Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Stanislaw Rasowski

Tablet-basierte Steuereinheit für einen fahrbaren 6-DoF-Assistenzroboter

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Stanislaw Rasowski

Tablet-basierte Steuereinheit für einen fahrbaren 6-DoF-Assistenzroboter

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Eingereicht am: 30. Oktober 2015

Stanislaw Rasowski

Thema der Arbeit

Tablet-basierte Steuereinheit für einen fahrbaren 6-DoF-Assistenzroboter

Stichworte

Robotersteuerung, Roboterarm, Kinematik, ROS, Android, Assistenzroboter

Kurzzusammenfassung

Diese Arbeit beschreibt die Entwicklung der Steuereinheit auf Basis der Softwareplattform Android für einen fahrbaren 6-DoF-Assistenzroboter.

Stanislaw Rasowski

Title of the paper

Tablet-based control unit for a mobile 6DoF assistance robot

Keywords

robot control, robotic arm, kinematics, ROS, Android, Assistant Robot

Abstract

This thesis describes the development of the control unit on the basis of software platform Android for a mobile 6-DoF Assistant Robot.

Inhaltsverzeichnis

Abbildungsverzeichnis v

1 Einleitung 1

2 Konzeption und Systemdesign 2

2.1 Assistenzroboter und Kinematik . . . 2
2.1.1 Grundlagen . . . 2
2.1.2 Inverse Kinematik . . . 3
2.2 Robot Operating System (ROS) . . . 4
2.2.1 Systemkonzept . . . 4
2.2.2 Computation Graph . . . 7
Nodes . . . 7
Nachrichten . . . 9
Topics . . . 10
Services . . . 11
Master . . . 13
Parameter Server . . . 15
2.3 ROS Schnittstelle für Android . . . 16
2.4 Matlab-Simulation des Roboterarms . . . 18
2.4.1 Robotics Toolbox für Matlab . . . 18
2.4.2 UDP Kommunikation zwischen Matlab und Android . . . 20

3 Implementierung 21

3.1 UDP Schnittstelle . . . 21
3.2 Direkte Parametersteuerung . . . 23
3.3 Virtueller Joystick . . . 26
3.4 Multitouch Steuerelement und Gestenerkennung . . . 30
3.5 Antriebssteuerung mit Beschleunigungssensor . . . 37

4 Zusammenfassung 40

Literaturverzeichnis 41

Abbildungsverzeichnis

2.1 Fahrbarer 6-DoF-Assistenzroboter 2

2.2 Die kinematische Kette 3

2.3 Typisches ROS Netzwerk [Morgan Quigley, 2009] 6

2.4 Ein Screenshot von rxgraph Tool, das das Computation Graph visualisiert. [roswikirxgraph, 2013] 8

2.5 Kommunikationsprinzip von Topics und Services. [roswikiconcepts, 2013] 12

2.6 Knoten „Camera“ veröffentlicht das Topic „images“[roswikimaster, 2012] 14

2.7 Knoten „Image_viewer“ abonniert das Topic „images“ [roswikimaster, 2012] 14

2.8 Die Knoten „Camera“ und „Image viewer“ kommunizieren miteinander mittels des Topics „images“ [roswikimaster, 2012] 15

2.9 Visualisierung des Roboterarms mit Robotics Toolbox 20

3.1 Klassendiagramm für UDPSender 22

3.2 Startbildschirm der Anwendung 23

3.3 Direkte Parametersteuerung 25

3.4 Joystick UI 26

3.5 Klassendiagramm für die Joystick-Anwendung 28

3.6 Joystick-Anwendung 29

3.7 Startbildschirm der GesturePad-Anwendung 30

3.8 Klassendiagramm für GesturePadViewUDP Klasse 33

3.9 Die Bewegung des Roboterarmes in einer radialen Ebene 34

3.10 Die Änderung des *Höhe*-Parameters 35

3.11 Die Änderung des *Neigung*-Parameters 36

3.12 Die Achsen des Beschleunigungssensors [andrdsens, 2015] 38

3.13 Antriebssteuerung mit Beschleunigungssensor 39

1 Einleitung

Die mobilen Geräte wie Handy oder Tablet sind heute weit verbreitet. Das Aufgabenspektrum von solchen Geräten ist sehr breit. Am häufigsten sind sie als Haushaltsgeräte für die Kommunikation und die Unterhaltung benutzt. Die Leistung der Tablets oft übertrifft die Leistung von Desktop-Computern. Die moderne Handys und Tablets haben folgende Stärken:

- Die hohe Prozessorleistung
- Die fortgeschritten Kommunikationstechnologien (Wi-Fi, 4G LTE Netzwerke, NFC)
- Der Touchscreen, der gleichzeitig mehrere Berührungen erkennt
- Das Preis-Leistungs-Verhältnis

Alle oben genannten Eigenschaften und die hohe Verfügbarkeit machen die mobilen Geräte sehr attraktiv unter den Entwicklern von unterschiedlichen Robotersystemen.

Diese Arbeit ist ein Versuch, ein Tablet in der untypischen Rolle zu verwenden. Auf Basis von Android-Gerät wird die Steuereinheit für einen fahrbaren 6-DoF-Assistenzroboter entwickelt und alle Stärken des Tablets werden berücksichtigt.

2 Konzeption und Systemdesign

2.1 Assistenzroboter und Kinematik

2.1.1 Grundlagen

Es gibt die unendliche Vielzahl von Roboter und deren Konstruktionen. Um eine Steuereinheit zu konzipieren, muss das Robotersystem gewissermaßen formalisiert werden. Ein abstrakter fahrbarer Roboter besteht zumindest aus zwei Hauptteilen: der fahrbaren Plattform und dem Aktuator oder Roboterarm (Abbildung 2.1).

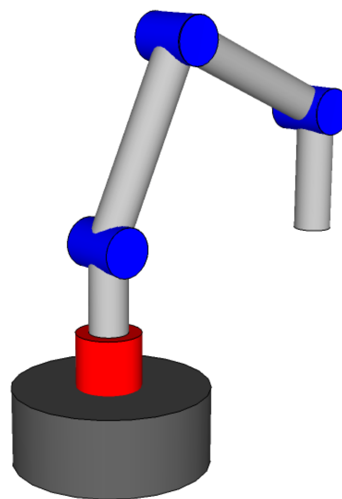


Abbildung 2.1: Fahrbarer 6-DoF-Assistenzroboter

Eine wichtige Charakteristik des Roboters ist die Anzahl von Freiheitsgrade. Der *Freiheitsgrad* ist die unabhängige Bewegungsmöglichkeit des Systems. Der in dieser

Arbeit beschriebene Roboter hat sechs Freiheitsgrade (engl. „DOF - degrees of freedom“). Der Roboterarm hat vier Freiheitsgrade: ein Rotationsgelenk (rot markiert) und drei Knickgelenken (blau markiert) (Abbildung 2.1). Die fahrbare Plattform hat zwei Freiheitsgrade: Linear- und Rotationsbewegung.

Die Steuerung des Arms erfolgt durch die Änderung der Gelenkwinkel. Jedoch kennt der Benutzer nur die Position des Endeffektors. Die Gelenkwinkel sind unbekannt und müssen berechnet werden. Das ist die Aufgabe der inversen Kinematik.

2.1.2 Inverse Kinematik

Die Position des Endeffektors wird in Zylinderkoordinaten beschrieben. Der Radius r zeigt wie weit ist der Endeffektor vom Zentrum der Koordinaten entfernt. Die Höhe h zeigt der Abstand eines Endpunktes des Arms von der Nullebene des Zylinderkoordinatensystems (Abbildung 2.2). Kennt man die alle Gelenkwinkel, so kann man diese Parameter berechnet. Und umgekehrt, falls r und h bekannt sind, können die alle Gelenkwinkel des Roboterarms berechnet werden.

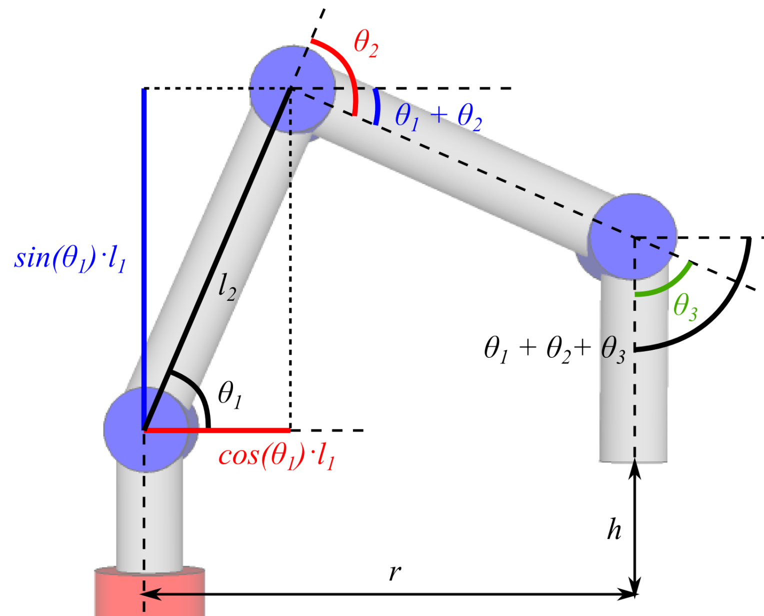


Abbildung 2.2: Die kinematische Kette

Das folgende Gleichungssystem zeigt die Abhängigkeit der Zylinderkoordinaten von Gelenkwinkel.

$$\begin{aligned} h &= l_1 + l_2 \sin(\theta_1) + l_3 \sin(\theta_1 + \theta_2) + l_4 \sin(\theta_1 + \theta_2 + \theta_3) \\ r &= l_2 \cos(\theta_1) + l_3 \cos(\theta_1 + \theta_2) + l_4 \cos(\theta_1 + \theta_2 + \theta_3) \\ \phi &= \theta_1 + \theta_2 + \theta_3 \end{aligned}$$

Um das Problem der inversen Kinematik zu lösen, muss die Nullstelle für das folgende nichtlineare Gleichungssystem berechnet werden.

$$\begin{aligned} 0 &= l_1 + l_2 \sin(\theta_1) + l_3 \sin(\theta_1 + \theta_2) + l_4 \sin(\theta_1 + \theta_2 + \theta_3) - h \\ 0 &= l_2 \cos(\theta_1) + l_3 \cos(\theta_1 + \theta_2) + l_4 \cos(\theta_1 + \theta_2 + \theta_3) - r \\ 0 &= \theta_1 + \theta_2 + \theta_3 - \phi \end{aligned}$$

2.2 Robot Operating System (ROS)

2.2.1 Systemkonzept

Robot Operating System (ROS) ist ein Open-Source Meta-Betriebsystem für Roboter entwickelt von Forschungslabor Willow Garage (https://www.willowgarage.com/) und der Universität von Südkalifornien (http://www.usc.edu/). Heutzutage ist ROS ein Standard für Robotersoftware. Das System ist in vielen Entwicklungen und Robotikprojekten verwendet und hat sehr breiter Entwickler- und Anwenderkreis. Das ROS ist nicht ein einzelnes Roboter-Betriebssystem auf dem Markt. Vor der Einführung dieses Systems es gab viele in gewissem Maße ähnliche Systeme. Die bekannteste sind:

- CARMEN (http://carmen.sourceforge.net/)
- Orcos (http://www.orocos.org/)

- YARP (http://eris.liralab.it/yarp/)
- Microsoft Robotics Studio (http://msdn.microsoft.com/en-us/robotics/default.aspx)
- Player (http://playerstage.sf.net/)

ROS ist kein Betriebssystem im gewöhnlichen Sinne. Es ist ein Framework (deu. „Rahmenwerk“), das besteht aus den zahlreichen Bibliotheken, Werkzeugen, Gerätetreiber und Hilfsprogrammen. ROS läuft auf Unix-basierten Plattformen wie Ubuntu, Mac OS oder Android. [roswikiintro, 2013]

Die wichtigste philosophische Ziele des ROS sind:

- Peer-to-Peer
- Mikrokernel (Werkzeugbasiert)
- Mehrsprachigkeit
- Schlank
- Kostenlos und Open-Source

Peer-to-Peer: Das System besteht aus vielen gleichzeitig laufenden Prozessen (Knoten), die miteinander zusammen verbunden sind. Die leistungshungrige Prozesse, wie zum Beispiel Bildanalyse oder Trajektorienberechnung, laufen auf den Fernhosts. Die Treiber für Geräte, wie Sensoren oder Manipulatoren, laufen direkt auf dem Roboter (Abbildung 2.3). Aber die Peer-to-peer Topologie ermöglicht es , alle Systeme miteinander zu verknüpfen und jeder Knoten hat Zugriff auf alle anderen. Die Peer-to-Peer-Topologie erfordert eine Art von Lookup-Mechanismus (Nameservice oder Master), um die Verbindung zwischen allen Prozessen zu erlauben. [Morgan Quigley, 2009]

Mikrokernel: Obwohl ROS ein sehr komplexes System ist, ist es nicht monolithisch gebaut. Es besteht aus mehreren kleinen Tools (Mikrokernels), die die Arbeit des ganzen Systems gewährleisten. Das Mikrokernel-Design macht das System stabil trotz der unerheblichen Produktivitätsverlust. [Morgan Quigley, 2009]

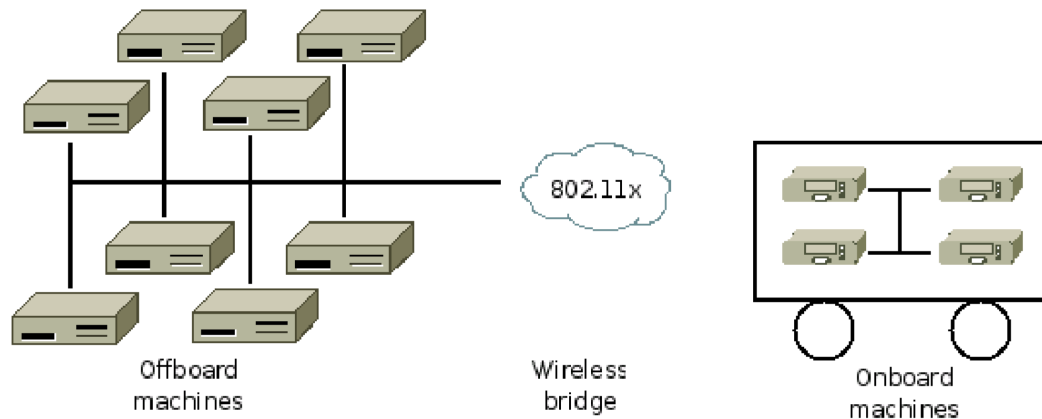


Abbildung 2.3: Typisches ROS Netzwerk [Morgan Quigley, 2009]

Mehrsprachigkeit: ROS ist programmiersprachenunabhängig und unterstützt derzeit vier sehr verschiedenen Programmierprachen: C++, Phyton, Octave und LISP. Um die Quersprachentwicklung zu unterstützen, benutzt ROS die formale sprachneutrale Schnittstellenbeschreibungssprache - IDL (Interface Denition Language). Die IDL beschreibt nur die Schnittstelle, ist aber für die Formulierung von Algorithmen ungeeignet. [Morgan Quigley, 2009]

Schlank: Viele Robotikprojekte beinhalten die Algorithmen und die Treiber, die in anderen Projekten wiederverwendet werden können. Leider gibt es eine Vielzahl von komplexen Gründen, die es unmöglich machen den Code zu „extrahieren“. Um dieses Problem möglichst zu vermieden ist ROS so konzipiert, dass alle komplexe Algorithmen, Funktionen und Treiber in großen alleinstehenden Bibliotheken realisiert sind. Diese Bibliotheken sind komplett unabhängig von ROS entwickelt und können auch in anderen Systemen benutzt werden. Anderseits bietet ROS kleine ausführbare Dateien, die die Funktionalität dieser Bibliotheken verwenden.

Dieses Systemdesign macht ROS schlank (engl. „thin“) und lässt den Code von vielen Open-Source Projekten wiederzuverwenden. Eine der am häufigsten verwendeten sind die folgenden Projekte:

- OpenCV - Bildverarbeitung und Vision Algorithmen (http://opencv.org/)

- OpenRAVE - Navigation und Bewegungsplanung (http://openrave.org/)
- Player project - Simulation und Schnittstellen für verschiedene Roboter und Sensoren (http://playerstage.sourceforge.net/)

[Morgan Quigley, 2009]

Kostenlos und Open-Source: ROS ist komplett kostenlos und der Quellcode für die aktuelle Version des Systems und früheren Versionen ist frei verfügbar (https://github.com/ros). ROS ist unter BSD-Lizenz verbreitet. Dies macht es möglich, sowohl kommerzielle als auch nicht kommerzielle Projekte zu entwickeln. [Morgan Quigley, 2009]

2.2.2 Computation Graph

ROS Computation Graph stellt der innere Aufbau des Systems dar. Die Hauptelemente des Graphs sind die *Knoten (Nodes)*, die *Topics*, die *Nachrichten* und die *Services*. Computation Graph zeigt die Verbindungen zwischen den Knoten und ist im Prinzip ein Netzwerk mit Peer-to-peer Topologie (Abbildung 2.4). [roswikiconcepts, 2013]

Nodes

Ein Knoten (engl. „Node“) ist ein Prozess, das die Berechnung durchführt oder ein Gerät mittels des Treibers darstellt. Normalerweise besteht ein typisches Robotersystem aus mehreren gleichzeitig laufenden Knoten. Zum Beispiel, ein Knoten steuert die Fahrmotoren, ein anderen steuert der Abstandssensor, der Dritte berechnet der Bewegungspfad und so weiter. Solcher Aufbau bietet wesentliche Vorteile dem ganzen System.

Der wichtigste Vorteil ist die *Fehlertoleranz*. Fällt ein Knoten heraus, bleiben die anderen Knoten und das ganze System laufend.

Der zweite Vorteil ist die *Leichtgewichtigkeit*. Die ganze Komplexität ist in den Knoten realisiert, die System bleibt leichtgewichtig und flexibel. Die Logik und die Algorithmen

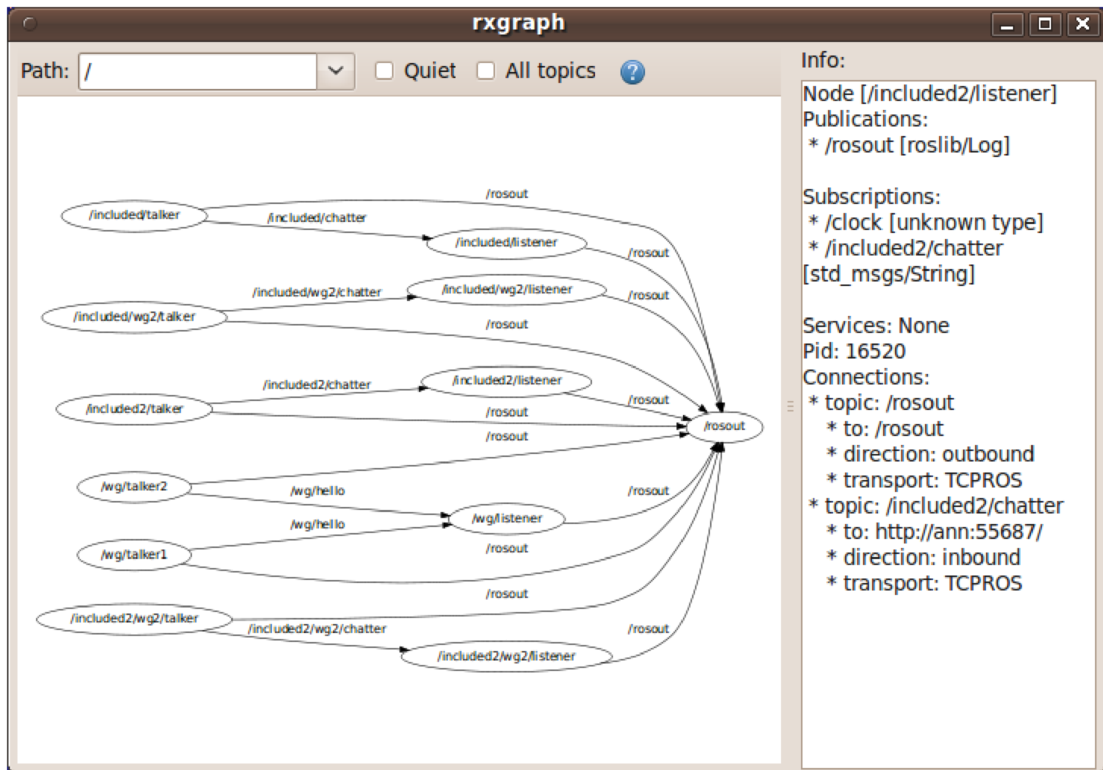


Abbildung 2.4: Ein Screenshot von rxgraph Tool, das das Computation Graph visualisiert. [roswikirx-graph, 2013]

sind für die anderen Knoten nicht sichtbar und die Kommunikation erfolgt durch das minimale API[1].

Alle Knoten haben seine eigenen einzigartigen Namen, um eindeutig identifiziert zu werden. Außerdem, haben alle Knoten ein Typ, der beschreibt den Pfad zur ausführbaren Datei des Knotens.

Um die Knoten handzuhaben und zu steuern, hat ROS ein Kommandozeilen-Tool wie `rosnode`. Die unterstützten Befehle sind wie folgt:

- rosnode info node_name: Print information about node
- rosnode kill node_name: Kill a running node

[1] API - (engl. „*A*pplication *P*rogramming *I*nterface“) Anwendungsprogrammierschnittstelle

- rosnode list: List active nodes
- rosnode machine hostname: List nodes running on a particular machine or list machines
- rosnode ping node_name: Test connectivity to node
- rosnode cleanup: Purge registration information of unreachable nodes

[roswikinodes, 2012] [Joseph, 2015]

Nachrichten

Die Kommunikation zwischen den Knoten erfolgt über die Nachrichten (engl. „Messages"). Eine Nachricht ist eine einfache Datenstruktur, die die Felder von unterschiedlichen Datentypen enthält. Die unterstützte Datentypen sind Integer, Float, Boolean etc. Die ROS Nachrichten unterstützen die Arrays von akzeptierten Datentypen und die Arrays von Nachrichten.

ROS hat viele vordefinierte Nachrichtentypen und es ist möglich ein eigenen Nachrichtentyp zu definieren. Die Nachrichtendatei mit .msg Erweiterung soll in `msg/` Ordner des Packages gespeichert werden.

Eine typische ROS Nachricht besteht aus zwei Teile: die Felder und die Konstanten. Die Felder definieren den Datentyp zum Übertragen, die Konstanten definieren die Namen der Felder.

Ein Beispiel für die Nachrichtentypdatei:

```
int32     id
float32   vel
string    name
```

Ein Sondertyp in ROS ist Header. Der fügt der Zeitstempel, das Frame etc. in die Nachricht hinzu. Der Header macht es möglich die Nachrichten zu nummerieren, sodass der Absender bestimmt werden kann.

Ein Beispiel für die Headerdatei:

```
1 uint32   seq
2 time     stamp
3 string   frame_id
```

Um die Nachrichentypen zu steuern, hat ROS ein Kommandozeilen-Tool `rosmsg`.Die unterstützten Befehle sind wie folgt:

- rosmsg show: Show message description
- rosmsg list: List all messages
- rosmsg md5: Display message md5sum
- rosmsg package: List messages in a package
- rosmsg packages: Lists packages that contain message

[roswikimsg, 2015] [Aaron Martinez, 2013]

Topics

Die Knoten verwenden die Topics (deu. "Themen") als die Busse zur asynchronen Datenübertragung. Die Themen haben anonyme Publish/Subscribe-Mechanismus. Das bedeutet, dass jeder Knote ein Thema anonym abonnieren oder veröffentlichen kann. Alle Themen sind über einen einzigartigen Namen identifiziert. Ein Knoten kann gleichzeitig mehreren Themen abonnieren und veröffentlichen. Die gesendete Nachricht bekommen alle Knoten, die bestimmte Thema abonnieren (one-to-many Kommunikation). Der Absender und der Empfänger wissen nicht voneinander, so sind Datenkonsum und Datenerzeugung entkoppelt.

Jedes Thema ist stark nach Nachrichtentyp typisiert. Das bedeutet, dass die abonnierenden Knoten nur die Nachrichten von einem bestimmten Typ bekommen können.

ROS unterstützt TCP/IP-basierte und UDP-basierte Datenübertragungsmechanismen. Der TCP/IP-basierte Datenübertragungsmechanismus ist als voreingestellt verwendet, und heißt TCPROS[2]. Der UDP-basierte Datenübertragungsmechanismus heißt UDPROS. Der bietet ein verlustbehafteten Datentransport mit geringer Wartezeit und ist für die Aufgaben wie Teleoperation gut geeignet.

Das Kommandozeilen-Tool für die Themensteuerung heißt `rostopic`. Die folgenden Befehle sind unterstützt:

- rostopic bw: Display bandwidth used by topic
- rostopic echo: Print messages to screen
- rostopic find: Find topics by type
- rostopic hz: Display publishing rate of topic
- rostopic info: Print information about active topic
- rostopic list: Print information about active topics
- rostopic pub: Publish data to topic
- rostopic type: Print topic type

[roswikitopic, 2014] [Aaron Martinez, 2013]

Services

Im Gegensatz zu Themen, die many-to-many Kommunikationsprinzip realisieren, sind Services für die Anfrage-Antwort Interaktion zwischen den Knoten verwendet (Abbildung 2.5).

[2] TCPROS - die Transportschicht für ROS Nachrichten und Services. Die benutzt TCP/IP Sockets für die Datenübertragung.

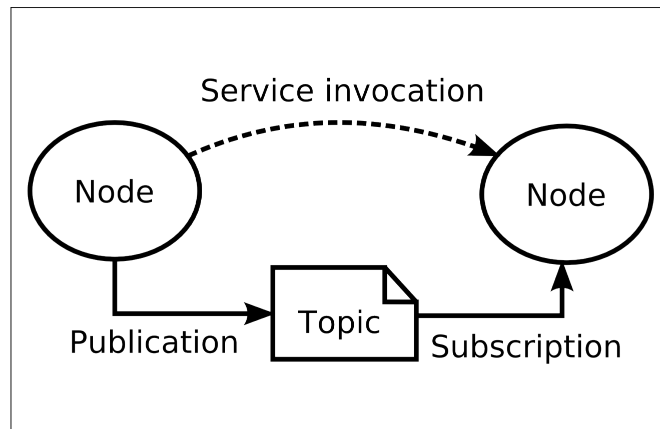


Abbildung 2.5: Kommunikationsprinzip von Topics und Services. [roswikiconcepts, 2013]

Tatsächlich bieten die Services (deu. „Dienste") die RPC[3]-Funktionalität, ähnlich wie bei Programmiersprachen. Einer Knoten stellt das Service unter einen Namen zur Verfügung, der Client-Knoten ruft den Service durch Senden die Anfragenachricht und wartet auf Antwort.

Um die Services zu verwalten, bietet ROS zwei Kommandozeilen-Tools: `rossrv` und `rosservice`.

Das Tool `rossrv` arbeitet mit .srv-Dateien genau wie `rosmsg` mit .msg-Dateien. Das zeigt die Beschreibung von Services, die Pakete, die .srv-Dateien beinhalten, und sucht die Quellcodedateien, die bestimmte Services benutzen. `rossrv` unterstützt folgende Befehle:

- rossrv show: Show message description
- rossrv list: List all messages
- rossrv md5: Display message md5sum
- rossrv package: List messages in a package
- rossrv packages: List packages that contain messages

[3]RPC - *R*emote *P*rocedure *C*all (deu. „Aufruf einer fernen Prozedur") ist eine Interprozesskommunikation, die den Aufruf von Funktionen oder Prozeduren in anderen Adressräumen ermöglicht

Um die Services zu suchen und auszuführen, benutzt man das Tool `rosservice`. Das unterstützt folgende Befehle:

- rosservice call: Call the service with the provided args
- rosservice find: Find services by service type
- rosservice info: Print information about service
- rosservice list: List active services
- rosservice type: Print service type
- rosservice uri: Print service ROSRPC uri

[roswikiservice, 2012] [Aaron Martinez, 2013]

Master

Der ROS-Master ist ein Kern des Systems, der die Namensgebung und die Registrierungsdienstleistungen für die übrigen Knoten bietet. In anderen Worten: der Master macht die anderen Knoten untereinander bekannt.

Zum Beispiel ein System beinhaltet zwei Knoten. Der Knoten „Camera“ erzeugt die Grafikdaten (Bilder), der Knoten „Image_viewer“ ist zuständig die Grafikdaten abzubilden. Um die Bilder zu veröffentlichen, meldet der Knoten „Camera“ das Topic „images“ beim Master an (Abbildung 2.6).

Das Topic „images“ ist jetzt erfolgreich angemeldet, aber es erscheint keine Datenübertragung, da es noch kein Knoten das Topic abonniert. Der Knoten „Image_viewer“ abonniert das Topic „images“ mittels Anfrage an Masterknoten (Abbildung 2.7).

Nun hat das Topic „images“ sowohl den Herausgeber als auch den Abonnent. Die Knoten „Camera“ und „Image_viewer“ können jetzt frei miteinander mittels des Topics „images“ kommunizieren (Abbildung 2.8).

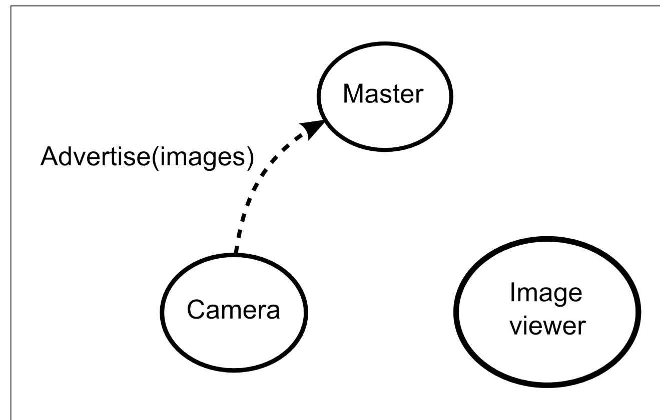


Abbildung 2.6: Knoten „Camera“ veröffentlicht das Topic „images“[roswikimaster, 2012]

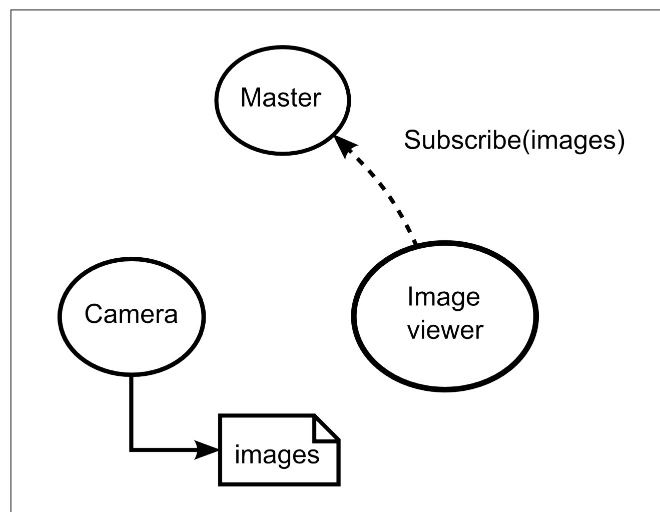


Abbildung 2.7: Knoten „Image_viewer“ abonniert das Topic „images“ [roswikimaster, 2012]

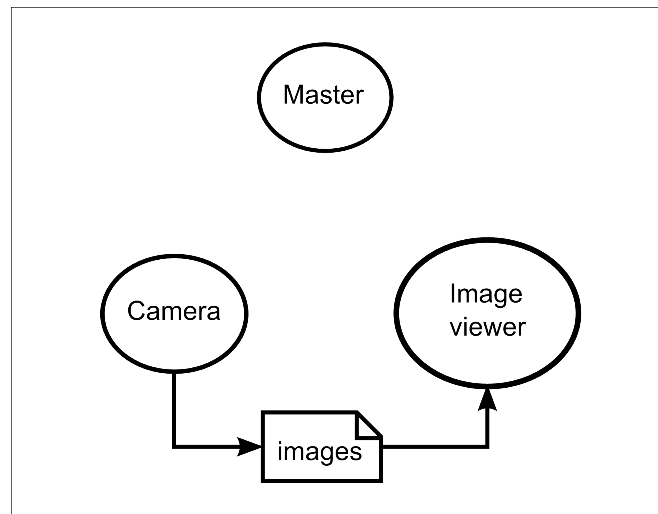


Abbildung 2.8: Die Knoten „Camera“ und „Image viewer“ kommunizieren miteinander mittels des Topics „images“ [roswikimaster, 2012]

Zusätzlich zu dem Registrierungsdienst bietet der ROS-Master auch der Parameter Server. [roswikimaster, 2012]

Parameter Server

Ein Parameter Server ist ein gemeinsames multivariates Wörterbuch, das über Netzwerk-APIs zu erreichen ist. Die Knoten benutzen den Server, um die Parameter in Laufzeit zu speichern und abzufragen. Da ein Parameter Server ist nicht hochleistungsfähig, ist er für die Abspeicherung von statischen Daten, wie zum Beispiel die Konfigurationsparameter, gut geeignet.

Der Parameter Server ist mit XML-RPC[4] implementiert und läuft innerhalb des ROS-Masters.

Der Parameter Server unterstützt folgende XML-RPC Datentypen:

- 32-bit Integers
- Booleans

[4]XML-RPC - die Spezifikation, die die Prozeduraufrufe über das Internet ermöglichst [xmlrpc, 1999]

- Strings
- Doubles
- ISO 8601 dates
- Lists
- Base 64-encoded binary data

Um den Parameter Server zu steuern, benutzt man das Tool `rosparam`. Die folgenden Befehle sind unterstützt:

- rosparam set: Set parameter
- rosparam get: Get parameter
- rosparam load: Load parameters from file
- rosparam dump: Dump parameters to file
- rosparam delete: Delete parameter
- rosparam list: List parameter names

[roswikiparam, 2013]

2.3 ROS Schnittstelle für Android

rosjava ist eine erste reine Java-Implementierung von ROS. Sie macht es möglich das ROS auf einem Java-fähigen Gerät, wie Handy oder Tablet, zu starten. Das Projekt ist von Google in Kooperation mit Willow Garage entwickelt und wurde im Jahr 2011 auf Google I/O 2011 präsentiert. Das Ziel der Implementierung ist die Entwicklung von fortgeschrittenen Robotik-Anwendungen, besonders für die Software-Plattform Android, zu ermöglichen.

Die Software ist heutzutage im Alpha-Release-Modus und ist immer noch in aktiver Entwicklung, aber die meisten Funktionen sind größtenteils fertig und funktionsfähig. rosjava bietet eine Client-Bibliothek, die Java-Programmierer ermöglicht, schnell und unkompliziert mit ROS Topics, Services und Parameter Server zu interagieren. Es bietet auch eine Java-Implementierung von roscore, das heißt, es ist möglich das Java-Gerät nicht nur als Teil (Knoten) des Systems (ROS) zu benutzen, sondern ein komplett unabhängiges System auf Basis von Gerät aufzubauen. [rosjavagit, 2015] [googleio11, 2011]

Um ein ROS-Knoten in Java zu erstellen, muss man ein Interface org.ros.node.NodeMain implementieren. Ein typisches Template für ein Knoten in Java sieht so aus:

```
import org.ros.node.NodeMain;
import org.ros.namespace.GraphName;
import org.ros.node.Node;

public class NewNode implements NodeMain {

    @Override
    public GraphName getDefaultNodeName() {
        return new GraphName("new_node");
    }

    @Override
    public void onStart(ConnectedNode node) {
    }

    @Override
    public void onShutdown(Node node) {
    }

    @Override
    public void onShutdownComplete(Node node) {
    }

    @Override
    public void onError(Node node, Throwable throwable) {
    }}
```

Der oben gezeigte Beispielcode stellt ein Lebenszyklus des Knotens dar. Die Methode getDefaultNodeName liefert den Namen des Knotens. Dieser Name repräsentiert der Knoten in Komputation Graph. Die onStart-Methode ist der Einstiegspunkt für das Programm. An dieser Stelle werden Publisher und Subscriber Instanzen initialisiert und das größte Teil der Knotenlogik realisiert. Die onShutdown-Methode wird erst nach Start der Shutdown-Prozess aufgerufen. Das ist der erste Austrittspunkt des Knotens. Die Herunterfahrprozesse von allen Publishers, Subscriber etc. werden in diesem Zeitpunkt pausiert und warten auf die Rückgabe von onShutdown-Methode des Knotens. Der letzte Austrittspunkt des Programms ist die Methode onShutdown-Complete. Diese Methode wird aufgerufen, sobald die Herunterfahrprozesse von allen Publishers und Subscribers abgeschlossen sind. [rosjava, 2013]

2.4 Matlab-Simulation des Roboterarms

2.4.1 Robotics Toolbox für Matlab

Das Robotics Toolbox ermöglicht die Modellierung und die Visualisierung des Roboters in Matlab. Das ist sehr flexibles und vielseitiges Werkzeug.

Um ein einfaches Robotersystem zu modellieren, müssen zuerst die Gelenke des Roboterarms wie folgt definiert werden.

```
1 l1 = 208;
2 l2 = 400;
3 l3 = 300;
4 l4 = 368;
5
6 L1=link([+pi/2  0  0  l1  ]);
7 L2=link([ 0    l2  0   0  ]);
8 L3=link([ 0    l3  0   0  ]);
9 L4=link([ 0    l4  0   0  ]);
```

Der Roboter wird mit dem Befehl *robot* erstellt.

```
schunk = robot({L1,L2,L3,L4}, 'Schunk Arm');
```

Die Startpose wird wie folgt definiert:

```
alpha =0;  r = 200;  h = 150;  phi_neig=-90;
```

Um die inverse Kinematik zu realisieren, muss das nichtlineare Gleichungssystem definiert werden.

```
function F = nonlin_gleichung(x, r, h, p)
F = [ 190*cos(x(1)) + 138*cos(x(1)+x(2)) + 206.1*cos(x(1)+x(2)+x(3)) - r;
      190*sin(x(1)) + 138*sin(x(1)+x(2)) + 206.1*sin(x(1)+x(2)+x(3)) + 171 - h;
      x(1) + x(2) + x(3) - p*pi/180 ];
```

Die Lösung des Gleichungssystems erfolgt mittels des Befehls *fsolve*.

```
x = fsolve(fkt, x0);
```

Sobald das Gleichungssystem gelöst ist, werden die berechnete Gelenkwinkel gespeichert und es wird ein Gelenkwinkelvektor erzeugt.

```
q2 = x(1);
q3 = x(2);
q4 = x(3);

q = [alpha q2 q3 q4 ];
```

Der Befehl *plot* zeigt den Roboter an (Abbildung 2.9).

```
plot(schunk, q);
```

[Corke, 1996]

2.4.2 UDP Kommunikation zwischen Matlab und Android

Um die Daten vom Android-Gerät zu bekommen, wird das Matlab-Objekt *udp* verwendet.

```
udpAlpha  = udp('192.168.2.100', 4012, 'LocalPort', 5050);
udpRadius = udp('192.168.2.100', 4012, 'LocalPort', 5051);
udpHeight = udp('192.168.2.100', 4012, 'LocalPort', 5052);
udpPhi    = udp('192.168.2.100', 4012, 'LocalPort', 5053);
```

Die Daten werden in einer endlosen Schleife abgelesen und die entsprechende Variable werden aktualisiert.

Das Ablesen des Radius:

```
if get(udpRadius, 'BytesAvailable') > 0
   dataRadius = fscanf(udpRadius, '%f');
end;
```

[matlabudp, 2015]

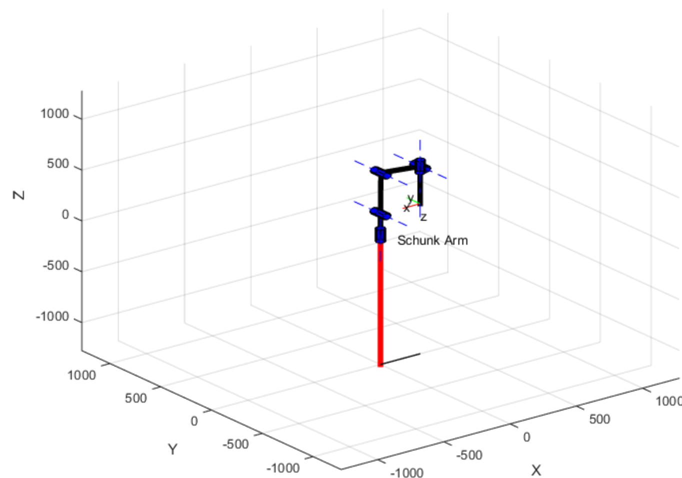


Abbildung 2.9: Visualisierung des Roboterarms mit Robotics Toolbox

3 Implementierung

Diese Kapitel beschreibt die Implementierung von Steuerkomponenten für Softwareplattform Android. Die folgenden visuellen Komponenten sind für die Steuerung des Roboterarms konzipiert:

- Direkte Parametersteuerung
- Virtueller Joystick
- Multitouch Steuerelement

Alle obengenannte Komponenten verwenden die Klasse UDPSender, die als die UDP-Schnittstelle zwischen Matlab und Android dient.

Die Implementierung von Antriebsteuerung basiert auf rosjava und android_core Bibliotheken und kommuniziert mit dem Robotersimulator Turtlesim. Dieser Simulator lauft in der ROS-Umgebung und stellt das entfernte Robotersystem dar.

3.1 UDP Schnittstelle

Um die Verbindung zwischen Matlab und Android zu erstellen, wird eine Klasse UDPSender entwickelt. Die Funktionalität der Klasse basiert auf ArrayBlockingQueue<String> Warteschlange. Die Abbildung 3.1 zeigt das Klassendiagramm, das die Klasse UDPSender beschreibt.

Die Hauptmethode der UDPSender ist die Methode *start()*. Diese Methode wird vom Konstruktor aufgerufen und startet ein Thread. Der Thread beginnt eine Endlosschlei-

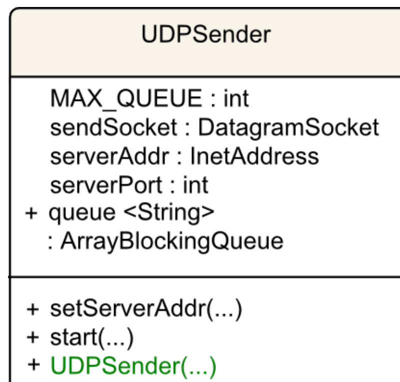


Abbildung 3.1: Klassendiagramm für UDPSender

fe. Die Methode *take().getBytes()* holt den Kopf der Warteschlange raus. Falls die Warteschlange leer ist, wartet die Methode, wenn nötig, bis ein Element verfügbar ist. [blkqueue, 2014]

```
1 while (true) {
2     sendData = queue.take (). getBytes ();
3     DatagramSocket socket = new DatagramSocket();
4     DatagramPacket packet =
5         new DatagramPacket(sendData, sendData.length, serverAddr,  serverPort );
6     socket.send(packet );
7     socket. close ();
8 }
```

Nach der erfolgreichen Ermittlung des Elementes wird neues Datagramm-Socket und neues Datagram Packet erstellt. Das Paket wird mit dem Befehl *socket.send(packet)* geschickt.

Um die Daten zu schicken, muss die Methode *queue.offer(<String>)* von außen aufgerufen werden (die Warteschlange *queue* ist als öffentliche Variable deklariert).

Zum Beispiel:

```
1     UDPSender sender = new UDPSender("localhost", 5050);
2     sender.queue. offer ( "Hello ,  world!" );
```

3.2 Direkte Parametersteuerung

Die Steuerung des Roboterarms erfolgt über die Änderung der folgenden Parameter:

- **Alpha** - Polarwinkel des Roboterarmes
- **Radius** - Abstand vom Mittelpunkt der Radialebene
- **Höhe** - Abstand eines Endpunktes des Manipulators von der Nullebene des Polarkoordinatensystems
- **Neigung** - Gesamtneigungswinkel des Armes

Alle oben genannten Parameter werden mit Hilfe von *android.widget.SeekBar* Komponenten dargestellt. Die Abbildung 3.2 zeigt den Startbildschirm der Anwendung. Die Benutzeroberfläche besteht aus einem Eingabefeld für die IP-Adresse des Simulators, der Set-Taste, die die Adresseingabe bestätigt, und vier SeekBars, die die Steuerungsparameter darstellen.

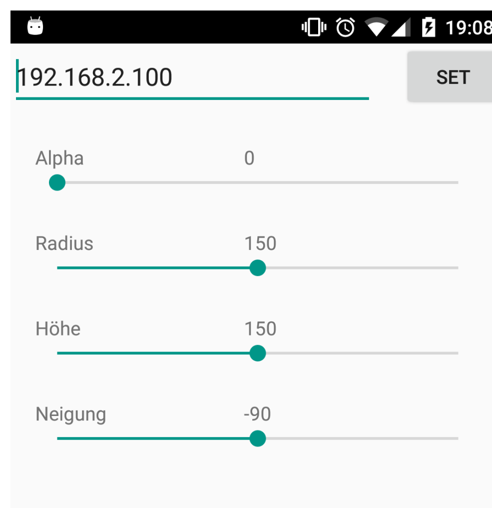


Abbildung 3.2: Startbildschirm der Anwendung

Um den bestimmten Parameter zu ändern, muss der Benutzer den Marker von Seekbar entweder nach links oder nach rechts ziehen. Der Wert wird entsprechend erhöht oder verringert.

Sobald der Benutzer den Wert ändert, erscheint das Ereignis *OnSeekBarChange*. Um dieses Ereignis zu bearbeiten, muss ein Ereignis-Listener[1] entsprechend eingestellt werden. Der folgende Code zeigt die Einstellung des Ereignis-Listeners:

```
seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
        textView.setText(Integer.toString(seekBar.getProgress()));
        sender_alpha.queue.offer(Integer.toString(seekBar.getProgress()));
     }...
});
```

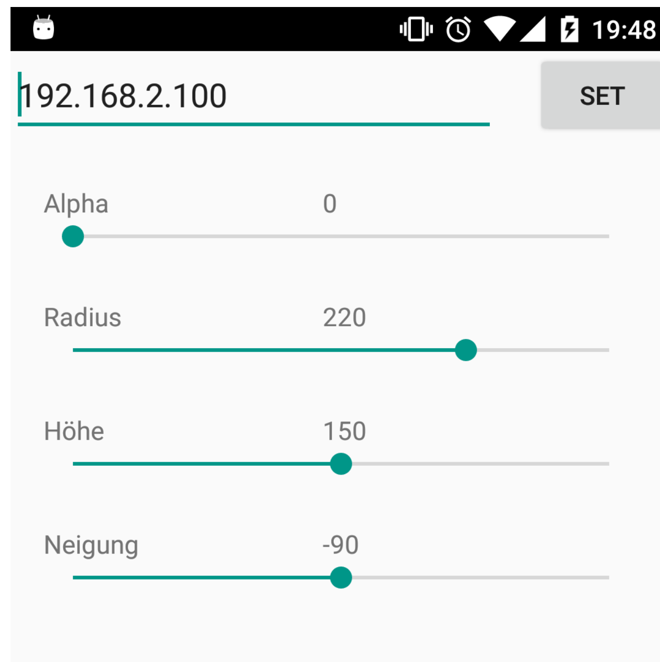
Es wir ein neuer Listener, der das Interface *SeekBar.OnSeekBarChangeListener* implementiert, erstellt. Die Override-Methode *onProgressChanged()* wird aufgerufen, sobald der Wert geändert wird. Die Methode aktualisiert das Textfeld, das aktueller Wert des Parameters zeigt, und schickt den neuen Wert an den Simulator per UDPSender-Objekt.[andrdseekb, 2015]

Jedem Parameter entspricht ein eigenes UDPSender-Objekt mit bestimmter UDP-Portnummer. Die folgende Portnummern sind entsprechend zugeordnet:

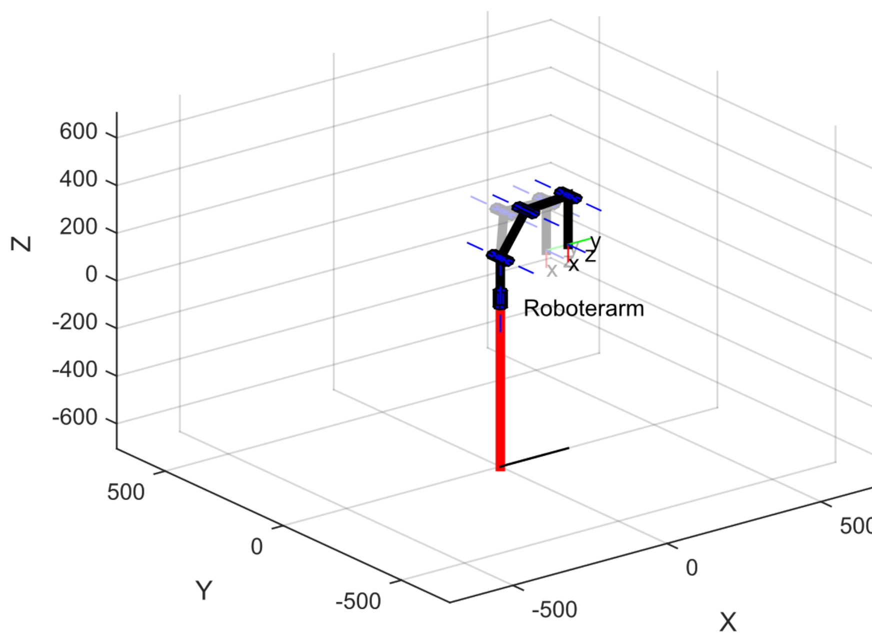
- Alpha - UDP-Portnummer: *5050*
- Radius - UDP-Portnummer: *5051*
- Höhe - UDP-Portnummer: *5052*
- Neigung - UDP-Portnummer: *5053*

Die Abbildung 3.3(a) zeigt den geänderten Parameter *Radius*, der auf 220 gesetzt wird. An der Abbildung 3.3(b) ist die Änderung des Roboterarms gezeigt. Die Startpose ist halbtransparent dargestellt.

[1] Ereignis-Listener oder Event Listener - deu. „Ereignisbehandlungsroutine“



(a) Parameteränderung: Radius auf 220 gesetzt



(b) Reaktion des Simulators

Abbildung 3.3: Direkte Parametersteuerung

3.3 Virtueller Joystick

Ein virtueller Joystick steuert die Bewegung des Endeffektors in einer radialen Ebene. Die Verschiebung des Zeigers entlang *Y-Achse* ändert den Polarwinkel des Roboterarmes (*Alpha* Parameter), die Verschiebung entlang *X-Achse* - den Abstand vom Mittelpunkt der Radialebene (*Radius* Parameter). Der Joystick ermöglicht die gleichzeitige Änderung der beiden Parameter. Die Ausrichtung des Armes (die Höhe und der Neigungswinkel) bleibt konstant und sind mit Hilfe von entsprechenden SeekBar-Komponenten direkt korrigierbar. Die Abbildung 3.4 zeigt den Startbildschirm der Joystick Anwendung.

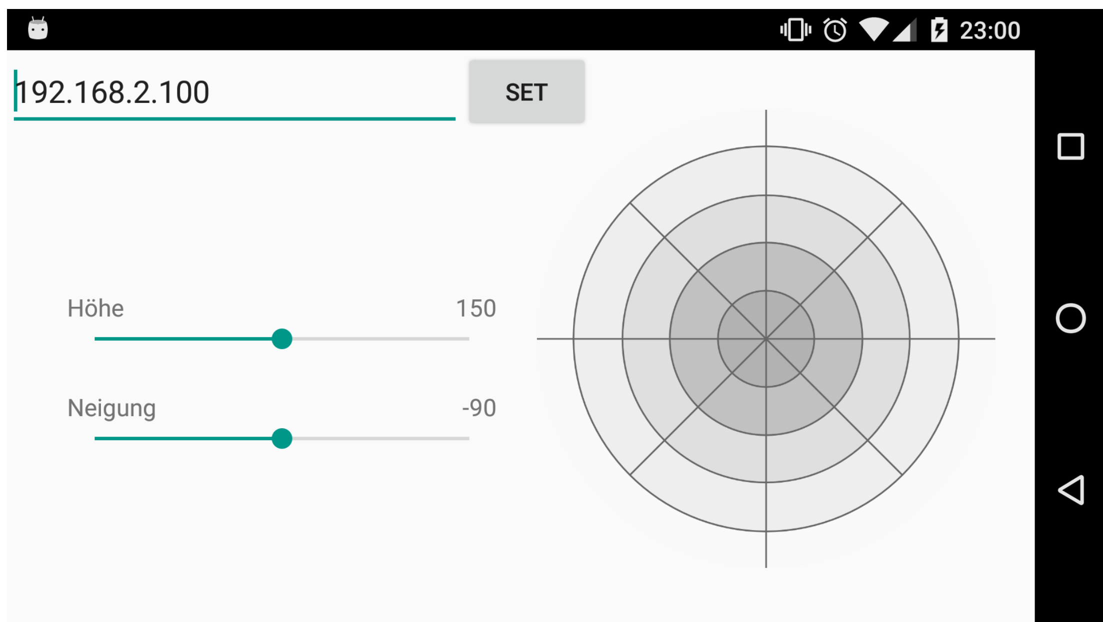


Abbildung 3.4: Joystick UI

Die Joystick-Komponente ist auf Basis von *android.widget.RelativeLayout* gebaut. Die Methode *drawPointer()* ist als Berührungsereignis-Listener eingestellt.

```
1 ltJoystick .setOnTouchListener(new OnTouchListener() {
2   public boolean onTouch(View view, MotionEvent event) {
3     newJoystick.drawPointer(event );
4     return true;
5   }});
```

Nach dem Aufruf konvertiert die Methode *drawPointer()* die Koordinaten des Fingers von absoluten Koordinaten in Joystick-Koordinaten mit dem Zentrum in dem Mittelpunkt des Joysticks. Der Abstand vom Mittelpunkt wird hier auch berechnet.

```
1 public void drawPointer(MotionEvent event) {
2     pos_x = (int) (event.getX() − (ltparams.width / 2));
3     pos_y = (int) (event.getY() − (ltparams.height / 2));
4     dist  = (float) Math.sqrt(Math.pow(pos_x, 2) + Math.pow(pos_y, 2));
```

Die Methode *drawPointer()* erkennt und behandelt die folgende Ereignisse:

- Das Ereignis *MotionEvent.ACTION_DOWN* erscheint, sobald die Berührungsgeste angefangen hat. An dieser Stelle wird geprüft, ob die Fingerkoordinaten sich in der aktiven Zone des Joysticks befinden. Der Joystickspointer wird gezeichnet. Die Flag-Variable *touchState* wird auf „true“ gesetzt.

```
1 case MotionEvent.ACTION_DOWN:
2     if( dist <= (ltparams.width / 2) − POINTER_OFFS) {
3         draw.position(event.getX(), event.getY());
4         draw();
5         touchState = true;}
6     break;
```

- Das Ereignis *MotionEvent.ACTION_MOVE* erscheint, sobald die Fingerkoordinaten sich geändert haben. An dieser Stelle prüft die Methode, ob die neue Koordinaten sich immer noch in dem aktiven Bereich befinden. Falls die Koordinaten gültig sind, werden sie an den Simulator per entsprechendem UDPSender-Objekt geschickt. Der Joystickspointer wird neu gezeichnet.

```
1 case MotionEvent.ACTION_MOVE:
2     if (touchState)
3         if (dist <= (ltparams.width / 2) − POINTER_OFFS) {
4             draw.position(event.getX(), event.getY());
5             draw();
6             sender_radius.queue.offer(Float.toString(pos_y));
7             sender_alpha.queue.offer(Float.toString(pos_x));
8     break;
```

- Das Ereignis *MotionEvent.ACTION_UP* signalisiert über den Abschluss der Berührungsgeste. An dieser Stelle wird der Joystick in die Ausgangsposition zurückgesetzt. Die Flag-Variable *touchState* wird auf „false“ gesetzt. Der Pointer wird entfernt. An den Simulator werden die Nullwerte geschickt.

```
case MotionEvent.ACTION_UP:
    mLayout.removeView(draw);
    touchState = false;
    sender_radius.queue.offer(Float.toString(0));
    sender_alpha.queue.offer(Float.toString(0));
    break;
```

[andrdmoevent, 2015]

Die Abbildung 3.5 zeigt das Klassendiagramm , das die Klasse Joystick im Kontext der Anwendung darstellt. Die Abbildung 3.6(a) zeigt die Anwendung im Einsatz. Hier werden alle vier Parameter geändert. Die Abbildung 3.6(b) zeigt die Reaktion des Simulators.

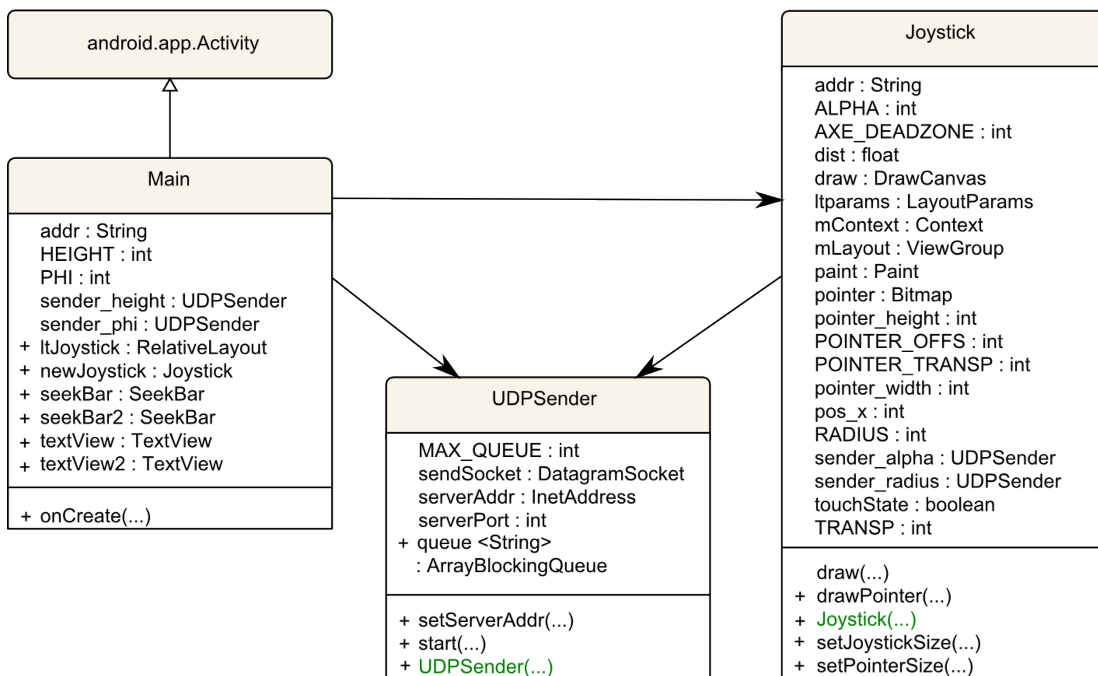
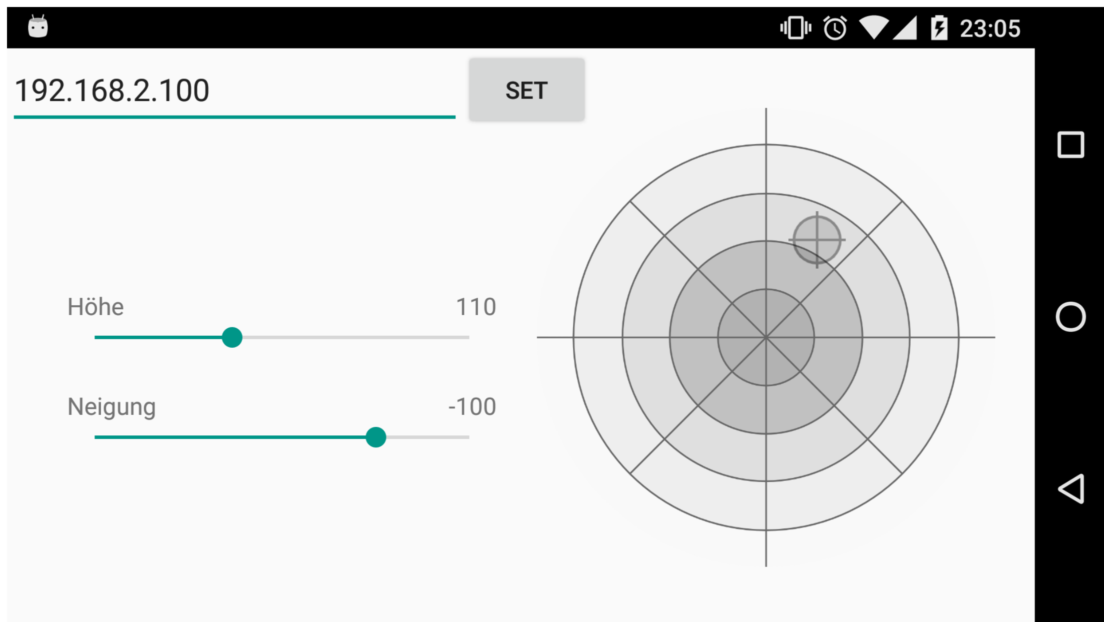
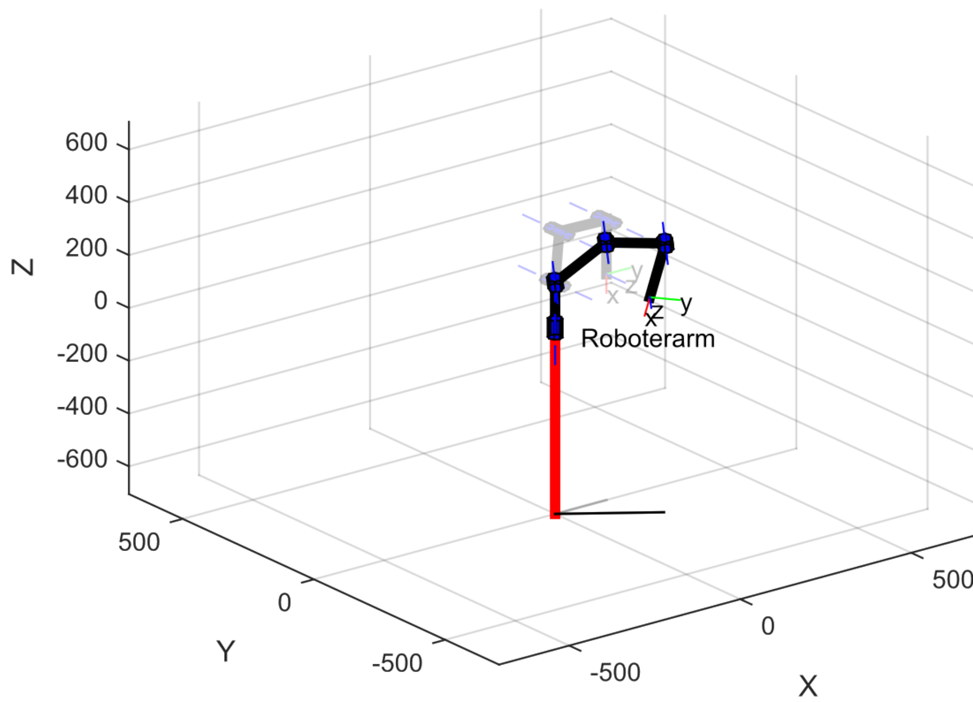


Abbildung 3.5: Klassendiagramm für die Joystick-Anwendung



(a) Joystick-Anwendung im Einsatz



(b) Reaktion des Simulators

Abbildung 3.6: Joystick-Anwendung

3.4 Multitouch Steuerelement und Gestenerkennung

Die *GesturePadView*-Komponente ist die Erweiterung der Konzeption des Joysticks. Diese Komponente erkennt nicht nur die Bewegung des einzigen Fingers, sondern auch die Multitouch-Bewegungen und die Mehrfinger-Gesten.

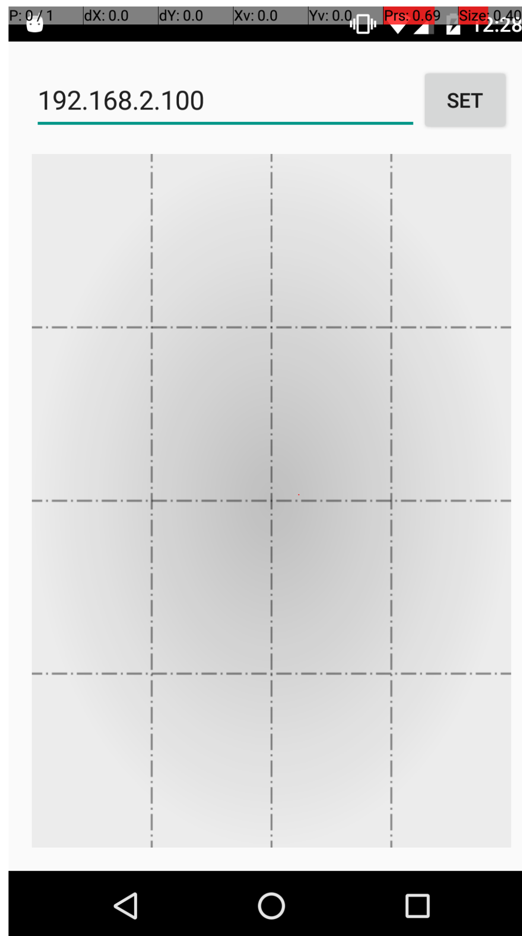


Abbildung 3.7: Startbildschirm der GesturePad-Anwendung

Die Abbildung 3.7 zeigt den Startbildschirm der GesturePad-Anwendung. Die Benutzeroberfläche besteht nur aus drei Elementen: dem Eingabefeld für die IP-Adresse des Simulators, der Set-Taste und der Berührungsfläche.

Die *GesturePadView*-Komponente ermöglicht die Steuerung von allen vier Parametern (Alpha, Radius, Höhe und Neigungswinkel). Die Steuerung erfolgt über unterschiedlichen Gesten.

- Um die Parametern *Alpha* und *Radius* zu ändern, muss man ein Finger entlang *X-Achse* und *Y-Achse* entsprechend bewegen (Abbildung 3.9). Das Prinzip ist mit dem Joystick-Prinzip vergleichbar, mit der Ausnahme, dass der Mittelpunkt in diesem Fall der Startpunkt der Bewegung ist.

- Die *Höhe* ändert man mit einer Pinch-Geste. Um den Wert zu erhöhen, muss man zwei Finger auf der Berührungsfläche gedrückt halten und dann auseinanderziehen, um den Wert zu verringern, muss man umgekehrt die Finger zusammenziehen (Abbildung 3.10).

- Der *Neigungswinkel* ändert man mit einer Zweifingerbewegung entweder nach oben oder nach unten (Abbildung 3.11).

Um die oben genannte Gesten zu erkennen und zu behandeln, muss der Ereignis-Listener von *GesurePadView*-Komponente wie folgt eingestellt werden. Die folgende Ereignisse werden bearbeitet:

- Das Ereignis *MotionEvent.ACTION_DOWN* erscheint, sobald die Berührungsgeste angefangen hat. An dieser Stelle werden die Startkoorinaten der Bewegung (*start_x*, *start_y*) gespeichert und die Flag-Variable *moveState* wird auf „true“ gesetzt.

```
1 case MotionEvent.ACTION_DOWN:
2     start_x = (int) event.getRawX();
3     start_y = (int) event.getRawY();
4     moveState = true;
5     break;
```

- Das Ereignis *MotionEvent.ACTION_POINTER_DOWN* erscheint, sobald der zweite Finger hat den Bildschirm betroffen. Dieses Ereignis bedeutet der Anfang der Bewegung der beiden Finger. An dieser Stelle wird die Y-Koordinate des ersten

Fingers als Startkoordinate gespeichert. Die Flag-Variable *move2FState* wird auf „true“ gesetzt. Die Flag-Variable *moveState* wird auf „false“ gesetzt.

```
1  case MotionEvent.ACTION_POINTER_DOWN:
2     move2FState = true;
3     moveState = false ;
4     start_y = (int) event.getY (0);
5     break;
```

- Das Ereignis *MotionEvent.ACTION_MOVE* erscheint, sobald die Fingerkoordinaten sich geändert haben. An dieser Stelle prüft die Methode zuerst, ob es um die Einfinger-Bewegung oder um die Zweifinger-Bewegung geht. Die Flag-Variablen *moveState* und *move2FState* zeigen die Art der Bewegung. Die Differenz zwischen den Start- und aktuellen Koordinaten wird auf die Zulässigkeit geprüft und an den Simulator, entsprechend der Art der Bewegung, geschickt.

```
1  case MotionEvent.ACTION_MOVE:
2     if (moveState) {
3        if (Math.abs(event.getRawY() − start_y) > AXIS_DEADZONE)
4           sender_radius.queue.offer (Float.toString (event.getRawY() − start_y ));
5        if (Math.abs(event.getRawX() − start_x) > AXIS_DEADZONE)
6           sender_alpha.queue.offer (Float.toString (event.getRawX() − start_x ));}
7     if (move2FState) {
8        if (Math.abs(start_y − (int) event.getY (0)) > AXIS_DEADZONE ){
9           sender_phi.queue.offer (Float.toString (event.getY (0) − start_y ));}}
10    break;
```

- Das Ereignis *MotionEvent.ACTION_POINTER_UP* erscheint, sobald der zweite Finger den Bildschirm verlassen hat. An dieser Stelle wird die Flag-Variable *move2FState* auf „false“ gesetzt und es wird der Nullwert an den Simulator geschikt.

```
1  case MotionEvent.ACTION_POINTER_UP:
2     move2FState = false ;
3     sender_phi.queue.offer (Float.toString (0));
4     break;
```

- Das Ereignis *MotionEvent.ACTION_UP* signalisiert über den Abschluss der Berührungsgeste. An dieser Stelle wird die Flag-Variable *moveState* auf „false" gesetzt. An den Simulator werden die Nullwerte geschickt. [andrdmoevent, 2015]

```
1 case MotionEvent.ACTION_UP:
2     sender_radius.queue.offer(Float.toString(0));
3     sender_alpha.queue.offer(Float.toString(0));
4     moveState = false;
5     break;
```

Um die Pinch-Geste zu erkennen, muss die Instanz von *android.view.ScaleGestureDetector* Klasse erzeugt werden. Die Ereignis-Listener ist wie folgt eingestellt. Die Override-Methode *onScale()* wird aufgerufen, sobald eine Pinch-Geste erscheint. Der Wert von *detector.getScaleFactor()* wird an den Simulator geschickt. [andrdscldet, 2015]

```
1 mScaleDetector = new ScaleGestureDetector(this.getContext(), new ScaleListener());
2 private class ScaleListener extends ScaleGestureDetector.SimpleOnScaleGestureListener {
3     @Override
4     public boolean onScale(ScaleGestureDetector detector) {
5         sender_height.queue.offer(Float.toString(detector.getScaleFactor()));
6         return super.onScale(detector);  }}
```

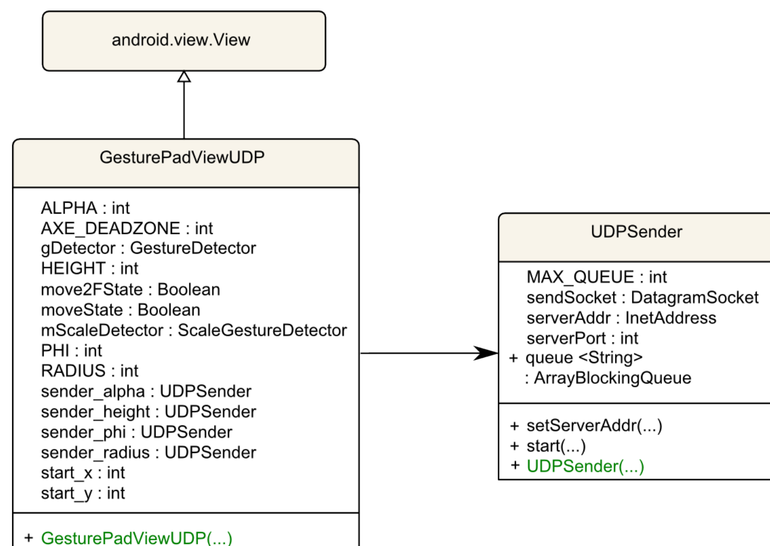


Abbildung 3.8: Klassendiagramm für GesturePadViewUDP Klasse

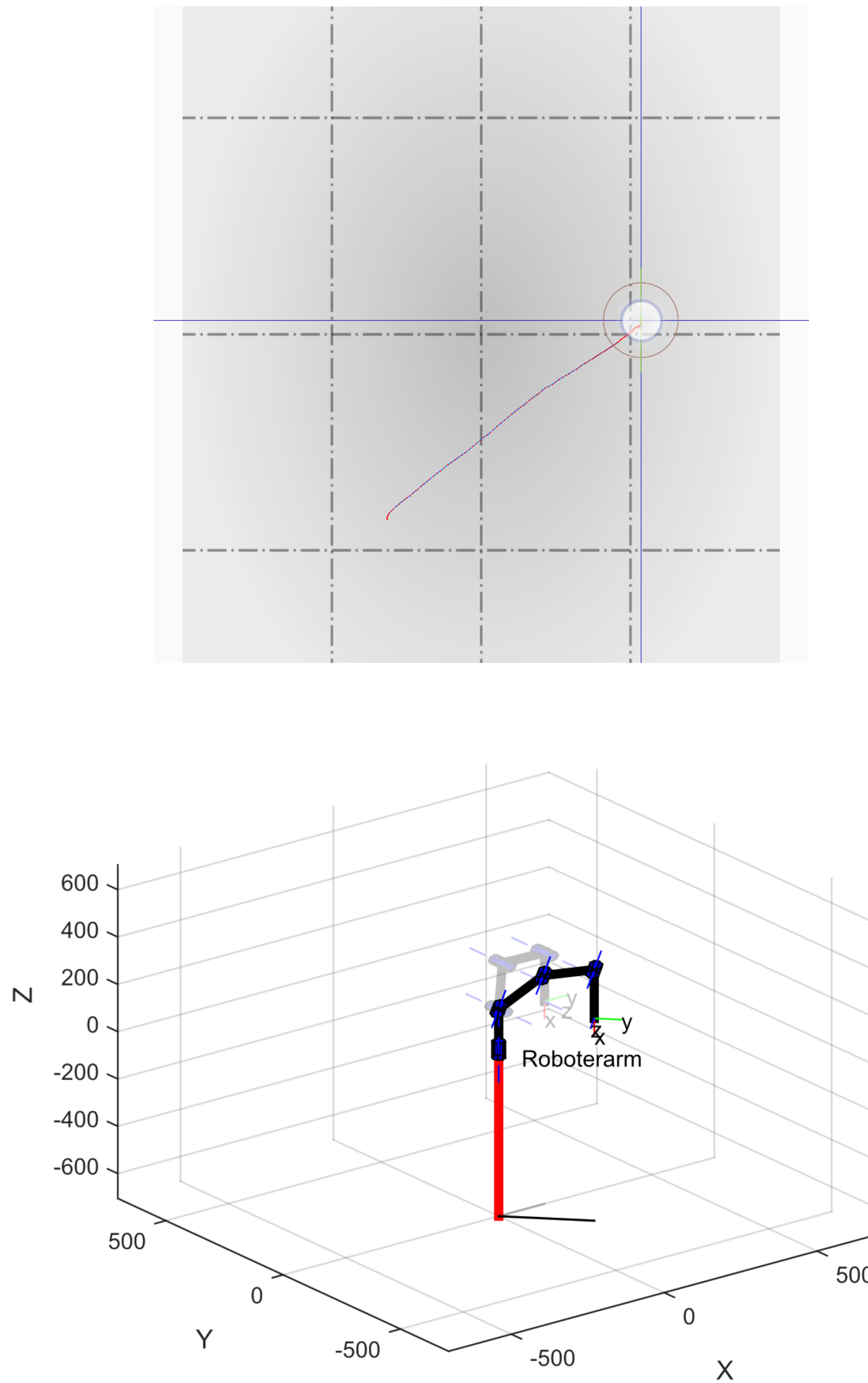


Abbildung 3.9: Die Bewegung des Roboterarmes in einer radialen Ebene

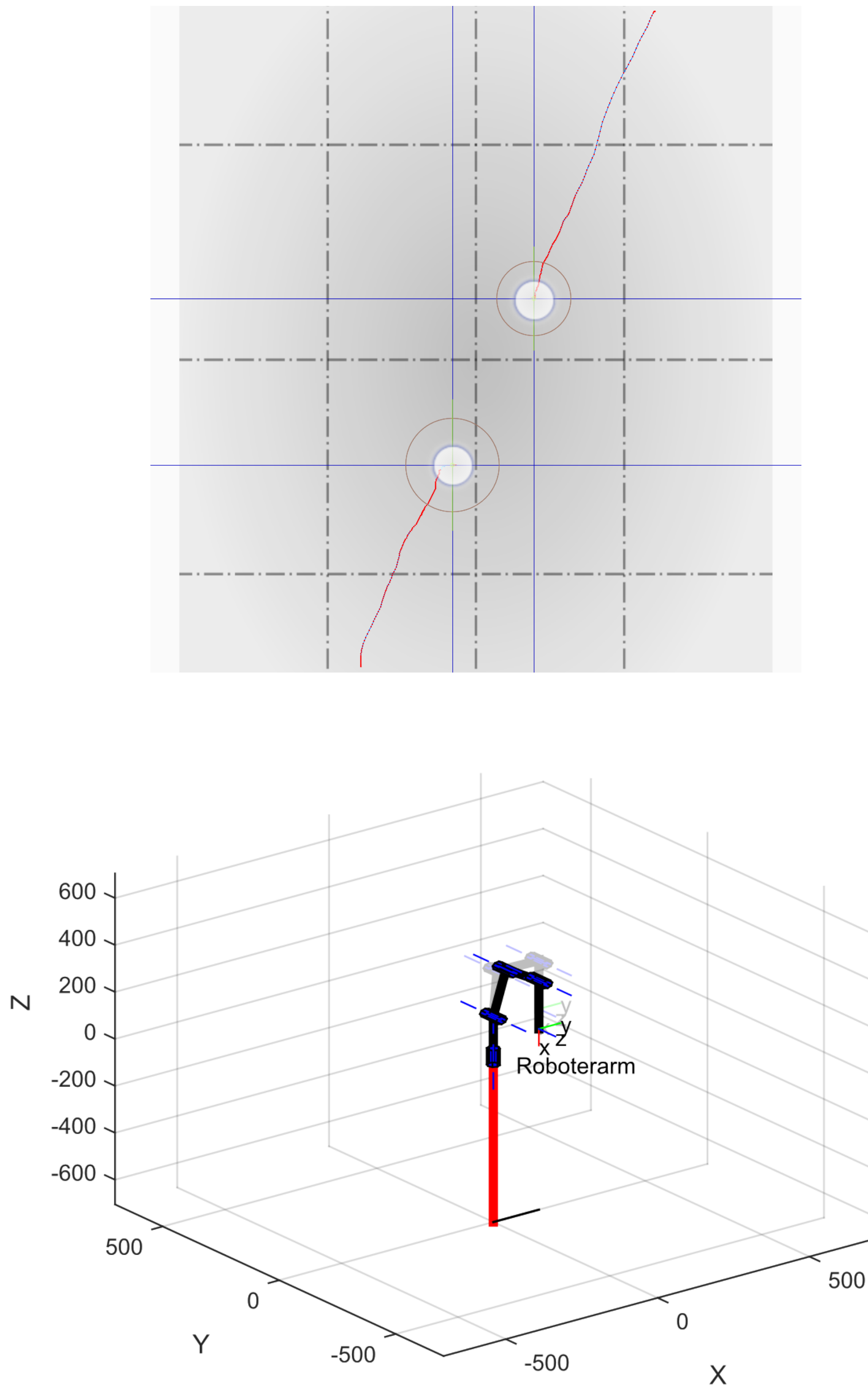
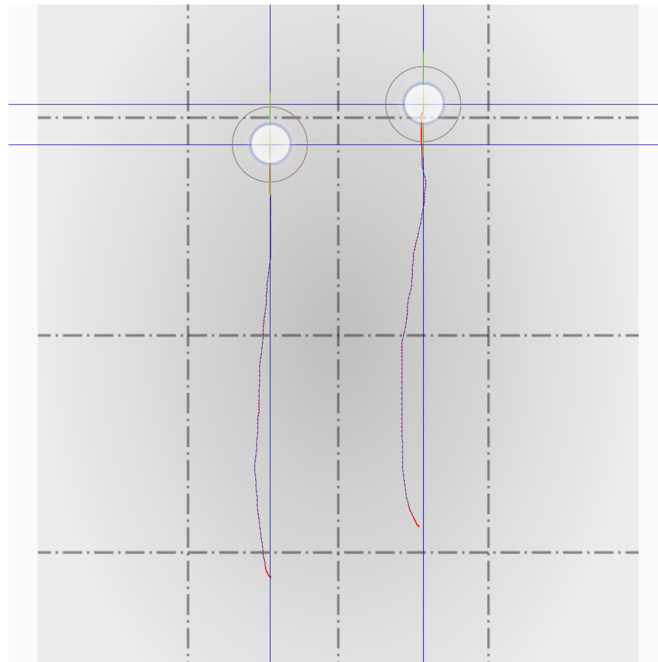


Abbildung 3.10: Die Änderung des *Höhe*-Parameters



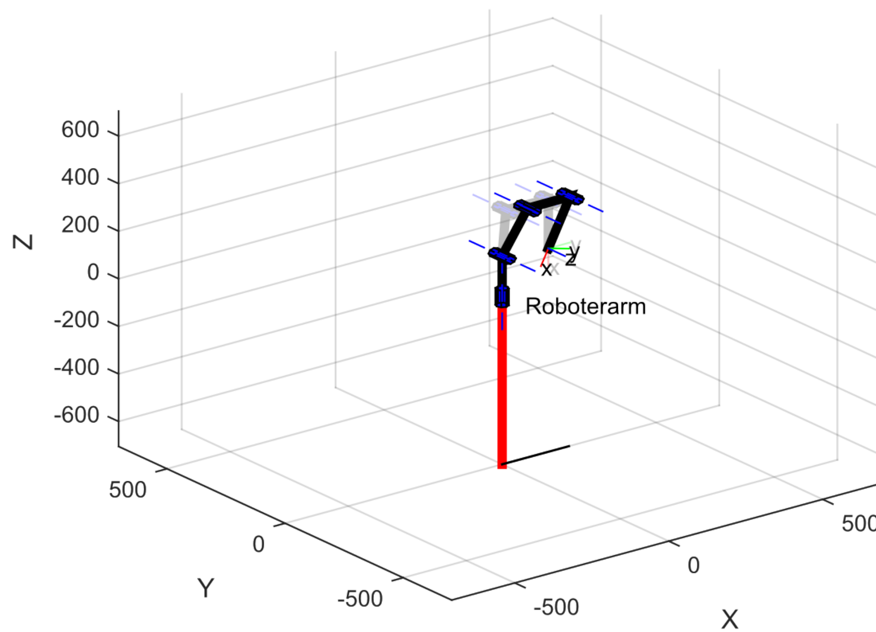


Abbildung 3.11: Die Änderung des *Neigung*-Parameters

3.5 Antriebssteuerung mit Beschleunigungssensor

Um eine ROS-fähige Android-Anwendung zu entwickeln, muss die Hauptaktivität (Main Activity) die Klasse *org.ros.android.RosActivity* erweitern. Die Override-Methode *init()* ist der Einstiegspunkt für das Programm und wird aufgerufen, sobald die Activity mit Master URI initialisiert ist und *NodeMainExecutorService* gestartet hat. An dieser Stelle werden die Knoten konfiguriert und gestartet.

```
1 @Override
2 protected void init (NodeMainExecutor nodeMainExecutor) {
3     aTalker = new AccelerationTalker(this );
4     NodeConfiguration nodeConfiguration = NodeConfiguration.newPublic(
5         InetAddressFactory.newNonLoopback().getHostAddress());
6     nodeConfiguration.setMasterUri(getMasterUri ());
7     nodeMainExecutor.execute(aTalker, nodeConfiguration );}
```

Die Klasse *AccelerationTalker* implementiert das Interface *org.ros.node.NodeMain* und realisiert die Funktionalität des ROS-Knotens. Die Hauptaufgabe dieses Knotens ist das Lesen von Daten von dem Beschleunigungssensor und das Senden dieser Daten an den Simulator Turtlebot.

Die Override-Methode *onStart()* ist der Einstiegspunkt des Knotens. An dieser Stelle wird das ROS Topic (Thema) *turtle1/command_velocity* erstellt und veröffentlicht. Als der Nachrichtentyp für dieses Topic wird der Typ *turtlesim/Velocity.msg* definiert.

```
1 final Publisher<turtlesim.Velocity> publisher =
2     node.newPublisher("turtle1/command_velocity", turtlesim.Velocity._TYPE);
```

Als Nächstes wird eine stornierbare Schleife *org.ros.concurrent.CancellableLoop* erstellt und gestartet. Die Aufgabe dieser Schleife ist die aktuelle Sensordaten zu veröffentlichen. Die Override-Methode *loop()* ist der Körper der Schleife. Hier wird eine neue Nachricht mit aktuellen Sensordaten erstellt und geschickt.

```
1 @Override
2 protected void loop() throws InterruptedException {
3     turtlesim.Velocity turtle_vel_msg = publisher.newMessage();
```

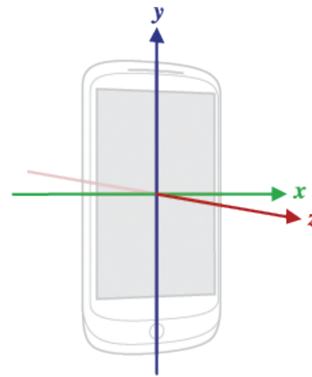


Abbildung 3.12: Die Achsen des Beschleunigungssensors [andrdsens, 2015]

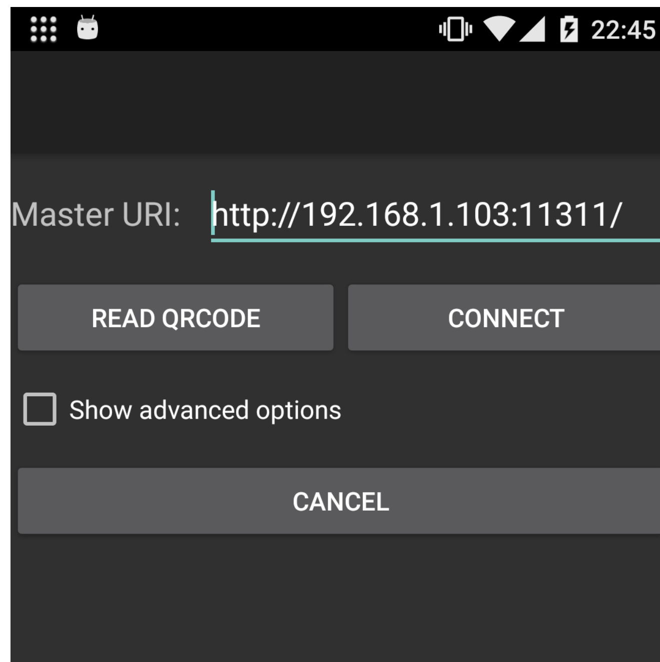
```
4    float [] aValue = aListener .getSensorValue ();
5    turtle_vel_msg .setAngular(aValue [0]);
6    turtle_vel_msg . setLinear (−1*aValue [1]);
7    publisher . publish ( turtle_vel_msg );}
```

Die Nachricht *turtlesim.Velocity* besteht aus zwei Felder, die die Winkel- und die Lineargeschwindigkeit des Roboters darstellen. Die Felder werden mit den Sensordaten befühlt. Diese Daten erzeugt das Objekt *AccelerationListener*, das das Interface *android.hardware.SensorEventListener* implementiert.

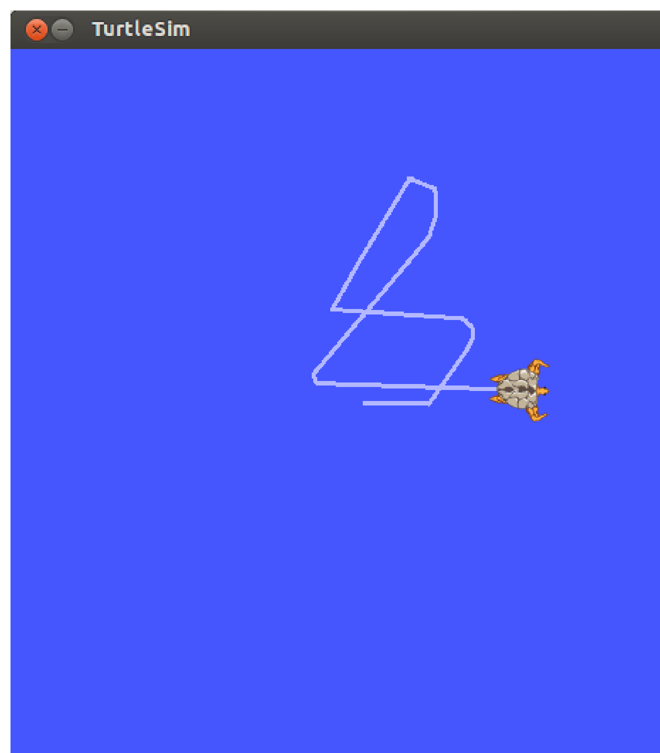
```
1  public class AccelerationListener implements SensorEventListener {
2  @Override
3  public void onSensorChanged(SensorEvent event) {
4    if (event . sensor . getType() == Sensor . TYPE_ACCELEROMETER) {
5        acceleration [0] = event . values [0];
6        acceleration [1] = event . values [1];
7        acceleration [2] = event . values [2]; }}}
```

Das Sensor-Ereignis des Typs *Sensor.TYPE_ACCELEROMETER* enthält die Array von Werten, die die Beschleunigungen auf den drei Achsen des Gerätes abbilden (Abbildung 3.12). Die Beschleunigungen auf X- und Y-Achsen werden als Winkel- und Lineargeschwindigkeiten interpretiert und an den Turtlebot Simulator zugeschickt.

Die Abbildung 3.13 zeigt der Startbildschirm der Anwendung und der Screenshot des Simulationsfensters.



(a) MasterChooser UI: Startbildschirm der Anwendung



(b) Screenshot von Turtlebot Simulator

Abbildung 3.13: Antriebssteuerung mit Beschleunigungssensor

4 Zusammenfassung

Die entwickelten Anwendungen zeigen ein hohes Potenzial des mobilen Geräts in dem Robotikbereich. Aber die tatsächliche Verwendung in realer Umgebung ist noch nicht möglich, da die Programme nur mit Simulatoren getestet sind.

Für ein komplexes Robotersystem können die Steuerkomponenten kombiniert werden. Allerdings macht das die grafische Oberfläche weniger intuitiv. In diesem Fall muss der Usability-Test durchgeführt werden.

Die Softwareplattform Android erlaubt auch die Videowiedergabe. Das kann sehr hilfreich sein für die Roboter, die mit der Videokamera ausgestattet sind. Auch die Darstellung von Navigationskarten und Sensordaten ist möglich und kann die Funktionalität der Steuereinheit deutlich erweitern. Die in dieser Arbeit erstellten Anwendungen sind eine gute Basis für solche Weiterentwicklungen.

Literaturverzeichnis

[Aaron Martinez 2013] Aaron Martinez, Enrique F.: *Learning ROS for Robotics Programming*. Packt Publishing, 2013 (Community experience distilled). – ISBN 9781782161448

[andrdmoevent 2015] : *MotionEvent | Android Developers*. 2015. – URL http://developer.android.com/reference/android/view/MotionEvent.html

[andrdscldet 2015] : *ScaleGestureDetector | Android Developers*. 2015. – URL http://developer.android.com/reference/android/view/ScaleGestureDetector.html

[andrdseekb 2015] : *SeekBar | Android Developers*. 2015. – URL http://developer.android.com/reference/android/widget/SeekBar.html

[andrdsens 2015] : *SensorEvent | Android Developers*. 2015. – URL http://developer.android.com/reference/android/hardware/SensorEvent.html

[blkqueue 2014] : *BlockingQueue (Java Platform SE 7)*. 2014. – URL https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html

[Corke 1996] Corke, P.I.: A Robotics Toolbox for MATLAB. In: *IEEE Robotics and Automation Magazine* 3 (1996), März, Nr. 1, S. 24–32

[googleio11 2011] : *Google I/O 2011: Cloud Robotics, ROS for Java and Android | Willow Garage*. 2011. – URL

http://www.willowgarage.com/blog/2011/05/12/google-io-2011-cloud-robotics-ros-java-and-android?page=7

[Joseph 2015] JOSEPH, L.: *Learning Robotics Using Python*. Packt Publishing, 2015 (Community experience distilled). – ISBN 9781783287543

[matlabudp 2015] : *Create UDP object - MATLAB udp - MathWorks Deutschland*. 2015. – URL http://developer.android.com/reference/android/hardware/SensorEvent.html

[Morgan Quigley 2009] MORGAN QUIGLEY, Brian G.: *ROS: an open-source Robot Operating System*. 2009. – URL https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf

[rosjava 2013] : *Getting started — rosjava_core 0.1.6 documentation*. 2013. – URL http://rosjava.github.io/rosjava_core/latest/getting_started.html

[rosjavagit 2015] : *rosjava/rosjava_core · GitHub*. 2015. – URL https://github.com/rosjava/rosjava_core

[roswikiconcepts 2013] : *de/ROS/Concepts - ROS Wiki*. 2013. – URL http://wiki.ros.org/de/ROS/Concepts

[roswikiintro 2013] : *de/ROS/Introduction - ROS Wiki*. 2013. – URL http://wiki.ros.org/de/ROS/Introduction

[roswikimaster 2012] : *Master - ROS Wiki*. 2012. – URL http://wiki.ros.org/Master

[roswikimsg 2015] : *Messages - ROS Wiki*. 2015. – URL http://wiki.ros.org/Messages

[roswikinodes 2012] : *Nodes - ROS Wiki*. 2012. – URL http://wiki.ros.org/Nodes

[roswikiparam 2013] : *Parameter Server - ROS Wiki*. 2013. – URL http://wiki.ros.org/Parameter%20Server

[roswikirxgraph 2013] : *rxgraph - ROS Wiki*. 2013. – URL http://wiki.ros.org/rxgraph

[roswikiservice 2012] : *Services - ROS Wiki*. 2012. – URL http://wiki.ros.org/Services

[roswikitopic 2014] : *Topics - ROS Wiki*. 2014. – URL http://wiki.ros.org/Topics

[xmlrpc 1999] : *Simple cross-platform distributed computing, based on the standards of the Internet*. 1999. – URL http://xmlrpc.scripting.com/

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

Hamburg, 30. Oktober 2015 ____________________
Stanislaw Rasowski