hochschule mannheim

Manuel Bäurle, Jan-Philipp Willem, Joscha Zander, Severin Kohler

# Realisierung eines modularen Kartierungsverfahrens unter ROS basierend auf Docker

**Projektdokumentation Robotik**


Hochschule Mannheim

Dozent: Prof. Dr. Thomas Ihme

Mannheim, Juli 2016

# Inhaltsverzeichnis

# Abstract

Motivation für diese Ausarbeitung war ein Projekt im Rahmen der Vorlesung Robotik an Institut für Robotik an der Hochschule Mannheim. Eines der zur Verfügung stehenden Themen war die automatisierte Kartierung der WLAN-Abdeckung in den Gebäuden und Räumen der Hochschule unter Verwendung des Robot Operating System. Der Ansatz bestand darin die bestehende Softwareintallation weiter auszubauen und um eine WLAN-Karteriungsfunktion zu erweitern. Es wurde allerdings schnell ein Punkt erreicht an dem unsere Kentnisse über die Installierte Software nicht ausreichte um eine lauffähige Umgebung für unsere Anforderungen zu schaffen. Wir entschieden uns deshalb dafür das Basissystem für den Pioneer und für die Kartierungsanwendungen neu aufzusetzten. Hierbei wollten wir vor allem Wert darauf legen dass nachfolgenden Projekten und Arbeiten eine solide Grundlage zur Verfügung steht, in die sich jeder, auch ohne große Vorkenntnisse, schnell einarbeiten kann. Wir sind dann recht schnell auf das Docker-Projekt gestoßen, welches genau für diesen Anwendungszweck entwickelt worden ist. Mit Docker haben wir die komeplexe ROS-Infrastuktur sauber in einzelne Subsysteme aufgeteilt. Ein weiterer Vorteil hierbei ist, dass sämtliche Installationsroutinen über eine Art Rezept bzw. Konfigurationsdatei jederzeit einsehbar und veränderlich sind. Dies erlaubt dem Entwickler auf einen Blick zu sehen welche Software für das Untersystem installiert wird und wie sie konfiguriert ist. Um den Einstieg in die Thematik weiter zu vereinfachen wollen wir in der folgenden Arbeit auf unser Projekt eingehen, die Grundlagen erläutern und versuchen unsere Umsetzung für andere verständlich zu machen.

# 1 Einleitung

Eine vollständige WLAN-Abdeckung ist in der heutigen Zeit für eine Hochschule enorm wichtig. Nahezu alle Fachbereiche nutzen die digitalen Medien für Ihre Vorlesungen und Labor-/Übungsarbeiten. In der Praxis ist es allerdings schwierig eine solche Abdeckung ohne weiteres zu erreichen, da viele Faktoren bei der tatsächlichen Verfügbarkeit des WLAN-Netzwerks eine Rolle spielen. Dazu gehört zum Beispiel der Grundriss der Gebäude oder die eingeschränkten Platzierungsmöglichkeiten für die Router und Access-Points. Auch schwankt die Verfügbarkeit je nach Auslastung des Netzwerks. Ein Ansatz dieses Problem zu lösen besteht darin die aktuelle Verfügbarkeit zu messen und so Lücken in der Vorsorgung aufzuspüren. Per Hand wäre dies allerdings für das gesamte Hochschulgelände sehr mühsam. Deshalb wollen wir mit unserer Arbeit die Grundlage dazu schaffen diese Aufgabe autom von einem Roboter übernehmen zu lassen.
Docker war ursprünglich nicht Teil der geplanten Arbeit, es hat sich allerdings herausgestellt das dies eine Willkommene Erweiterung bzw. Verlagerung unseres Projekts darstellt. Aus diesem Grund befasst sich ein Großteil der Arbeit mit den Eigenschaften und Funktionen für Docker. Hierbei wird versucht sich auf die nötigen Grundlagen zu beschränken, um die Komplexität unserer Ausarbeitung nicht noch weiter zu erhöhen und Neulingen einen guten Überblick über die Thematik zu verschaffen.
Die Ausarbeitung beginnt mit den nötigen Grundlagen. Danach wird auf die Problemstellung und Realiserung eingegangen. Abschließend wird versucht die Arbeitsergebnisse zusammenzufassen und einen Ausblick auf weiterführende Arbeiten zu geben.

# 2 Grundlagen

Dieses Kapitel soll dazu dienen einen Überblick über die eingesetzten Methoden und Lösungsansätze zu bekommen. Dadurch soll vor allem Lesern die keine umfangreichen Vorkenntnisse über die verwendeten Verfahren verfügen ein Einstieg ermöglicht werden.

## 2.1 Pioneer 3-DX


Abbildung 2.1: Pionner 3-DX

Der Pioneer 3-DX ist ein kleiner, zweirädriger Übungs- und Forschungsroboter welchen wir zur Realisierung unseres Projektes eingesetzt haben. Neben der Standardausstattung wurde unser Bot noch mit einem Laserscanner

(Sick LMS-200), einer Kinect-Sensorleiste[1], einem kleinen Computer (Linux/Ubuntu 14.04) und Bumper-Kontaktsensoren [2] an Front und im Heck ausgestattet. (Abbildung 2.1).

Zur späteren 2D-Kartierung der Räume wird hierbei der Laserscanner und die Odometrie eingesetzt. Die WLAN-Feldstärke wird mit einem üblichen WLAN-Stick analysiert. Der WLAN-Stick und die Kinect werden über den USB-Port angesprochen. Motor, Bumper und Ultraschallsensoren werden mithilfe das Pioneer Steuerboards (Mikrocontroller) über eine serielle Schnittstelle angesprochen.

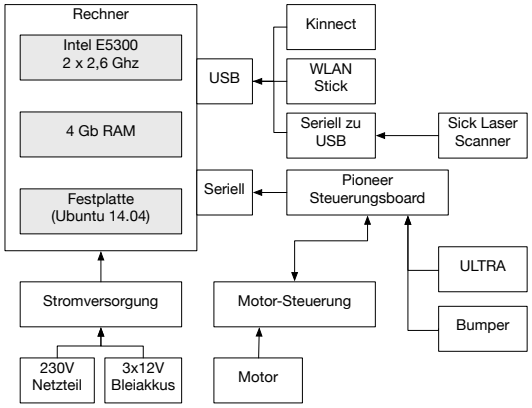


Abbildung 2.2: Blockschaubild des Pioneersystems

Der Laserscanner wird mit einem SerialToUsb-Konverter an den USB-Port des Linux-Rechners angeschlossen (Abbildung 2.2). Im Entwicklungsbetrieb wurde der Roboter hauptsächlich über ein Netzteil(12V) mit Strom versorgt. Im Live-/Testbetrieb empfiehlt es sich allerdings auf Grund der eingeschränkten Mobilität auf passende 12V Blei-Akkumulatoren umzusteigen.

[1]Kinect: Hardware mit PrimeSense-Tiefensensor, 3D-Mikrofon, Farbkamera die Objekt Erkennung ermöglicht (für unser Vorhaben nicht relevant)
[2]Bumper: Kollisionsflächen um Kollisionen mit Hindernissen festzustellen

## 2.2 Robot Operating System

Das Robot Operating System ist ein flexibles, abstrahierendes Framework um Software speziell für Roboter zu entwickeln. Es ist eine Sammlung aus Werkzeugen, Bibliotheken und Konventionen um das Verfassen von komplexen Robotersystemen so einfach und modular wie möglich zu gestalten. Durch die Abstrahierung der Soft- und Hardware wird eine hohe Kompatibilität der Software erreicht, so dass diese ggf. auch in anderen System eingesetzt werden kann. Installieren kann man es entweder manuell, oder ganz einfach über die jeweilge Paketverwaltung des Betreibssystems.

ROS besitzt eine eigene Paketverwaltung. Diese ermöglicht es fertige Pakte einfach und schnell in das eigene System zu integrieren. Für viele Anwedungsmöglichkeiten gibt es bereits fertige Packages die meisten auch einen offenen Quellcode besitzen. Ein Ros-package besteht normal aus Bibliotheken, Nodes und Script-/Konfigurationsdateien auf. Es kann aber auch Treiber oder andere Software (Demos, Simulationen, ...) enthalten.

### 2.2.1 Nodes

Nodes sind Prozesse die eine Berechnung tätigen. Dabei werden diese zusammen kombiniert und kommunizieren über Topics, Remote Procedure Call Services oder den Parameter Server. Beispielsweise gibt es eine Node für die Laser Entfernungs-/ und Transformationsdaten, ein weiteres für die Kontrolle der Radmotoren und zum Path-Planning mithilfe der Odometriedaten. Durch das Zusammenspiel können also komplexe Aufgaben gelöst werden. Die Nodes stellen der Applikation die Logik und Funktion zur Verfügung.

### 2.2.2 Topics

Topics dienen zur Kommunikation zwischen einzelnen Nodes und der Kommunikation mit dem Master (roscore). Die Kommunikation wird über das Publisher/Subscriber Pattern abgebildet. Benötigt eine Node eine Kommunikation mit z.B. einer anderen Node meldet (subscribe) diese sich einfach bei der anderen an. Gibt es eine Nachricht zu übertragen werden dann alle Subscriber benachrichtigt (publish). Hierbei wird per TCP/IP [3] oder UDP[4] kommuniziert. TCP/IP basierte Kommunikation (TCPROS) sendet die Nachrichten über persistente TCP/IP Verbindungen. Die UDP Kommunikation(UDPROS) wird nur in roscpp[5] unterstützt und separiert Nachrichten in UDP Pakete. Hierbei bestimmen die Nodes ihre Kommunikationsart erst zur Laufzeit. Falls ein Node also UDPROS bevorzugt jedoch der Kommunikationspartner dies nicht unterstützt, kann dieser auch noch zur Laufzeit auf TCP/IP wechseln (fallback).

### 2.2.3 Master

Der Master ist der Namens- und Registrierungsservice für die Kommunikation der Nodes. Dieser findet Publisher und Subscriber für die einzelnen Topics und verwaltet diese. Er wird vom Roscore gestartet, dieser ist das Kernstück von ROS und enthält verschiedene Programme und Nodes die unabdinglich für die Verwendung von ROS sind. Dieser startet z.B. den ROS Master, ROS Parameter Server und den rosout logging node. Der Roscore muss gestartet sein um ROS-Systeme lauffähig zu machen.

### 2.2.4 Kommunikations-Beispiel

Dieser Sachverhalt ist an der (Abbildung 2.3) nochmals veranschaulicht. Wir haben uns für ein einfaches, anschauliches Beispiel einer Fotokamera entscheiden.

[3] TCP(Transmission Control Protocol)/IP(Internet Protocol): Familie von Netzwerkprotokollen
[4] UDP(User Datagram Protocol): ist ein minimales, verbindungsloses Netzwerkprotokoll
[5] roscpp: C++ Implementierung von ROS

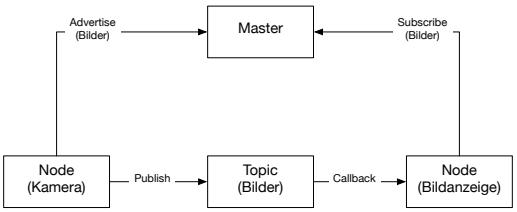


Abbildung 2.3: Beispiel Kommunikation ROS-intern

Hier stellen die Bilder das Topic dar, die Kamera veröffentlicht dieses auf der Node. Die Bildanzeige bekommt ein Callback vom Publisher und zeigt das Bild nachdem es geschossen wurde an. Die Bildanzeige und die Kamera können also miteinander über das Topic kommunizieren. Der Master verwaltet den publish und subscribe Vorgang.

### 2.2.5 Launch-Dateien

Die .launch-Files spezifizieren welche welche Nodes/Topics gestartet werden sollen. Man kann sie als *Rezept* für die gewünschte Anwendung betrachten, welches gleichzeitig auch zum System-Start dient. Außerdem ist es möglich neben den auszuführenden Unterprogrammen auch Parameter für die Laufzeit zu setzten. Mithilfe von Launch-Files können mehrere Nodes bzw. Sub-Systeme mit einem Aufruf gestartet werden. Es können auch zusätzlich noch andere ROS-Attribute definiert werden. Launch-Files werden mit dem Programm roslaunch gestartet. Die Launch-Files lassen sich außerdem verschachteln, d.h. sie können vom Inhalt anderer Launch-Files erben (include).

## 2.3 Docker

Docker ist eine Software zum automatisierten Deployment[6] von Applikationen innerhalb von sogenannten Containern[7]. In diesen wird alles was zum Ausführen der Software im Container benötigt wird beschrieben. Dazu gehören vor allem Installationsroutinen, aber auch das setzen von Umgebungsvariablen und das starten der Softwarepakte. Hierbei teilen sich die Container einen zentralen Docker-Kernel[8] und laufen als isolierte Prozesse auf dem Host Betriebssystem. Die Container sind dabei nicht an eine spezifische Hardware-Infrastruktur gebunden, dies hat den Vorteil das sie auf jedem Heimrechner und Server Funktionsfähig sind welche die Systemanforderungen für Docker erfüllen (Abbildung 2.4).

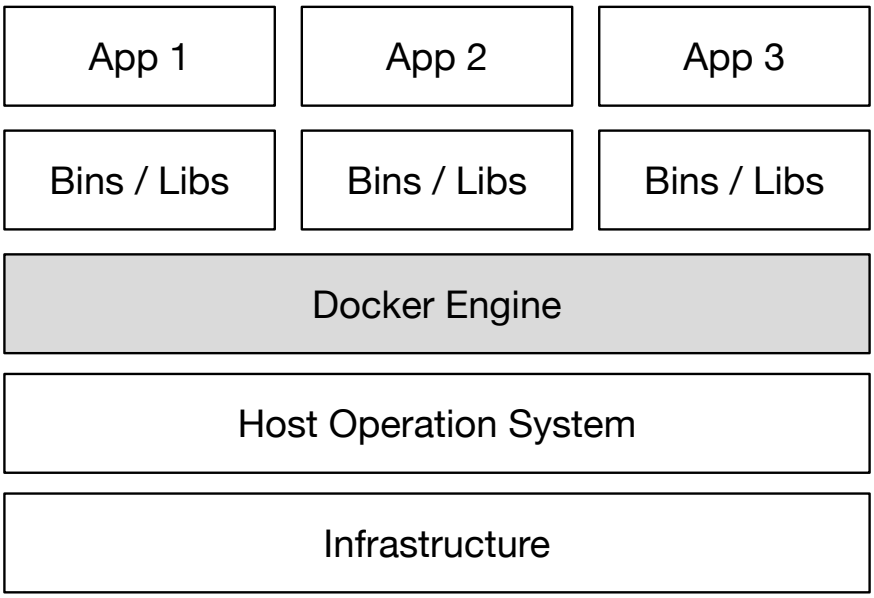


Abbildung 2.4: Docker Aufbau

Mithilfe der Docker-Images[9] können beliebige Kombinationen von Con-

[6] Routine zur Installation von Softwaresystemen auf verschiedenen Plattformen
[7] In Docker-Containern werden die fertig gebauten Subsysteme aufgeführt (Docker-Images)
[8] Kernel auch Betriebssystemkern, zentraler Bestandteil eines (Unter-)Betriebssystems
[9] Fertig gebautes (Unter-)System

tainern erstellt werden. Diese können je nach Anwendungszweck individualisiert werden. Änderungen bleiben hierbei begrenzt auf den jeweiligen Container. Um das Verhalten eines Containers zu steuern muss das jeweilige Docker-File angepasst werden und anschließend das Image neu gebaut werden. Das Docker-File ist eine Art von Konfigurationsdatei-/rezept und enthält eine Reihe von Befehlszeilen die automatisch ausgeführt werden um das Image zu erstellen.

Um mehrere Container der Images gleichzeitig zu starten und mit dem virtuellen Netzwerk zu verbinden existiert das Tool Docker-Compose. Darin können weitere, systemspezifische Konfigurationen unabhängig der eingebundenen Dockerfiles angegeben werden. Beispielsweise Port-forwarding[10] oder Freigaben für die Hardware des Host-Systems. Hierbei werden Abhängigkeiten der einzelnen Images bestimmt und dementsprechend die Container gestartet. Dadurch entfällt das ggf. aufwendige Starten der einzelnen Container. Diese Schritte werden in der docker-compose.yml[11] einmalig konfiguriert und können anschließend mit einem Befehl ausgeführt werden.

Des weiteren bietet Docker einen eigenen Repository-Host speziell für Dockerfiles an (Docker-Hub)[12].

Die von unser Entwickelte Umgebung, sowie eine Anleitung zur Installation des Systems findet sich auf github (`https://github.com/jwillem/docker-ros`). Es ist außerdem geplant das Repository auf dem github-Account der Hochschule zu hinterlegen (stand Juli 16).

[10] Weiterleitung einer Verbindung, die über einen Host auf einen bestimmten Port veröffentlicht werden

[11] Beispiel siehe Anhang

[12]

Außerdem gibt es viele Erweiterungsmöglichkeiten was die Architektur und Funktionalität des Grundsystems angeht. Zwei besonders nützliche Beispiele wollen wir hier nennen.
Eine Entwickler-Schnittstelle zu den Grundfunktionen von Docker bietet der libcontainer[13]. Durch ihn kann man Namespaces, Control Groups[14], Ressourcen, AppArmor-Profiles[15], Netzwerkinterfaces und Firewall-Regeln für Docker beeinflussen. Dabei ist dieser unabhängig von LXC[16] oder anderen, ähnlichen Lösungen welche auch versuchen virtuelle Untersysteme für Programme zu realisieren (*sandboxing*).

Außer dem Basispaket von Docker gibt es noch viele weitere Bibliotheken welche den Umfang der Funktionalität weiter erweitern. Um z.B. Docker-Contaier zu dezentralisieren gibt es die Bibliothek swarm[17], hierdurch kann ein Pool von Docker-Hosts zu einem Virtuellen Host zusammengefasst werden (Cloud-Infrastruktur). Docker-Netzwerkfunktionen können über libchan[18] angesprochen werden. Diese Bibliothek bietet die Möglichkeit das Netzwerk-Dienste miteinander über eine Kanäle kommunizieren können.

[13] Github: `https://github.com/opencontainers/runc/tree/master/libcontainer`
[14] Linux Kernel Feature das zur Limitierung von Ressourcen(CPU, Memory, Disk I/O, Network, etc.) für eine Gruppe von Prozessen dient.
[15] AppArmor Profile zur Konfiguration der Sicherheit von Applikationen
[16] LinuX Container (LXC) bieten die Möglichkeit, Prozesse und Prozessgruppen zu isolieren, indem Kernel-Ressourcen virtualisiert und gegeneinander abgeschottet werden
[17] Github: `https://github.com/docker/swarm`
[18] Github: `https://github.com/docker/libchan`

## 2.4 Kartierungsverfahren

Als Kartierungsverfahren bezeichnet man in unserem Fall die grafische Abbildung von Objekten und Raumverhältnissen in der realen Welt. Diese zeichnet z.B. eine Karte eines Stockwerks der Fakultät Informatik. Hierbei wird per Laserscanner und Odometrie kartiert und man spricht von laserbased SLAM[19].

### 2.4.1 Odometrie

Die Odometrie ist eine Methode zur Schätzung von Position und Orientierung des Roboters in der realen Welt. Hierbei wird anhand der Radumdrehung die zurückgelegte Strecke und deren Verlauf geschätzt. Im Alltag findet man dies in vereinfachter Form als Kilometerzähler in Kraftfahrzeugen. Hierbei wird die Anzahl n von Raddrehungen zwischen zwei Messzeitpunkten gezählt und zusammen mit dem bekannten Radumfang $\pi \cdot d$, zu der Wegdifferenz $\Delta s$ umgerechnet (3.1).

$$\Delta s = n \cdot \pi \cdot d \tag{2.1}$$

Für das Erfassen der Odometriedaten werden bei dem Pioneer 3-DX die Bewegungen, die Winkellage und die Position der Achse mit sogenannten Drehgebern gemessen. Der Pioneer 3-DX hat pro Rad einen optischen Quadratur-Encoder. Im Normalfall besteht ein Optischer Drehgeber aus 5 Teilen: einer LED, ein Kondensor, einer Rotationsscheibe, einer Abtastplatte und Photodioden. Zwischen der Leuchtdiode und den Photodetektoren befindet sich eine mit Schlitzen versehene Rotationsscheibe (Abbildung 2.5).

[19] Simultaneous Localization and Mapping

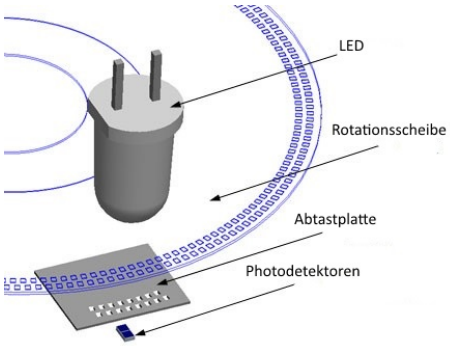


Abbildung 2.5: Aufbau eines optischen Drehgebers hier ohne Kondensor

Diese Platte rotiert wenn der Roboter fährt, dabei werden die Photodetektoren Abwechselnd beleuchtet. Aus diesen Ausgangssignalen kann der Empfänger Informationen über die Geschwindigkeit, die Distanz und die Position zurück errechnen ( Abbildung 2.6).

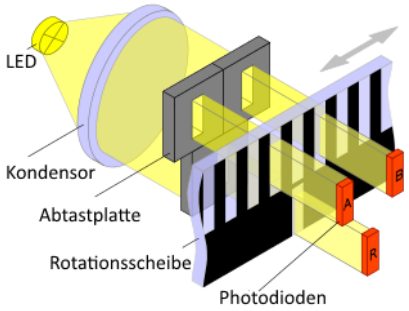


Abbildung 2.6: Schematischer Aufbau eines optischen Drehgebers

Im Falle des Pioneers 3-DX sind pro Drehgeber 2 Dioden und 2x2 Photodetekoren eingesetzt. Diese besitzen eine Phasenverschiebung von 90 Grad zueinander. Das dient dazu die Drehrichtung des Roboters genauer zu bestimmen und die Auflösung der Bewegungsdaten zu vervierfachen. Das Verfahren welches hier eingestzt wird nennt man Quadratur oder auch Quadratur des Kreises[20].

[20]Wikipedia: `https://de.wikipedia.org/wiki/Quadratur_des_Kreises`

Kurz gesagt minimiert man dadurch die numerische Ungenaugikeit der unendlichen Zahl $\pi$. Die Odometrie wertet diese Daten dann aus und reicht sie an die Kartierungssoftware weiter. Das Verfahren eignet sich aber nur als Komponente bzw. Hilfe für die Präzisierung der Laser und/oder anderer wellenbasierter Verfahren wie z.b. dem SONAR (Ultraschall) oder der Kinect (optisch/infrarot). Grund dafür ist die Ungenauigkeit (*schlupf*) der Odometriedaten, dem sog. Odometrie Drift. Dies kommt durch Reibung und Oberflächenanomalien wie z.B. Türschwellen zu Stande.

### 2.4.2 Laserscanner

Die 2D-Kartierung erfolgt bei diesem Projekt abgesehen von der Odometrie mithilfe des SICK LMS-200 Laserscanner. Hierbei arbeitet dieser durch Lichtstrahlen die der Scanner aussendet. Diese prallen an Hindernissen ab, werden also reflektiert und enden wieder bei der Empfangseinheit.

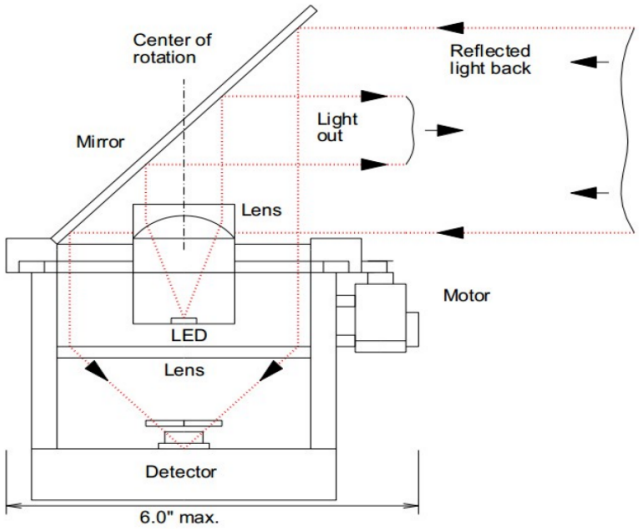


Abbildung 2.7: Schematischer Aufbau eines ORS-1 Laserscanners

Bei der Abbildung 2.7 handelt es sich um einen ORS-1 Laserscanner doch ist die Funktionsweise übertragbar auf den Sick LMS 200. Damit der Laser-

scanner nicht nur einen einzigen Abtaststrahl aussendet, muss dieser durch Umlenkspiegel fächerförmig ausgesendet werden( Abbildung 2.8 ).

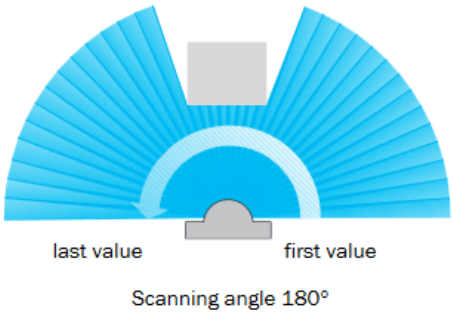


Abbildung 2.8: Richtung des Scannens bei maximalen Winkel des LMS 200

Der maximale Ausdehnungsradius beträgt hierbei bei dem LMS 200, 180 Grad. Des Weiteren ist der LMS 200 für einen Abstand von 10 Metern gebaut worden. Der Scanvorgang wird immer von der Standrichtung des Roboters von Rechts nach Links getätigt. Während des Scanvorgangs werden dabei hunderte Strahlen pro Sekunde ausgesendet. Die ausgewerteten Daten werden dann an den Rechner gesendet und können anschließend ausgewertet werden.

Dabei gibt es 2 Messverfahren. Das Impulsverfahren ist besonders gut für große Entfernungen geeignet. Hier wird ein der Trägerwelle aufgeprägter Impuls ausgesendet, ein von einer genauen Uhr gesteuerter Laufzeitmesser bestimmt die Zeitdifferenz $\Delta t$ zwischen dem Aussenden und dem Empfangen des reflektierten Impulses. Anhand dessen kann dann die Entfernung bestimmt werden. An die Zeitmessung werden hohe Anforderungen gestellt, so entspricht z.B. die zeitliche Auflösung 0,1 ns einem (einfachen) Weg von 1,5 cm. Das Phasenvergleichsmessverfahren dagegen sendet niederfrequenten Messwellen aus. An den Reflektierten Wellen beim Empfang wird die Phasendifferenz $\Delta\Phi$ zwischen ausgesandter und empfangener Welle gemessen. Daraus wird dann die Entfernung berechnet. Dieses Verfahren eignet sich besonderes für das exakte vermessen von komplizierteren Umgebungen die nicht zu großflächig sind.

### 2.4.3 SLAM

SLAM (Simultanious Localisation And Mapping) bezeichnet das komplexe Problem der Robotik eine unbekannte Umgebung zu erfassen und aufzuzeichnen (Kartierung), wobei ständig die eigene Position (Pose) des Roboters in dieser Karte ermittelt, bzw. geschätzt werden muss. Für SLAM existieren bereits viele Lösungsansätze[21] und ist ständig Teil der Robotik-Forschung.

Zum Erfassen der Umgebung werden 2D/3D bildgebende Verfahren verwendet. Eingesetzte Tiefenbildkameras [22] erzeugen hierbei Teilaufnahmen der Umgebung, die erst zusammen mit der bestimmten Position in Relation zueinander gesetzt werden können, um letztlich eine Karte zu erzeugen. Zur Positionsbestimmung können u.a. Odometrie, Landmarken (Beacon) und visuelle Sensorik kombiniert werden, um über Triangulation die aktuelle Pose zu ermitteln.
Bei der Kartierung (Mapping) werden die erfassten Umgebungsdaten mit Hilfe der bestimmten Position arrangiert und kombiniert. So entsteht iterativ eine Karte, die aufgrund der Ungenauigkeit der Schätzung zur Position jedoch verfälscht wird. Hierfür werden Optimierungsverfahren genutzt, die aktuelle Sensordaten mit bereits aufgezeichneten Kartenteilen abgleichen und z.B. Winkel und Skalierung korrigieren.

Für den Pioneer 3-DX ist durch die Odometrie die aktuelle Pose schätzbar und die Umgebung zeichnet der Laserscanner auf. Korrekturen des Odometrie Drifts müssen bei der Kartierung geschehen (siehe Kapitel 2.5.2).

## 2.5 Verwendete ROS-Pakete

Im Folgenden werden die wichtigsten verwendeten ROS-Pakete aufgelistet und erklärt. Ausführlichere Informationen zu den einzelnen Paketen sollten der aktuellen Dokumentation von ROS entnommen werden.

[21] http://openslam.org/ - Sammlung von open source Algoithmen und Tools für SLAM
[22] Punkt Wolken erzeugende Kameras, wie z.B. Laserscanner, ToF (Time of Light), Ultraschall oder Kinect.

### 2.5.1 gmapping

Das Paket gmapping[23] beinhaltet einen ROS Wrapper für GMapping [24] von OpenSLAM. Dieses unterstützt laser-based SLAM und ermöglicht so die Erstellung einer 2-D Karte aus Laser- und Odometriedaten.

### 2.5.2 heatmap

Das Paket heatmap[25] ermöglicht für eine gegebene Karte das autonome Ausmessen der WLAN-Feldstärken. Hierfür kann der Nutzer (z.B. in Rviz) in besagte gegebene Karte Bereiche einzeichnen, die der Roboter nach einer durchgeführten Wegplanung autonom abfahren kann. Messungen zur WLAN-Feldstärke werden einstellbar nach Entfernung zueinander durchgeführt. Diese Messpunkte werden abschließend über die zu Anfangs zugeteilten Bereiche interpoliert (siehe Abbildung 2.9).

Zur autonomen Wegplanung und Lokalisierung während der Abarbeitung der Wegpunkte wird das Paket amcl[26] genutzt.

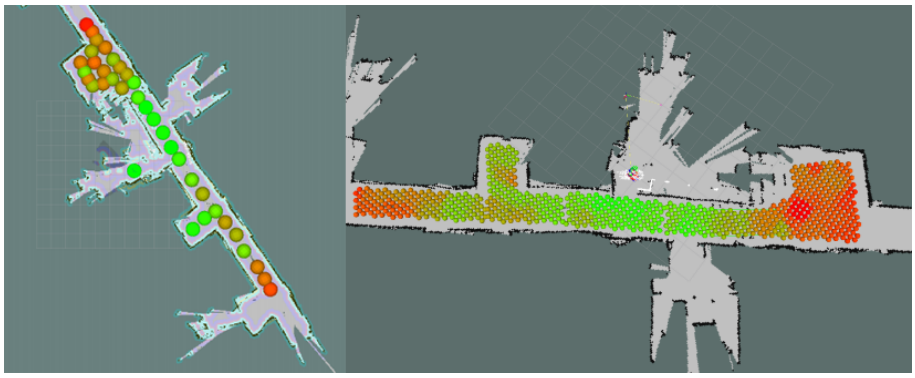

Abbildung 2.9: Beispiel Messpunkte der WLAN-Feldstärke (raw und interpoliert)

[23] ROS: `http://wiki.ros.org/gmapping`
[24] OpenSLAM: `http://openslam.org/gmapping.html`
[25] ROS: `http://wiki.ros.org/heatmap`
[26] adaptive monte carlo localization

### 2.5.3 p2os

Dieses Paket stellt u.a. die p2os Treiber (p2os_driver) zur Verfügung, welche die P2OS und ARCOS Firmware unterstützen. Somit dient p2os als Schnittstelle, um die Hardware des Pioneer 3-DX anzusprechen, auszulesen und zu Verarbeiten.

### 2.5.4 p2os_teleop

Dieses Paket ermöglicht das Ansteuern des (Roboter-) Motors per Joy-Pad oder Tastatur.

# 3 Problemstellung

## 3.1 Linux & ROS

Bei unseren ersten Startversuchen der Lösung des vorherigen Semesters wurde schnell deutlich, dass sich der Einstieg für unerfahrene Entwickler (in Linux und ROS) als recht umständlich gestaltet. Die Komplexität der ROS-Laufzeitumgebung (z.B. roscore, rviz, rjoy müssen separat gestartet werden, oder per Script) ist die größte Hürde, die es anfänglich zu überwinden gilt.

Unter anderem konnten folgende Fragen nicht auf Anhieb geklärt werden: Wo liegt ROS auf dem System? Welche ROS-Pakete werden verwendet? Wie startet man die Steuerung für den Roboter? Wie startet man die Kartierung? Wo wird die erzeugte Karte gespeichert? u.v.a.m.

Eine zentrale Verwaltung der Software ist nicht vorhanden. Die bestehende Dokumentation erwies sich im Umfang als unzureichend um einen einfachen und schnellen Einstieg ins Projekt zu ermöglichen. Desweiteren war z.B. das ROS-Paket zur Steuerung per Joystick fehlerhaft installiert und konnte nicht korrekt gestartet werden. Dieses mussten wir zur Einarbeitung und zum erstmaligen Testen neu installieren und konfigurieren.

## 3.2 Kartierungsprobleme

Eine 2D Kartierung der für uns Studenten frei zugänglichen Räume im 2.OG des A-Gebäudes der HS-Mannheim gestaltet sich einzeln betrachtet schon als schwierige Aufgabe. Die Positionsbestimmung des Roboters und damit

die Wahrheitstreue der Kartierung steht und fällt mir der Präzision der Odometrie. Fährt der Roboter z.B. zu schnell über Boden- und Türschwellen und bietet somit einen zu großen Schlupf für die Antriebsräder, entstehen "Knicke" in geraden Fluren und Räume werden verzerrt. Dieses Verhalten bezeichnet man als Odometrie Drift. Die erzeugte Karte entspricht letztendlich nicht mehr der Realität und muss über Optimierungsverfahren bereinigt und verbessert werden. Dies könnte z.B. durch erneutes (teilsweise) Durchführen der Kartierung geschehen mit abschließenden Zusammenführen der alten und der neu erzeugten Karte über ICP[1]. Aus dem gleichen Grund gilt es den Kontakt mit Hindernissen wie Tischen und Stühlen zu vermeiden, um die Odometrie nicht weiter zu verfälschen. Besagte Tische, Stühle, Kabel und Rechner erkennt man teilweise auf der erzeugten Karte, da der Laser diese natürlich detektiert. Zu Anfang der Kartierung entstehen Zickzackmuster, wenn der Laser den Wald aus Hindernissen unter den Tischen abtastet. Erst wenn der Roboter komplett um ein Hindernis herum gefahren ist und den Raum von allen Seiten kartiert hat, sind die Hindernisse als solches zu erkennen.

## 3.3 Autonomes Fahren

Zur Zeit wäre es nicht verantwortbar den Roboter komplett ohne Aufsicht herumfahren zu lassen. Das autonome Fahren kann zwar über das ROS-Paket navigation realisiert werden, aber bevor dieses genutzt werden kann, muss die Hardware am Roboter noch überarbeitet werden. Z.B. können gefährliche Treppenabsätze nicht erkannt werden, da der Laser-Scanner nur horizontal in einer Höhe von ca 25cm misst. Hier könnten zukünftige Arbeiten anknüpfen um z.B. die Kinect zur Standortbestimmung und Umgebungserkennung ins System mit einzubinden. Hierbei könnte man direkt auch eine 3D-Karte erzeugen und diese mithilfe der Laser-Scanner Daten überprüfen und berichtigen. Auch werden Glasfächen wie Fenster und Türen nicht erkannt, da der Laser die Transparenz nicht erkennt.

[1] ICP (Iterative Closest Point) Algorithmus zum Zusammenführen von Punkt Wolken

# 4 Realisierung

In diesem Kapitel geht um die eigentliche Realisierung unseres Vorhabens. Nachdem auf die Auswahl der Technologien eingangen wurde, werden die wichtigsten ROS-Pakete beschrieben, die bei der Umsetzung verwendet wurden. Anschließend werden diese Pakete in der Architektur und deren Kommunikation untereinander veranschaulicht.

## 4.1 Auswahl der Entwicklungsumgebung

Als Basissystem wurde ROS gewählt. ROS ist einer der ersten Ansätze ein Betriebssystem nur für Roboter zu entwickeln. Die aktuelle, erste Version befindet sich noch in einem frühen Stadium wenn man die Tools und Pakete die aktuell verfügbar sind mit denen anderen Fachdomänen (Web-/Appentwicklung, Enterprise, IoT) vergleicht. Außerdem wurde am Institut für Robtik an der Hochschule Mannheim schon einige Erfahrung damit gesammelt. Als Host-Betriebssystem entschieden wir uns für Ubuntu 14.04 LTS mit dem ros-indigo-igloo Paket welches sich schnell und zuverlässig über die systemeigene Paketverwaltung apt-get installieren lässt.

Unser Ansatz Docker für unsere Anforderungen zu verwenden entstand vor allem aus der Not heraus, das wir mit der bestehenden Softwareumgebung auf unserem Bot nicht so wirklich zurecht kamen. Wir haben Fehler erhalten und wussten nicht ob die Software oder die Hardware Fehler verursacht da nicht nachvollziehbar war was aktuell an Pakten auf dem System installiert ist und wie diese konfiguriert waren.

Um zu gewährleisten, dass das unser neues System transparent und einfach zu verstehen ist, entschieden wir uns beim Aufbau des Systemes dazu Docker zu verwenden. Es bietet die Möglichkeit ein komplexes System

aus vielen Komponenten zu strukturieren und in Container aufzuteilen. Durch die genaue Abgrenzung bietet Docker auch einen einfacheren Einstieg in das unbekannte System. Mithilfe der Konfigurationsdateien kann jeder Entwickler sofort sehen wie die Anwendung strukturiert ist. Außerdem müssen die verschieden Komponenten nicht mehr per Hand aufgerufen werden und lassen sich zusätzlich noch flexibel miteinander kombinieren. Dadurch soll der Einsteig, vor allem für Neulinge, erheblich vereinfacht werden.

## 4.2 Architektur

Die Abbildung 4.1 zeigt die Kommunikation der verwendeten ROS-Pakete zur Laufzeit. ROS-Nodes werden hier durch Kasten dargestellt, ROS-Topics stehen als Beschriftung an den Richtungspfeilen.

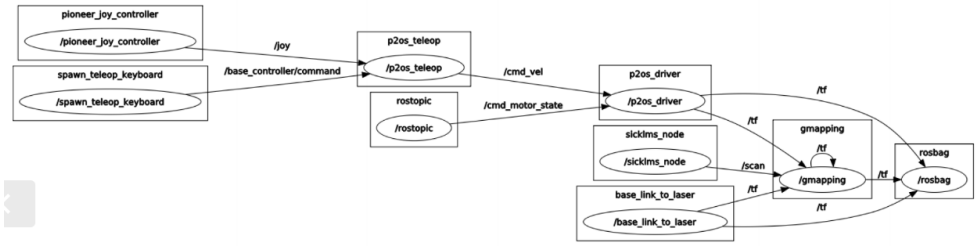


Abbildung 4.1: Übersicht der Kommunikation zwischen verwendeten Nodes zur Laufzeit.

## 4.3 Steuerung des Roboters

Zur manuellen Steuerung des Roboters per Joystick (pioneer_joy_controller) und Tastatur (spawn_teleop_keyboard), werden Eingaben des Kontrollers über die ROS-Node p2os_teleop registriert. Im p2os_driver läuft die Eingabeabfrage (p2os_teleop) und die eigentliche Motor-Steuerung des Roboters rostopic/cmd_motor_state zusammen und ermöglicht so die manuelle Steuerung.

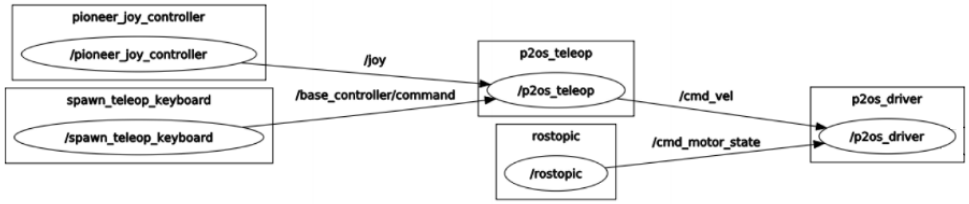


Abbildung 4.2: Übersicht der Kommunikation zwischen verwendeten Nodes zur Laufzeit für die manuelle Steuerung.

## 4.4 Kartierung

Die Kartierung wird über das ROS-Paket gmapping realisiert. Hier fließen alle Informationen aus der Steuerung (p2os_driver) und den Laser-Scanner Daten (sicklms_node und base_link_to_laser) zusammen und eine Karte wird iterativ erzeugt. Alle Rohdaten, sowie die erzeugte Karte werden im rosbag zur Präsentation oder späteren Verarbeitung gespeichert. Das ROS-Paket tf (Transformation) übernimmt die nötigen Transformationen der verschiedenen Koordinatensysteme der verschiedenen Software- und Hardwarekomponenten.

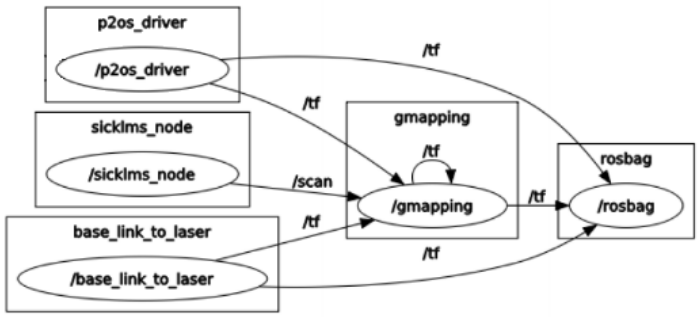


Abbildung 4.3: Übersicht Kommunikation zwischen verwendeten Nodes zur Laufzeit für die Kartierung.

## 4.5 WLAN Heatmapping

Zur Erzeugung der WLAN-Heatmap wird das oben vorgestellte Paket heatmap genutzt. Der Nachteil hier ist die Kartierung der Stockwerke zweimal durchgeführt werden müssen, da heatmap eine bestehende Karte zur Lokalisierung mit amcl nutzt. Allerdings sollte es möglich sein das Paket so zu modifizieren, so dass es sich die Kartierungsdaten direkt aus dem Navigation-Stack zieht und dadurch beide Schritte ein einem umgesetzt werden können. Ebenso kann die WLAN Kartierung noch nicht ohne größere Gefahren autonom durchgeführt werden (siehe Kapitel 3) und muss daher ständig beaufsichtigt werden .

# 5 Tests und Ergebnisse

Die Kartierung wurde erfolgreich getestet. Leider haben wir es nicht mehr geschafft auch das WLAN-Heatmapping vollständig zu integrieren. Allerdings stellt unsere Lösung nun einen guten Stand dar um 2D-Karten zu erfassen und auszuwerten. Es gab aus zeitlichen Gründen nur einen *Feldest* mit Akkus, dieser konnte nicht das gesamte Stockwerk überdauern. Andere Gruppen kamen hierbei zu anderen Ergebnissen, so dass wir davon ausgehen eher die älteren Akkus geladen zu haben. Mit neuen, voll geladenen Akkus sollte es möglich sein zumindest ein Stockwerk in einem Durchlauf zu erfassen.

Des Weiteren wurde getestet ob per eingerichteten, lokalen AccessPoint auf dem Roboter eine Live-Übertragung der Kartierungsdaten machbar ist. Unsere Beobachtung war allerdings, dass wir mit einem einfachen Netzwerk-Kabel deutlich performantere Ergebnisse erhielten. Das WLAN war teilweise instabil und hat vor allem in den grafischen Tools zu abstürzen und Time-Outs geführt. Eventuell lässt sich dieses Problem durch robustere, leistungsoptimierte WLAN-Umgebungen oder andere Kommunikationsverfahren beheben. Eine alternative um dieses Problem zu umgehen ist es dem rosbag zu nutzen und diesen später lokal an der Workstation wieder während der Laufzeit auszupacken. Ebenso könnte man versuchen die Workstation und den Roboter über das HS-MA Netzwerk kommunizieren zu lassen (Freischaltung benötigt).

Die WLAN-Feldstärken Kartierung wurde bisher erfolgreich nur in Simulationen getestet. Das Ziel lag vorerst darin sicherzustellen das der Roboter jedes verfügbare Netzwerk aufzeichnet, bevor dieser einem größeren Kartierungstest unterzogen wird. Daher wurde ein Fritz WLAN-Stick verwendet, da dieser beide Frequenzbänder verarbeiten kann. Allerdings wird das parallele Aufzeichnen von allen erreichbaren AccessPoints von dem von uns

gewählten Paket so nicht unterstützt. Es empfiehlt sich also für zukünftige Arbeiten das Paket entsprechend zu modifizieren oder eine eigene, unabhängige Lösung zu entwickeln. Da unsere Lösung die Kartierung soweit beherrscht kann ggf. auch in Erwägung gezogen werden die WLAN-Feldstärke extra aufzuzeichen, außerhalb von ROS. Wobei natürlich eine ROS-Lösung aufgrund der Kompatibilität bevorzugt werden sollte.

# 6 Ausblick

In Zukunft bietet der bisherige Stand die Möglichkeit das autonome kartieren der WLAN-Feldstärke sauber und modular umzusetzen. Dadurch könnte der Roboter selbstständig und ohne administrative Aufsicht seine Arbeit erledigen. Dabei beachten sollte man allerdings vor allem Treppen und andere gefährlich hohe Absätze. Unsere aktuell Lösung erkennt solche Gefahren nicht. Problematisch hierbei ist hauptsächlich das Risiko Hardware-Komponenten zu beschädigen oder gar den ganzen Aufbau zu zerstören. Dies kann eventuell durch Beacons[1], Aktive Erkennung (z.B. durch die Kinect) oder per QR-Codes[2] an den Gefahrenstellen gelöst werden.

Anhand der WLAN-Heatmap können die optimalen Positionen der Access Points bestimmt werden, dies würde es dem Rechenzentrum der Hochschule stark vereinfachen WLAN-Löcher zu erkennen und effektiv zu beseitigen.

Außerdem sollte durch diese Arbeit der Grundstein für eine saubere und einfache Einarbeitung in die Thematik geschaffen sein. Wer die Grundlagen (insbesondere die Grundlagen-Tutorials) von ROS verinnerlicht hat, sollte mit unserem Aufbau ohne großen Aufwand schnell in der Lage dazu sein die bestehende Lösung zu optimieren und zu verwalten.

[1] Beacon: Referenzobjekt, das z.B. per Bluetooth erkannt werden kann.
[2] Scanbarer 2D Code

# 7 Zusammenfassung

Zusammengefasst lässt sich sagen das ROS in Kombination mit Docker eine zufriedenstellende Lösung für unseren Anwendungszweck bietet. Wir haben es geschafft ein modulares, transparentes System zu bauen, welches unseren Anforderungen gerecht wird und einen einfachen Einsteig für Projektneulinge bieten kann. Die etwas höhere Einarbeitungszeit durch Docker wird unserer Meinung nach dadruch kompensiert, dass dem Entwickler anschließend eine solide Umgebung zur Verfügung steht, die er nach belieben modifizieren und erweitern kann. Man muss sich außerdem keine Gedanken mehr darüber machen etwas am Hostsystem kaputt zu machen da die Container jederzeit neu geladen werden können. So bekommt der Entwickler gleichzeitig ein hoch verfügbares Test- und Stagingsytem, welches gleichzeitig natürlich auch für den Produktivbetrieb genutzt werden kann.

Unsere Lösung übernimmt sämtliche Aufgaben die zur 2D-Kartierung mithilfe des Pioneers benötigt werden. Damit haben wir eine optimale Grundlage für weitere Erweiterungen und Experimente geschaffen. Dies ist unserer Meinung nach ein besonders großer Vorteil, da hierdurch das aufwenidige Installieren und Konfigurieren eines eingenen Entwicklungssystems wegfällt. Dadruch wird die Möglichkeit geschaffen während laufenden Vorhaben mehr Zeit für andere Aktivitäten zu haben.

Auch wenn wir unsere ursprüngliche Aufgabe, das Kartierung der WLAN-Verfügbarkeit nicht mehr realisieren konnten, nicht mehr implementieren konnten haben wir einige Sachverhalte schonmal vorab geklärt und untersucht. Wir hoffen deshalb das die vollstädnige Umsetzung der WLAN-Analyse in zukünftigen Arbeiten erfolgreich zu Ende gebracht werden kann.

# 8 Literaturverzeichnis


[ROS, 2016] "Enzyklopädie über ROS", http://wiki.ros.org/, 18.07.2016.

[WPI, 2014] "PARbot A Personal Assistive Robot", https://www.wpi.edu/Pubs/E-project/Available/E-project-043014-173250/unrestricted/PARbot_MQP_Final_Report.pdf, 18.07.2016.

[Dynapar, 2016] "What is an Optical Encoder?", http://www.dynapar.com/technology/optical-encoders/, 18.07.2016.

[Adept Mobile Robots, 2011] "Datasheet Pioneer 3-DX", www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx, 18.07.2016.

[Wikipedia, 2016] "Inkrementalgeber", https://de.wikipedia.org/wiki/Inkrementalgeber, 18.07.2016.

[Wikipedia, 2016] "Docker ", https://de.wikipedia.org/wiki/Docker_(Software), 19.07.2016.

[Docker, 2016] "Build, Ship, Run. An open platform for distributed applications for developers and sysadmins ", https://www.docker.com/, 19.07.2016.

[Docker Documentation, 2016] "the Docker Documentation", https://docs.docker.com/, 19.07.2016.

# 8 Literaturverzeichnis


[AppArmor, 2016] "AppArmor is an effective and easy-to-use Linux application security system. ", http://wiki.apparmor.net/index.php/Main_Page, 19.07.2016.

[UbuntuUsers, 2016] "Ständig wachsende Sammlung von Anleitungen und Problemlösungen für Ubuntu und Linux", https://wiki.ubuntuusers.de/Startseite/, 19.07.2016.

[Docker Github, 2016] "Repository of Docker the container engine", https://github.com/docker/docker, 19.07.2016.

[Sick LMS 200, 2007] "Technical Description of the LMS200/211/221/291", http://sicktoolbox.sourceforge.net/docs/sick-lms-technical-description.pdf, 19.07.2016.

[Universal Lexikon, 2014] "die Bestimmung des Abstandes zwischen zwei Punkten an der Erdoberfläche oder im Raum durch mechanische, optische oder elektronische Messverfahren.",http://universal_lexikon.deacademic.com/234125/Entfernungsmessung, 20.07.2016.