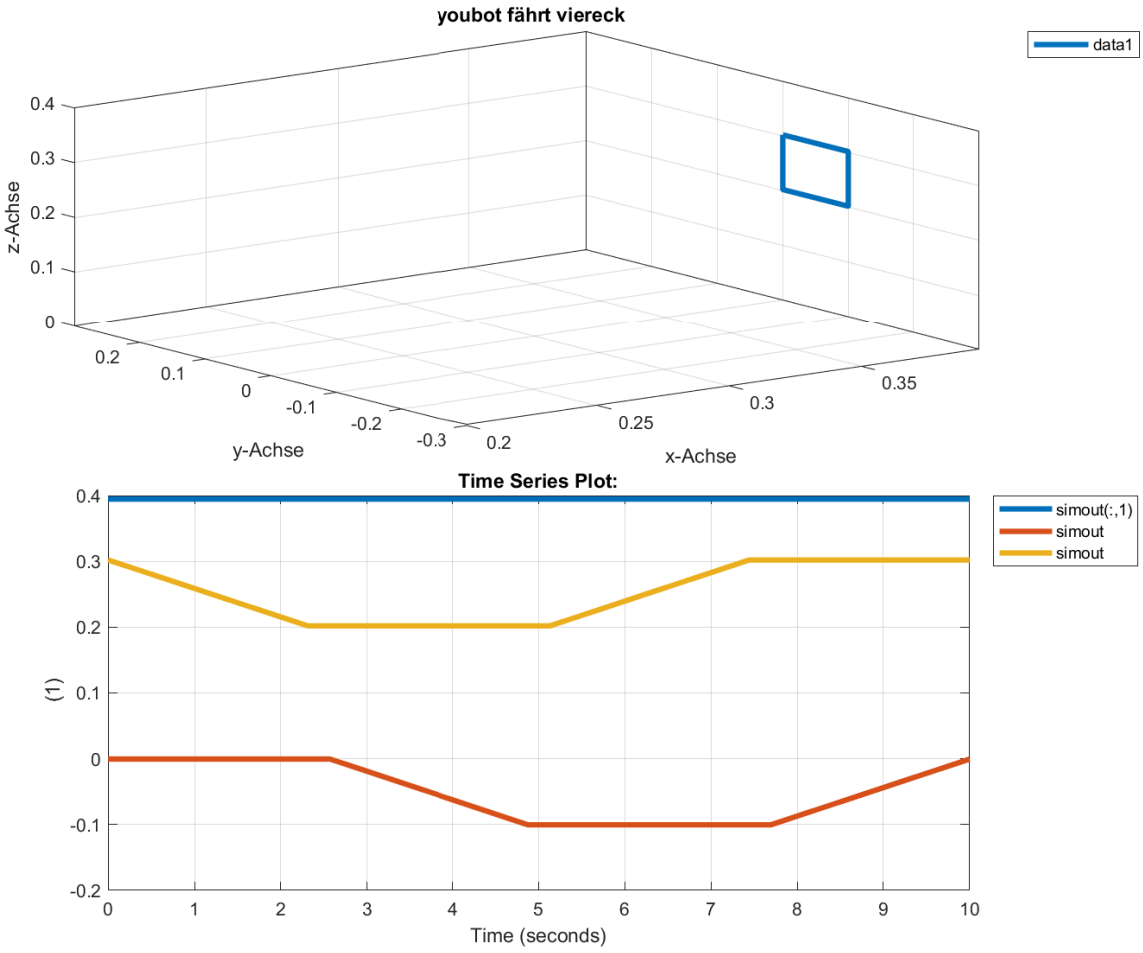

Abbildung 5.9.: youBot fährt ein Viereck

Der zweite Plot in der Abbildung 5.9 zeigt den zeitlichen Verlauf der kartesischen Koordinaten des Endeffektors. In dem zeitlichen Verlauf lässt sich erkennen, dass das Viereck in der YZ-Ebene beschritten wird. Der reale Roboter kann auf Grund seiner Motorregelparameter

kein richtiges Viereck zeichnen. Die Abbildung 5.10 zeigt, wie der reale Roboter ein Viereck mit dem Endeffektor beschreitet.

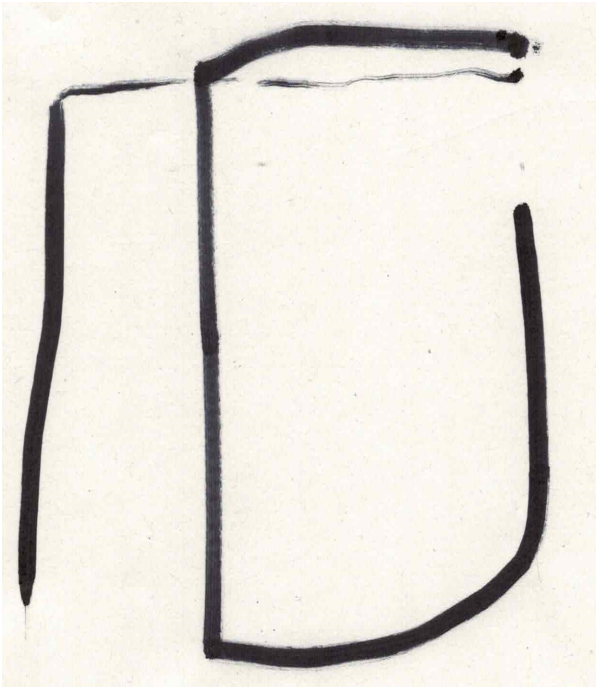

Abbildung 5.10.: Gezeichnetes Viereck des youBot

In der Abbildungen 5.11 und 5.12 sind der Viereckfahrt zugehörigen Achsen und Gelenkwinkel in MATLAB-Plots dargestellt.

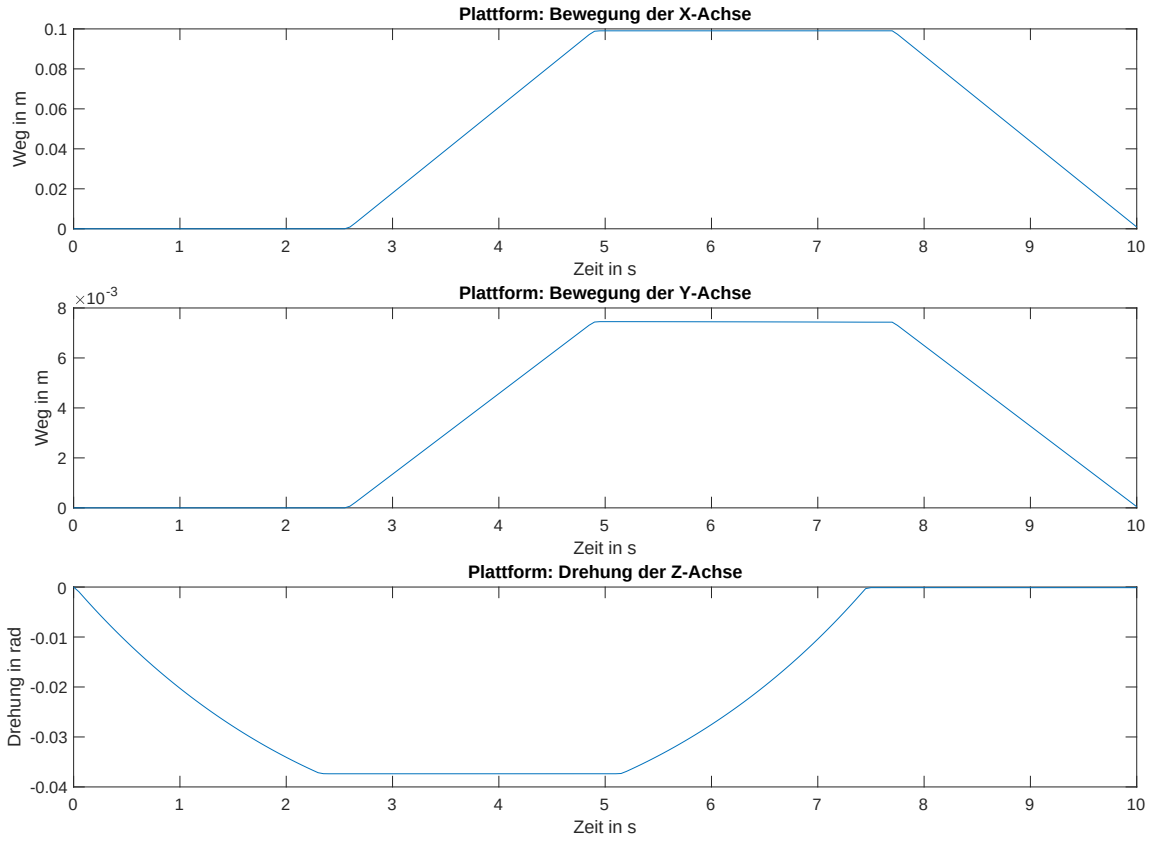

Abbildung 5.11.: youBot Plattform zeichnet ein Viereck

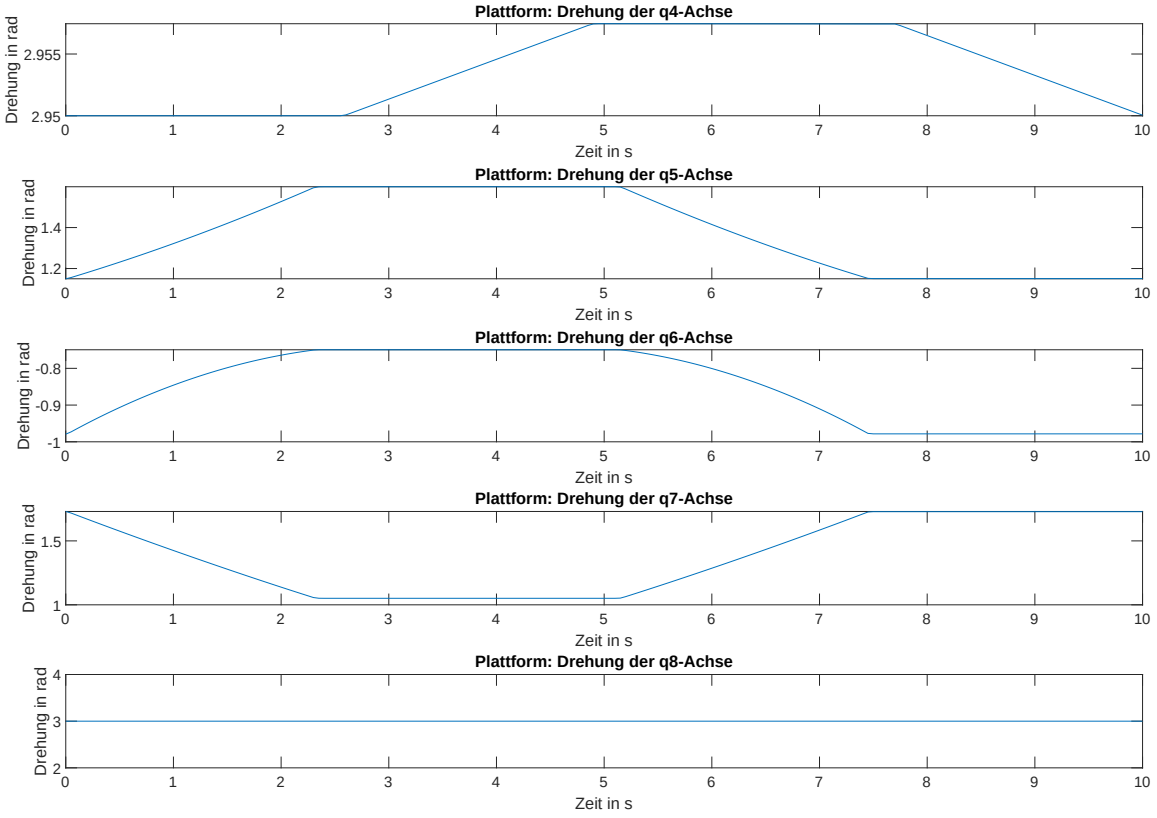

Abbildung 5.12.: youBot Arm zeichnet ein Viereck

5.6. youBot hält die Pose

In diesem Szenario soll der Endeffektor seine Pose beibehalten während die Plattform eine Bewegung ausführt. Dabei werden Schubachsen q_1 und q_2 vorgegeben. Die Simulationszeit beträgt 10s. Die vorgegebenen Schubachsen sind $q_1 = 0.05m$ und $q_2 = 0.05m$. Hier wird die Plattform in horizontaler Ebene bewegt und dabei der TCP in seiner Position und Orientierung nicht verändert. Dieses Szenario funktioniert in Simulation einwandfrei.

Bei dem realen Roboter gibt es jedoch Fehler. Der youBot empfängt die berechneten Daten aus ROS, dann versucht er seine Pose zu halten. Am Anfang der Bewegung bewegt sich die Base in xy-Richtung und der TCP hält seine Pose. Während der Bewegung werden einige Zwischenwerte empfangen, so dass der youBot sprunghafte Werte erhält. Dieses führt dazu, dass der TCP seine Position verliert. Nach solch einem Fehler versucht der youBot seine Position zu korrigieren. Nach dem Ende der Simulation, kommt der youBot nicht wieder an seine Anfangsposition zurück. Der Fehler zwischen Startposition und Endposition des Endeffektors beträgt ca. $20mm$. Das zeigt, dass die gesendeten Daten an den Roboter entweder zu viel sind und nicht geordnet verarbeitet werden können oder der Regler des nicht richtig Roboters funktioniert.

In der Abbildung 5.13 sind die Schubachsen dargestellt. Mit den eingegebenen Werten fährt der Roboter innerhalb von 10s auf der horizontale Ebene 5cm. Der Endeffektor hält dabei seine Pose.

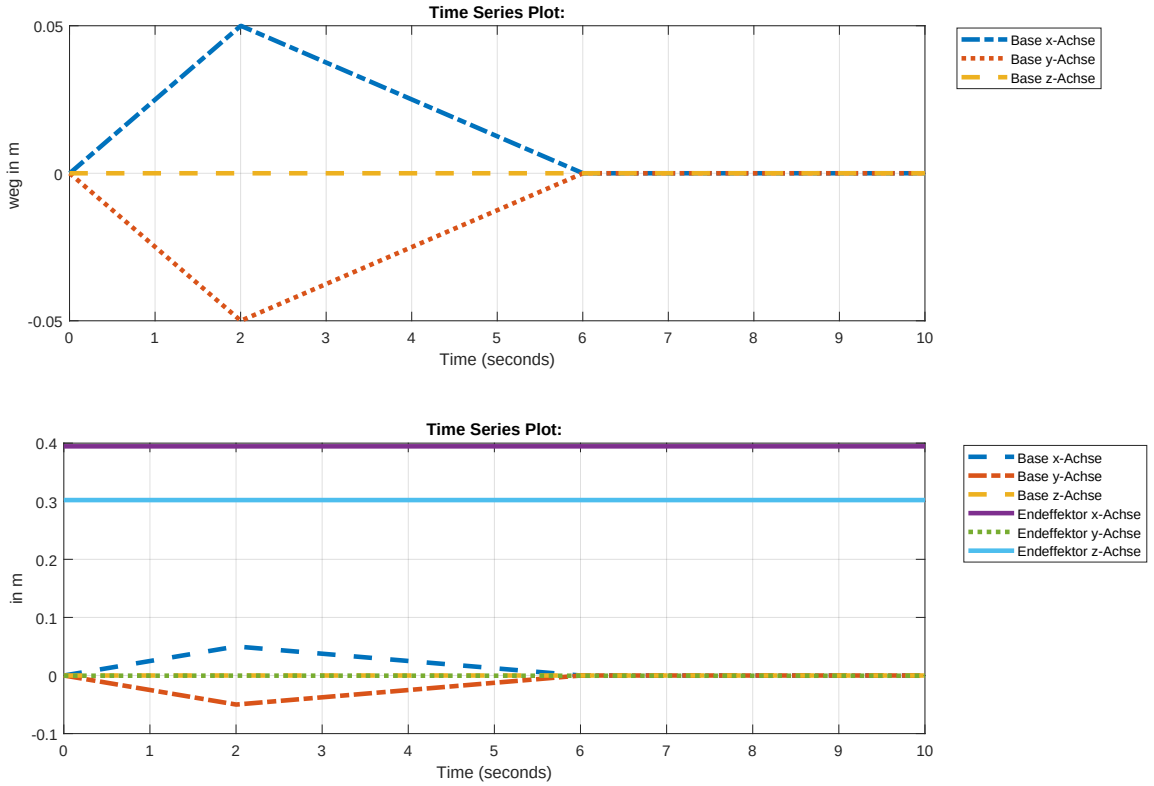

Abbildung 5.13.: youBot Endeffektor hält die Pose

In den Abbildungen 5.14 und 5.15 sind die zugehörigen Achsenvorgabe und Gelenkwinkel dargestellt.

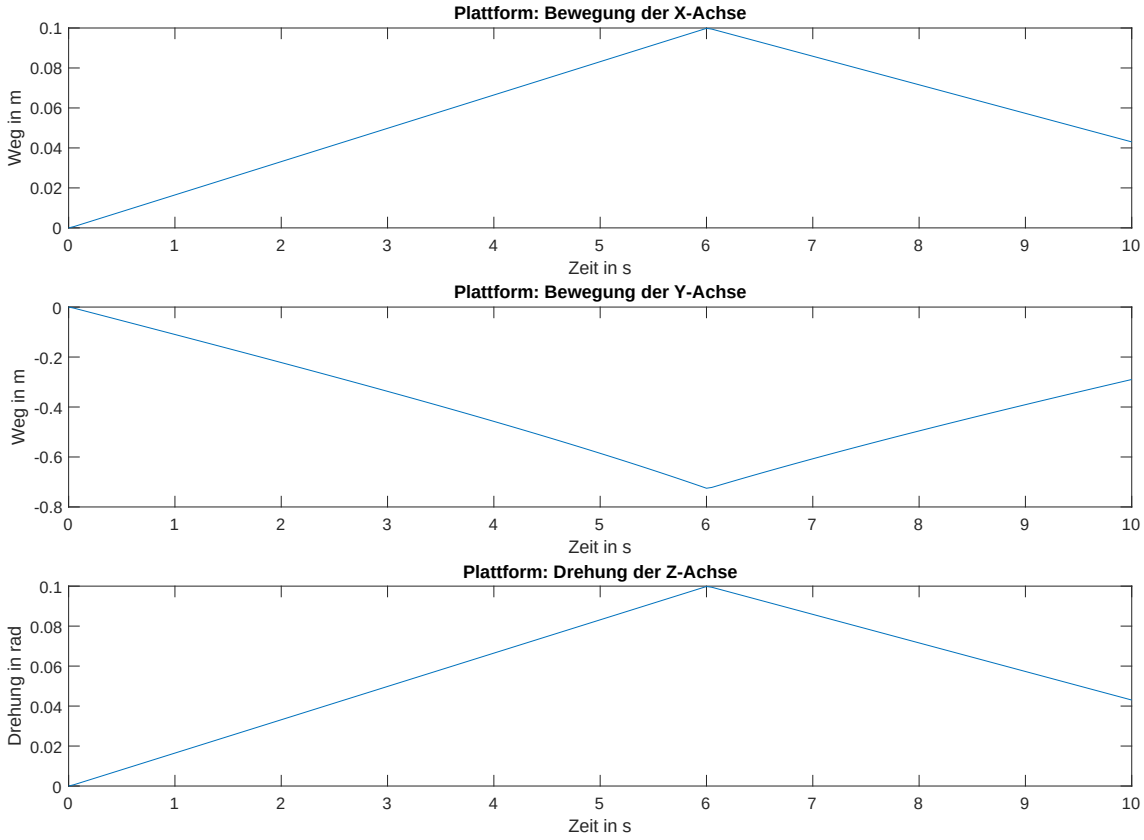

Abbildung 5.14.: Gelenkvorgabe für die youBot Plattform

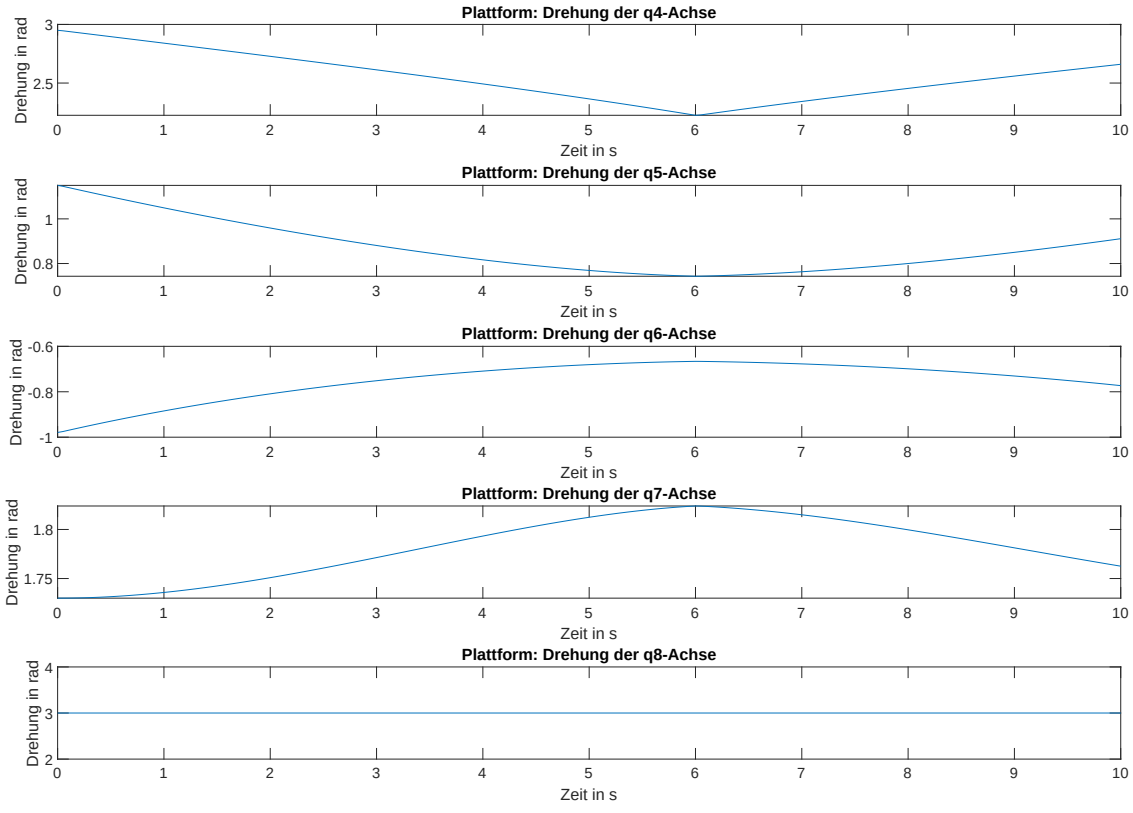

Abbildung 5.15.: Der youBot Arm hält die Pose

Die Abbildung 5.16 veranschaulicht die Kreisfahrt der Base. Dieses Szenario ist mit der Schubachsenvorgabe realisiert. Als Vorgabe ist hier eine Kreisformel, mit cos und sin in der "parameter.m" Datei eingegeben. Der berechnete Kreis wird an das Simulations-Modell geschickt. Als Kreisradius ist der maximal mögliche Radius von 0.07m genutzt. Mit diesem Wert kann der Roboter einen Kreis fahren, ohne dass der Endeffektor seine Position verlässt. Wenn der Kreisradius $\geq$ 0.075m ist, kann der Roboter keinen Kreis fahren und seine Pose nicht mehr halten. Hier ist das Problem, dass der eingegebene Werte größer ist, der youBot trotzdem versucht diesen Punkt zu erreichen. Dieser Punkt liegt jedoch außerhalb seines Arbeitsbereichs. Aus diesem Grund kann der youBot keinen Kreis fahren.

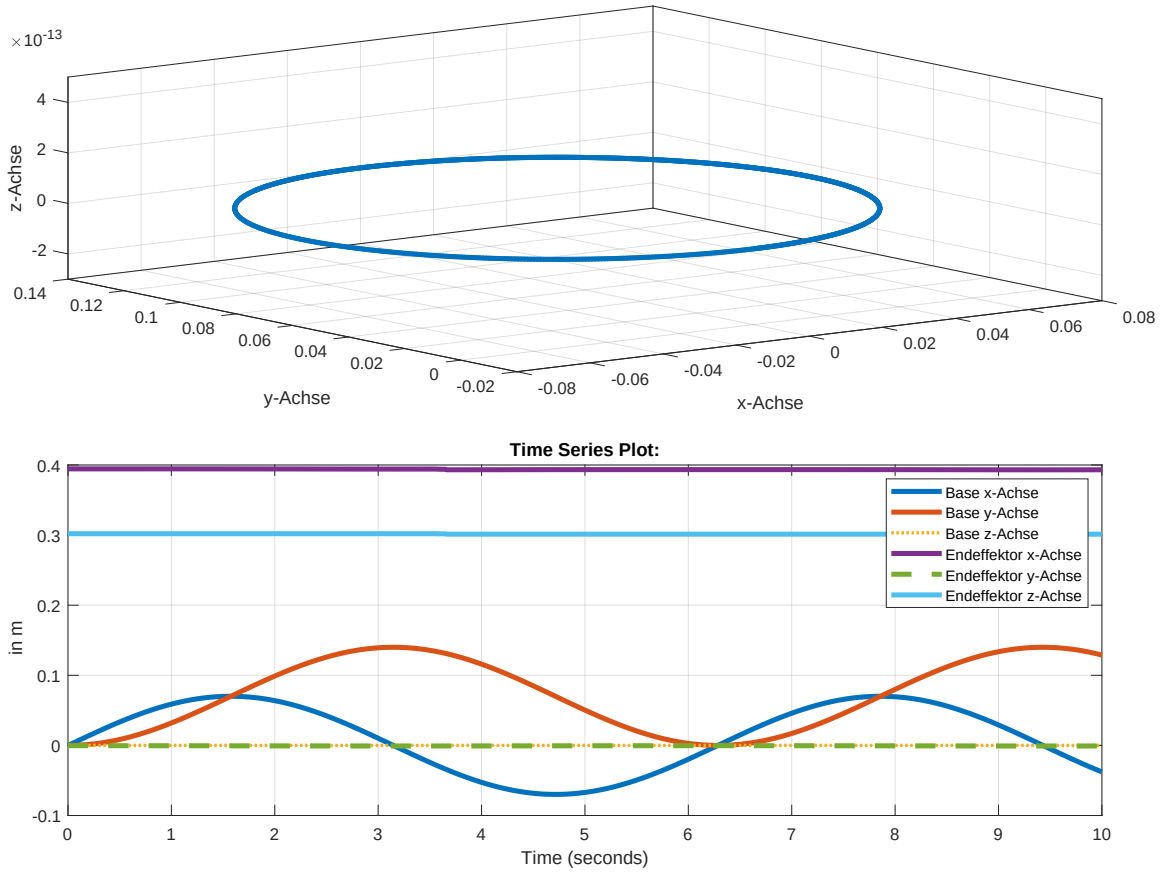

Abbildung 5.16.: youBot fährt einen Kreis

5.7. Überblick über die Quellcodes

In diesem Abschnitt wird ein Überblick über die Quellcodes geschaffen, welche in MATLAB Skripten verwendet wird.

5.7.1. MATLAB ROS Action Library

Der ROS-actionlib Stack bietet eine standardisierte Schnittstelle für die Anbindung von unterbrechbaren Aufgaben. Der ActionClient und ActionServer kommunizieren über ein "ROS Action Protocol", das auf ROS-Nachrichten aufgebaut ist. Der Client und der Server stellen eine einfache API bereit, mit der Benutzer Ziele (auf der Clientseite) anfordern oder Ziele (auf der Serverseite) über Funktionsaufrufe und Rückruffunktion ausführen können. In dieser Datei wurde eine Verbindung zwischen ROS Action Server und ROS Action Client (MATLAB) hergestellt. In der Datei A.1.1 kann auf die global definierten Variablen "youbotClient" und "youbotGoalMsgs" von MATLAB zugegriffen werden. Die "waitForServer" wartet, bis der Action-Server gestartet wird und kann Ziele senden. In der for-Schleife werden die Trajektoriendaten aus der Variable youbotTrajectory in der point Variable gespeichert. Die Trajektorie Daten werden an den ros_action_server geschickt (A.1.1).

5.7.2. Parameters für den Roboter

Hier sind die Startparameter für den Roboter eingetragen. Diese Parameter werden beim Simulationsstart geladen. Diese Datei enthält die Parameter für die Schubachsenvorgaben, Zeitkonstanten, DH-Parameter, Längen und Winkeln. Diese Datei A.1.2 kann beliebig modifiziert werden.

5.7.3. Trajektorie generieren

Diese Datei generiert die Trajektorie für den Roboter. Die MATLAB-Funktion "generate_trajctory.m" hat drei Parameters, welche "positions", "time" und "xstart" Werte erwartet. Die Trajektorie wird Anzahl der Punkten, zwischen zwei Punkten gefahren. Zu Beginn wird die Startposition der Strecke vorgegeben. Die Trajektorie wird schrittweise aufgebaut. Die erste for-Schleife generiert die alle Punkten (x,y,z,alpha,beta und gamma). Danach wird die Endposition der Strecke generiert. Die zweite for-Schleife berechnet die Strecke und speichert diese in einem Puffer. Die aktuelle Strecke wird an die Trajektorie angehängt. Die Endposition der Strecke ist die Startposition des nächsten Punkts. Wenn der Roboter die vorgegebene Trajektorie gefahren ist, soll er in die Startposition zurückkehren. Mit t wird

ein Zeitvektor erzeugt. Der Vektor wird jeder Trajektorie angehängt A.1.3. Diese Wert sind bereits in "parameter.m" eingegeben.

5.7.4. Der youBot Arm fährt ein Viereck

Hier werden die Werte für die Position des Endeffektors eingegeben. Der Roboter kann seine Gelenkwinkel für die vorgegebene Position berechnen A.1.4. Die vorgegebene Positionen kann der Roboter auf drei Wegen erreichen. Der erste Weg ist nur mit dem Arm, der zweite Weg ist mit der Plattform und der dritte ist mit Arm und Base zusammen. Die Positionen werden in der "generate_trajectory.m" Datei geladen und in dem MATLAB Workspace unter der **simin** Variablen gespeichert. Diese Datei kann beliebig modifiziert werden. Hier können neue Werte für den Roboter ausprobiert werden.

5.8. youBot rqt Graph

rqt ist ein Qt-basiertes Software-Framework für die Graphical User Interface Entwicklung für ROS. In diesem Paket sind verschiedene GUI-Tools in Form von Plugins implementiert. rqt besteht aus drei Teilen bzw. Metapackages.

- rqt
- rqt_common_plugins
- rqt_robot_plugins

Der rqt [35] Befehl kann direkt in der Kommandozeile eingegeben werden. Nach dem Ausgeführt eines Befehls wird ein GUI-Fenster eingeblendet. In diesem können verschiedene Plugins im ROS-System ausgewählt werden. rqt_common_plugins Metapackage stellt eine ROS-Backend-Tool-Suite für grafische Anwendungen zur Verfügung, die für die Roboterlaufzeit verwendet werden können. rqt_robot_plugins wird angewendet, wenn der Roboter im Einsatz ist.

In der Abbildung A.1 ist ein "rqt-Graph" für den youBot erstellt. Dieses Diagramm zeigt, welche Nodes und Topics in dem System aktiv sind. Die elliptisch dargestellten Formen sind ROS-Nodes, welche Programme in ROS ausgeführt werden. Die Namen in den quadratischen Formen sind Topics, welche mit einem Namespace dargestellt werden. Topics interagieren mit dem Nodes. In der nächsten Abbildung A.2 ist ein rqt-Graph nur mit Nodes erstellt. Nodes kommunizieren miteinander durch den Topic. Hier ist detailliert zu sehen, welche Topics genau mit welchem Nodes interagieren.

6. Zusammenfassung und Ausblick

Die mobile Robotik ist ein vielfältiges Thema, das großes Interesse bei Unternehmen und Entwicklern geweckt hat. In diesem Gebiet gibt es zahlreiche bereits verfügbare Produkte und viele Forschungsprojekte. In der Industrierobotik standen bislang Kriterien wie Wiederholgenauigkeit, Kraft und Schnelligkeit im Mittelpunkt. Die Zukunft der Fabrik fordert Mobilität, Anpassungsfähigkeit, Selbstständigkeit und Flexibilität. Anstatt einprogrammierter Befehlsfolgen für die Lösung ein und desselben Problems, werden dem Roboter nur Regeln vorgegeben, mit denen er die ihm gestellten Aufgaben autonom lösen kann. Die Grundlage dafür, dass der Roboter selbstständig seine Handlungen in Echtzeit planen kann, bildet seine Wahrnehmung mittels Sensoren, z.B. mit Kameras, Triangulationssensoren, Laserscannern oder Ultraschall- und Infrarot-Entfernungsmessgeräten. Um sich in einer ihm unbekannten Umgebung zurechtzufinden, erkundet der mobile Roboter zunächst seine Umwelt und legt das Wissen, das er während seiner Bewegungen erwirbt, in einer Art elektronischen Karte ab. Mit dieser Karte kann der mobile Roboter anschließend seinen optimalen Pfad planen. Das Neue an dieser Form der Robotik liegt also darin, dass der Roboter eine vorgegebene Aufgabe autonom löst und selbständig Entscheidungen treffen kann. Der Roboter kann folglich, mithilfe seines internen Weltbildes, auf wechselnde Anforderungen und Problemstellungen seiner dynamischen Umwelt reagieren und sein Verhalten dementsprechend anpassen, ohne Änderungen von außen am softwaretechnischen System.

Im Rahmen dieser Bachelorarbeit wurde der Roboter eigene Rechner gegen einen Einplatinenrechner (Raspberry Pi) getauscht. Der mit ROS installierte Computer kann mit dem youBot kommunizieren. Zur Initialisierung des Roboters werden ROS-Pakete für den youBot ausgeführt. Diese Pakete können die Sensordaten vom Roboter auslesen und auswerten.

Die inverse Kinematik für das überbestimmte System, wie beim youBot, konnte nicht per Hand berechnet werden. Für solche komplexen Berechnungen wurde MATLAB angewendet. Hier wurde die Jacobimatrix für die inverse Kinematik des Endeffektors mit MATLAB berechnet. Die Ergebnisse der Berechnungen werden an die zugehörigen Simulinkblöcken mit Hilfe des MATLAB-Skripts geschickt.

Um den Roboter mit ROS zu steuern wurde ein Beispielprogramm A.1 angewendet. Mit Hilfe dieses Programms kann MATLAB mit dem Roboter über ROS kommunizieren. Die Gelenke und die Achsen wurden mit dem Simulinkblock 5.4 getestet. Diese zeigt, dass der Roboter über MATLAB steuerbar ist.

Als nächstes wurde der Roboter in SolidWorks Eins-zu-eins zusammengebaut, was ein nicht ganz unerheblichen Teil dieser Arbeit darstellt. Dieses eingebaute Modell, wie bereits in 4.8.2 erklärt, kann aus SolidWorks exportiert werden. Das hilft dem Anwender um die zu sparen, anstatt das gleiche Modell in Simulink zusammenzubauen. Das exportiertes Modell wurde in einer "Extensible Markup Language (XML)" Datei gespeichert und diese Datei wurde in MATLAB mit dem "smimport" Befehl importiert. Daraus wurde eine Datei mit Simulinkblöcken erstellt. Wenn die Simulation mit diesen Blöcken gestartet wird, soll das Robotermodell sich ohne weitere Eingaben in der Simulationsumgebung "Simulink" bewegen. Das in der Abbildung 5.8 bereits dargestellte exportierte Modell aus SolidWorks wurde mit weiteren Blöcken erweitert. Das heißt, dass der Anwender jedes Gelenk des Roboters steuern kann. Während der Steuerung des Roboters ist bekannt geworden, das der Regler vom Roboter schwingt. Dieses Problem kann gelöst werden, wenn die Regelung für den Roboter neu geschrieben wird.

Als Resultat lässt sich festhalten, dass das gesamte System mit der Jacobimatrix zur Berechnung der inverse Kinematik realisiert und das Robotermodell erweitert sowie in die Simulationsumgebung eingebettet wurde. Zudem wurde eine Schnittstelle von Simulink/MATLAB auf den youBot eingerichtet. Die einfache Vorgabe der Schubachsen q_1 und q_2 war nicht ausreicht um die Plattform unabhängig vom Arm zu bewegen. Zum erreichen des gewünschten Verhaltens mussten die Schubachsen aus der inversen Kinematik entkoppelt werden.

6.1. Ideen zur Verbesserung des Systems

Das Steuerungssystem des Roboters funktioniert, solange er nicht zu viele Werte kriegt. Das verursacht einige Probleme bei der Regelung. Der Roboter soll in der Zukunft in verschiedenen Umgebungen fahren und mit Menschen oder mit Objekten interagieren können. Es sollte noch einiges an dem Roboter verbessert werden. Die Vorschläge zur Verbesserung sind:

- Der Roboterarm und die Plattform können zwar jetzt einige Trajektorien nachfahren, aber es fehlt eine richtige Bahnplanung. Um zum Beispiel seine Pose besser halten zu können, so dass der Arm alle Module gleichzeitig bewegt.
- Die jetzigen Reglerparameter verursachen plötzliche Sprünge in der Plattform. Das bringt den Fehler in der Positionsgenauigkeit. Um diesen Fehler zu vermeiden, soll der Regler für die Plattform neu geschrieben werden.
- Das Programm für die Gelenksteuerung des Manipulators funktioniert nicht immer einwandfrei. Es wäre besser, wenn Steuerungsprogramm untersucht wird, gegebenenfalls sollte hier ein neues geschrieben werden. Dies löst ein anderes Problem aus. Die Gelenkwerte kommen ständig an deren Limits, die Ströme in den Gelenken werden

zu hoch, daher steigen die Gelenke aus und können nicht mehr gesteuert werden. In solch einem Fall verliert der Roboter die Kommunikation zum Robot Operating System.

In der Zukunft kann die Hardware und die Software des Roboters beliebig erweitert werden. Als weitere Entwicklungsideen können die verschieden Optionen in Betracht gezogen werden.

- Mit der Visualisierungssoftware Gazebo [9] kann der gesamte Roboter in der Simulation nachgebaut werden. Es bringt den Vorteil, ohne einen realen Roboter, in der Simulation mit einem visuellen Roboter zu kommunizieren. Dies ermöglicht es, dass manche Sensoren (Kamera oder Encoder [8]) auf dem Roboter eingebaut werden können. In der Simulationsumgebung kann der Roboter ohne Kosten mit verschiedenen Objekten erweitert werden. Im Anwendungsfall können Objekte mit einer Kamera erkannt oder der Weg mit einem Encoder gemessen werden. Der Gazebo Simulator ist in ROS integriert.
- Außer Gazebo gibt es andere Simulatoren, zum Beispiel die virtual robot experimentation platform [31], die so ähnlich wie Gazebo ist. Diese bietet die Möglichkeit verschiedene Roboter zu steuern. Der Unterschied zwischen Gazebo und virtual robot experimentation platform (v-rep) kann hier [15] recherchiert werden.
- Auf dem realen Roboter können weitere Hardwarekomponente angebracht werden wie Kamera, Ultraschallsensor und Laserscanner. Eine Kamera hilft Objekte zu erkennen. Zur Hinderniserkennung können ein oder mehrere Ultraschallsensoren eingebaut werden. Wenn sich der Roboter z.B. in einem Raum befindet, kann er mit Hilfe des Laserscanners die Lokalisation und Kartierung durchführen. Die erzeugten Daten könnten gespeichert werden und beim nächsten Anwendungsfall wieder verwendet werden.
- Der Einplatinenrechner bietet die Möglichkeit sich mit einem Mikrocontroller zu verbinden. Das heißt, wenn sich ein Sensor nicht an den Computer anschließen lässt, wird ein Mikrocontroller zwischen geschaltet. Er kann als Erweiterungsplatine verwendet werden.

Im idealen Fall ist eine zu große Komplexität des Systems zu vermeiden. Aus diesem Grund sollten alle diese Erweiterungen nacheinander integriert werden. Abhängig vom Anwendungsfall kann jedoch die Kombination von verschiedenen Sensoren auf dem vorhandenen Roboter aufbauen.

Literaturverzeichnis

[1] BODO HEIMANN, Tobias Ortmaier und Lutz R.: *Mechatronik*. Carl Hanser Verlag GmbH & Co. KG, 2015. – ISBN 978-3-446-44451-5

[2] CAROL FAIRCHILD, Dr.Thomas L. H.: *ROS Robotics By Example*. Packt Publishing, 2016. – ISBN 978-1-78217-519-3

[3] CLEARPATHROBOTICS: *ROS Tutorials*. 2014. – URL https://support.clearpathrobotics.com/hc/en-us/categories/200165835-ROS

[4] CRAIG, John J.: *Introduction to Robotics*. Pearson Education International, 2005. – ISBN 0-13-123629-6

[5] ERDOGAN, Cagri: *Anwendbarkeit des Robot Operating Systems und des Gazebo Simulators auf Robotikwettbewerbe*. 2015

[6] EXPO21XX: *KUKA Youbot Arbeitsbereich*. https://www.expo21xx.com/automation21xx/20317_st3_industrial_research/default.htm Zugriffsdatum: 15.12.2017

[7] FRISCHGESELL, Prof. Dr.-Ing. T.: *Robotik Vorlesung*. 2015

[8] GAZEBO: *Sensoren für Gazebo*. http://osrf-distributions.s3.amazonaws.com/gazebo/api/7.0.0/classgazebo_1_1sensors_1_1Sensor.html Zugriffsdatum: 20.06.2018

[9] GAZEBO: *Simulation für ROS*. http://gazebosim.org/ Zugriffsdatum: 20.06.2018

[10] HARDKERNEL: *Odorid C2*. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G145457216438 Zugriffsdatum: 17.12.2017

[11] HARDKERNEL: *Odroid*. http://www.hardkernel.com/main/main.php Zugriffsdatum: 17.12.2017

[12] III, Carl D. C. ; DUFFY, Joseph: *Kinematic analysis of robot manipulators*. Cambridge University Press, 2008. – ISBN 978-0-521-57063-3

[13] JOSEPH, Lentin: *ROS Robotics Projects*. Packt Publishing, 2017. – ISBN 978-1-78355-471-3

[14] KUKA: *KUKA Youbot*. http://lh3.ggpht.com/-4VZDwEyqBSU/TXfIbbmSdTI/AAAAAAAAAOI/X8TO-Y6XcR0/3J8G2615f_sq.jpg/
Zugriffsdatum: 15.12.2017

[15] LENKASPACE.NET: *Robotik Simulatoren Vergleich*. http://lenkaspace.net/blog/show/120
Zugriffsdatum: 20.06.2018

[16] LOCOMOTEC: *KUKA youBot User Manual*. 2012. – URL https://github.com/ChefOtter/youbot_documentation

[17] MARTINEZ, Aaron ; FERNÁNDEZ, Enrique: *Learning ROS for Robotics Programing*. Packt Publishing, 2013. – ISBN 978-1-78216-144-8

[18] MATHWORKS: *rospublisher*. https://de.mathworks.com/help/robotics/ref/rospublisher.html?searchHighlight=rospublisher&s_tid=doc_srchtitle
Zugriffsdatum: 29.05.2018

[19] MATHWORKS: *SolidWorks Plugin für MATLAB*. https://de.mathworks.com/products/simmechanics/download_smlink.html
Zugriffsdatum: 29.05.2018

[20] MATLAB: *MATLAB ROS*. https://de.mathworks.com/hardware-support/robot-operating-system.html
Zugriffsdatum: 24.12.2017

[21] MATLAB: *MATLAB ROS Netzwerk*. https://de.mathworks.com/hell/robotics/examples/get-started-with-ros.html
Zugriffsdatum: 27.12.2017

[22] MCAREE, Owen: *Control of a Kuka youBot from MATLAB using ROS and the Robotic System Toolbox*. https://github.com/omcaree/youBot_MATLAB. Abruf: 12.11.2017

[23] O'KANE, Jason M.: *A Gentle Introduction to ROS*. University of South Carolina, 2014. – URL http://www.cse.sc.edu/~jokane/agitr/. – ISBN 978-14-92143-23-9

[24] PAULUS, Jan: *youBot API*. https://janpaulus.github.io/main.html
Zugriffsdatum: 29.05.2018

[25] PI, Raspberry: *Raspberry Pi*. https://www.raspberrypi.org/
Zugriffsdatum: 12.12.2017

[26] PI, Raspberry: *Raspberry Pi*. https://www.raspberrypi.org/app/uploads/2016/02/IMG_4090.jpg
Zugriffsdatum: 15.12.2017

[27] PROF. OUSSAMA KHATIB, Prof. Bruno S. und: *Handbook of Robotics*. Springer, 2008. – ISBN 978-3-540-23957-4

[28] REZNIK, Max: *Simulation von reduntaten Roboter am Beispiel eines redundanten YouBot-Modells*. Hochschule für Angewandte Wissenschaften Hamburg. 2017

[29] ROBOCUP: *RoboCup at Work*. http://www.robocupatwork.org/
Zugriffsdatum: 12.12.2017

[30] ROBOCUP: *RoboCup Industrial*. http://www.robocup.org/leagues/16
Zugriffsdatum: 12.12.2017

[31] ROBOTICS, Coppelia: *virtual robot experimentation platform*. http://www.coppeliarobotics.com/
Zugriffsdatum: 20.06.2018

[32] ROS: *Raspberry Kinetic Installation*. https://wiki.ros.org/kinetic/Installation/Ubuntu
Zugriffsdatum: 22.12.2017

[33] ROS: *ROS*. http://wiki.ros.org
Zugriffsdatum: 20.12.2017

[34] ROS: *ROS Kinetic*. http://wiki.ros.org/kinetic
Zugriffsdatum: 20.12.2017

[35] ROS: *rqt GUI für ROS*. https://wiki.ros.org/rqt
Zugriffsdatum: 06.06.2018

[36] ROS: *ROS Tutorials*. http://wiki.ros.org/ROS/Tutorials. 2015

[37] SCHMIEDECKE, Christoph: *Aktor-Sensor-Simulation für Assistenzroboter*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2013. – URL http://edoc.sub.uni-hamburg.de/haw/volltexte/2014/.../MA_Schmiedecke.pdf

[38] SIZEMORE, Jim ; MÜLLER, John P.: *Matlab for Dummies*. John Wiles & Sons Inc, 2015. – ISBN 978-1-118-882010-0

[39] WIKIPEDIA: *Einplatinenrechner*. https://de.wikipedia.org/wiki/Einplatinencomputer
Zugriffsdatum: 12.12.2017

[40] WIKIPEDIA: *Hardware Abstraction*. https://en.wikipedia.org/wiki/Hardware_abstraction
Zugriffsdatum: 20.12.2017

[41] WIKIPEDIA: *Jacobi Matrix*. https://de.wikipedia.org/wiki/Jacobi-Matrix
Zugriffsdatum: 03.03.2018

[42] WIKIPEDIA: *Pseudo Inverse*. https://de.mathworks.com/hardware-support/robot-operating-system.html
Zugriffsdatum: 31.12.2017

[43] WIKIPEDIA: *Robotik*. https://github.com/omcaree/youBot_MATLAB
Zugriffsdatum: 11.12.2017

[44] WIKIPEIDA: *Klassische DH Konvention*. https://en.wikipedia.org/wiki/DenavitE28093Hartenberg_parameters
Zugriffsdatum: 03.06.2018

[45] WOERNLE, Prof. Dr.-Ing. C.: *Mehrkörpersysteme, Eine Einführung in die Kinematik und Dynamik von Systemen starrer Körper*. Springer Verlag, 2011. – ISBN 978-3-642-15981-7

A. Quellcode

A.1. youbot.m

```
1  classdef Youbot < handle
2      %  YOUBOT A class to handle ROS control of a Kuka youBot
3      %  Instansiate the object with the ROS namespace of the youBot,
           if one
4      %  has been set, otherwise no arguments are needed. For example
           :
5      %
6      %     myYoubot = Youbot('youbot1')
7      %
8      %  To move/orient the base of the youBot
9      %
10     %     myYoubot.BaseVelocity(xVel, yVel, omega)
11     %
12     %  where xVel and yVel are in meters/second and omega is
13     %  radians/second
14     %
15     %  To stop the youBot at any time
16     %
17     %     myYoubot.Stop()
18     %
19     %  To move the arm
20     %
21     %     myYoubot.ArmPosition(jointPositions)
22     %
23     %  where jointPositions is a 5 element vector containing the
24     %  individual joint positions in radians
25     %
26     %  To move the arm back to a stowed position
27     %
28     %     myYoubot.StowArm()
29     %
30     %  To move the gripper to a specific position
```

```
31  %
32  %     myYoubot.MoveGripper(distance)
33  %
34  %  where position is the distance to move each finger in meters
35  %
36  %  To open/close the gripper
37  %
38  %     myYoubot.OpenGripper()
39  %     myYoubot.CloseGripper()
40  %
41  %  Upon destruction of the object, the Stop and StowArm methods
        are
42  %  called automatically
43  %
44  properties (SetAccess = private)
45     Name
46  end
47  properties (Access = private)
48     ArmPublisher
49     BasePublisher
50     GripperPublisher
51  end
52  properties (Access = private, Dependent)
53     Namespace
54     ArmTopic
55     BaseTopic
56     GripperTopic
57  end
58
59  methods
60     function this = Youbot(varargin)
61        switch (nargin)
62           case 0
63              this.Name = '';
64           case 1
65              this.Name = varargin{1};
66        end
67        %Hier werden rospublisher die Publisher erstellen und die
            Topic-Name setzen.
68        this.ArmPublisher = rospublisher(this.ArmTopic);
69        this.BasePublisher = rospublisher(this.BaseTopic);
70        this.GripperPublisher = rospublisher(this.GripperTopic);
71     end
72     function path = get.Namespace(this)
```

```
73          if (isempty(this.Name))
74             path = '/';
75          else
76             path = ['/' this.Name '/'];
77          end
78       end
79       % Wenn youbot an ROS-Master bzw. ROS-Netzwerk angemeldet ist
            , kann mit der Befehl "rostopic list"
80       % gesamte Topcis gelistet werden.
81
82       function path = get.ArmTopic(this)
83          path = [this.Namespace 'arm_1/arm_controller/
               position_command'];
84       end
85       function path = get.BaseTopic(this)
86          path = [this.Namespace 'cmd_vel'];
87       end
88       function path = get.GripperTopic(this)
89          path = [this.Namespace 'arm_1/gripper_controller/
               position_command'];
90       end
91
92       function BaseVelocity(this, x, y, omega)
93          message = rosmessage(this.BasePublisher);
94          message.Linear.X = x;
95          message.Linear.Y = y;
96          message.Angular.Z = omega;
97          send(this.BasePublisher,message);
98       end
99       function Stop(this)
100         this.BaseVelocity(0,0,0);
101      end
102      function ArmPosition(this,armPosition)
103         if length(armPosition) ~= 5
104            return;
105         end
106         for i=1:length(armPosition)
107            joints(i) = rosmessage('brics_actuator/JointValue');
108            joints(i).JointUri = ['arm_joint_' num2str(i)];
109            joints(i).Value = armPosition(i);
110            joints(i).Unit = 'rad';
111         end
112         message = rosmessage(this.ArmPublisher);
113         message.Positions = joints;
```

```
114             send(this.ArmPublisher,message);
115         end
116         function StowArm(this)
117             this.ArmPosition([0.011 0.011 -0.016 0.023 0.12]);
118         end
119         function MoveGripper(this, position)
120             gripper(1) = rosmessage('brics_actuator/JointValue');
121             gripper(1).JointUri = 'gripper_finger_joint_l';
122             gripper(1).Value = position;
123             gripper(1).Unit = 'm';
124             gripper(2) = rosmessage('brics_actuator/JointValue');
125             gripper(2).JointUri = 'gripper_finger_joint_r';
126             gripper(2).Value = position;
127             gripper(2).Unit = 'm';
128             message = rosmessage(this.GripperPublisher);
129             message.Positions = gripper;
130             send(this.GripperPublisher,message);
131         end
132         function OpenGripper(this)
133             this.MoveGripper(0.0114);
134         end
135         function CloseGripper(this)
136             this.MoveGripper(0);
137         end
138 
139         function delete(this)
140             this.Stop();
141             this.StowArm();
142         end
143     end
144 
145 end
```

A.1.1. kuka_simulink_ros_action.m

```
function kuka_simulink_ros_action
   global youbotClient youbotGoalMsgs;
   [youbotClient,youbotGoalMsgs] = rosactionclient('/moveit/
      arm_controller/follow_joint_trajectory', 'control_msgs/
      FollowJointTrajectory');
   waitForServer(youbotClient);
   youbotGoalMsgs.Trajectory.JointNames = {'arm_joint_1','
      arm_joint_2','arm_joint_3','arm_joint_4','arm_joint_5','
      virtual_theta', 'virtual_x','virtual_y'};
   youbotGoalMsgs.GoalTimeTolerance = rosduration(0,500000000);
   points = rosmessage('trajectory_msgs/JointTrajectoryPoint');
   trajectory = evalin('base', 'youbotTrajectory');
   trj = trajectory.Data;
   for i=1:length(trajectory.Data)
      points(i) = rosmessage('trajectory_msgs/JointTrajectoryPoint
         ');
      points(i).Positions = [trj(i,4),trj(i,5),trj(i,6),trj(i,7),
         trj(i,8),trj(i,1),trj(i,2),trj(i,3)];
      points(i).Velocities = [0,0,0,0,0,0,0,0];
      points(i).Accelerations = [0,0,0,0,0,0,0,0];
      points(i).TimeFromStart = rosduration(trajectory.Time(i));
   end
   youbotGoalMsgs.Trajectory.Points = points;
   sendGoal(youbotClient,youbotGoalMsgs);
   %delete(youbotClient);
end
```

A.1.2. parameter.m

```
%Parameterfile Kuka Youbot
% Setzen des des zugehoerigen Startwertes fuer q
% in nicht singuliaerer Konfiguration;

% Startwerte üfr die Gelenke
wstart = [0 0 0 0 0 pi/2 0 0];
xstart = j2p(wstart);

%%
% Zeitkonstante
T = 10; % Simulationslaufzeit
dt = 0.01;
% t = [0 6 13 21]; % Zeitvektor
% t = [0 2 6 11]; % Zeitvektor

% parameter fuer youbot haelt die Pose
% vq1 = [0 0.1 0 0];
% vq2 = [0 0.1 0 0];

% parameter fuer youbot haelt die Pose
% vq1 = [0 0.1 -0.1 0];
% vq2 = [0 0.1 -0.1 0];

%%
% hier wird Kreisumfang berechnet
% cos und sin kreis
% circle = -2*pi:0.1:2*pi;
% yc = sin(circle);
% xc = cos(circle);
% plot(xc,yc);

% xc= sin(t*2*pi)/T;
% yc= cos(t*2*pi)/T;

% t = [0:0.01:20];
% xc= sin(0:0.01:20);
% yc= cos(0:0.01:20);

% die Plattform wird eine Kreisbewegung machen
% die Pose des Endeffektors wird behalten
% t = (0:dt:2*pi);
t = linspace(0,2*pi,128);
```

```
43 radius = 0.07;
44 xc= sin(t)*radius+radius;
45 yc= cos(t)*radius+radius;
46 vq1 = xc;
47 vq2 = yc;
48
49 % hier wird Kreisumfang berechnet
50 % vq1 = [0,xc(1:end-1)];
51 % vq2 = [0,yc(1:end-1)];
52 % vq1 = xc;
53 % vq2 = yc;
54 % plot(yc,xc)
55
56 % Kreisumfang wird berechnet
57 % t = linspace(0,2*pi,128);
58 % xc = cos(t);
59 % yc = sin(t);
60 % vq1 = xc;
61 % vq2 = yc;
62 % figure; patch( x, y, 'r' )
63 %%
64 % Gelenkkoordinaten (indirekt)
65 X = [  xstart(1)  xstart(1)     xstart(1) xstart(1)];
66 Y = [  xstart(2)  xstart(2)     xstart(2) xstart(2)];
67 Z = [  xstart(3)  xstart(3)     xstart(3) xstart(3)];
68 RX = [ xstart(4)  xstart(4)     xstart(4) xstart(4)];
69 RY = [ xstart(5)  xstart(5)     xstart(5) xstart(5)];
70 RZ = [ xstart(6)  xstart(6)     xstart(6) xstart(6)];
71
72 q1 = 0;
73 q2 = 0;
74 q3 = 0;
75 q4 = 0;
76 q5 = 0;
77 q6 = 0;
78 q7 = 0;
79 q8 = 0;
80
81 %% DH
82 %          Theta|       d |      a |    alpha
83 Matrix = [ 0    ,       q1,     0,      pi/2;
84            pi/2 ,       q2,     0,      pi/2;
85            q3   ,        0,     0.1505, pi ;
86            q4   ,   -0.267,     0.033,  pi/2;
```

```
87              q5-pi/2,      0,      0.155,    0;
88              q6,           0,      0.135,    0;
89              q7+pi/2,      0,      0,      pi/2;
90              q8+pi,    0.218,      0,        0;
91          ];
92
93  % parameter fuer das Simscape modell
94  Theta = Matrix(:,1)
95  d     = Matrix(:,2)
96  a     = Matrix(:,3)
97  alpha = Matrix(:,4)
98
99  % Plot Ausgabe fuer simout Variable
100 plot3(simout.Data(:,1),simout.Data(:,2), simout.Data(:,3))
101 grid on
102 box on
103
104 % Plot Ausgabe wird als pdf gespeichert
105 h=gcf;
106 set(h,'PaperOrientation','landscape');
107 set(h,'PaperPosition', [1 1 28 19]);
108 print(gcf, '-dpdf', 'youbot_plattform_faehrt_kreis.pdf');
109
110 % plot3(simout1.Time(:,1),simout1.Data(:,1), simout1.Data(:,2))
111 % plot3(simout1.Data(:,3),simout1.Data(:,4), simout1.Data(:,5))
112 % plot3(simout1.Data(:,6),simout1.Data(:,7), simout1.Data(:,8))
```

A.1.3. generate_trajectory.m

```
function simin = generate_trajectory(positions,time,xstart)
n = 10;
start_pos = [0,0,0,0,0,0];
trajectory=[];
for i=1:size(positions,1)
   end_pos = positions(i,:);
   simin1=[];
   for j=1:6
     simin1= [simin1;linspace(xstart(j)+start_pos(j),xstart(j)+
        end_pos(j),n)];
   end
   trajectory = [trajectory;simin1];
   start_pos = end_pos;
end
end_pos = [0,0,0,0,0,0];
      simin1=[];
   for j=1:6
      simin1= [simin1;linspace(xstart(j)+start_pos(j),xstart(j)+
         end_pos(j),n)];
   end
   trajectory = [trajectory;simin1];
t = linspace(0,time, size(trajectory, 1));
simin = [t trajectory];
end
```

A.1.4. youbot_drive_square.m

```
n = input('Enter a number:');

switch n
   case 1
% Die Plattform faehrt viereck, dabei Arm bleibt stehen.
positions = [0,0,-0.1,0,0,0;
          0,-0.1,-0.1,0,0,-0.1620;
          0,-0.1,0,0,0,-0.1620];
   case 2
% Roboter zeichnet quadrat mit plattform bewegung
positions = [0.1,0,0,0,0,0;
          0.1,0.1,0,0,0.1620,0;
          0,0.1,0,0,0.1620,0];

% Roboter zeichnet quadrat mit dem Plattform
% positions = [0,0,-0.1,0,0,0;
%           0,-0.1,-0.1,0,0,0;
%           0,-0.1,0,0,0,0];

   case 3
% endeffektor position einstellen mit euler winkel x,y,z, alpha,
   beta und
% gamma
% Roboter faehrt auf der Y Achse hin und üzurck
positions = [0,0.1,0,0,0,0.3;
          0,-0.1,0,0,0,0;
          0,0,0,0,0,0];
   case 4
% Arm zeichnet der Viereck und Plattform hilft dabei
positions = [-0.1,0,0,0,0,0;
          -0.1,-0.1,0,0,0,0;
          0,-0.1,0,0,0,0];
   case 5
% Endeffektor test positionswerte
positions = [pi/64,pi/64,pi/64,pi/64,pi/64,pi/64;
          pi/128,pi/128,pi/128,pi/128,pi/128,pi/128;
          0,0,0,0,0,0];
   case 6
% Plattform faehrt quadrat arm bleibt stehen
positions = [-0.1,0,0,pi/4,pi/4,pi/4;
          -0.1,-0.1,0,pi/4,pi/4,pi/4;
          0,-0.1,0,pi/4,pi/4,pi/4];
```

```
42
43 %----------------Pose halten----------------
44   case 7
45 % Die Pose wird gehalten, wenn die Werte q1 und q2 vorgegeben wird
46 positions = [0,0,0,0,0,0;
47           0,0,0,0,0,0;
48           0,0,0,0,0,0];
49 end
50
51 simin = generate_trajectory(positions,10,xstart);
```

A. Anhang

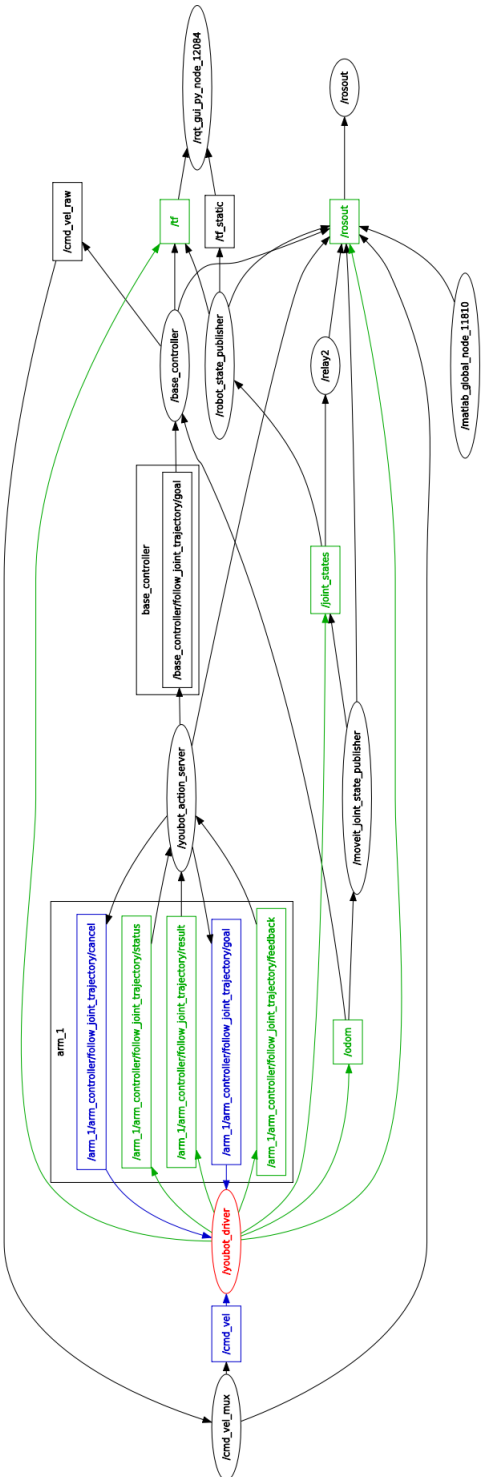

Abbildung A.1.: rqt Graph mit Topics und Namespaces für youBot

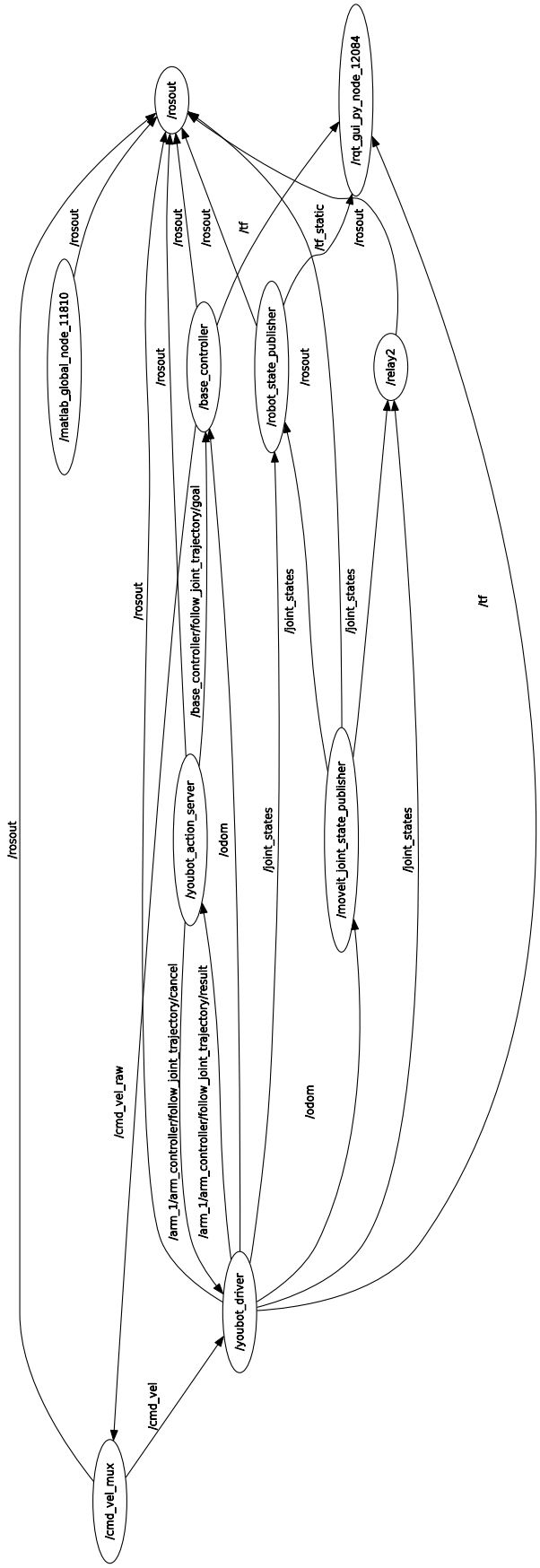

Abbildung A.2.: rqt Graph Nodes für youBot

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 28. Juni 2018	
Ort, Datum	Unterschrift